Patrick Knöbelreiter

Deep Learning with Structured Models

DOCTORAL THESIS

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Thomas Pock
Institute for Computer Graphics and Vision
Graz University of Technology, Austria

Prof. Dr.-Ing. Andreas Geiger
Department of Computer Science
University of Tübingen, Germany

Graz, Austria, October 2020

To my family.

The more I see, the less I know for sure.

John Lennon (1940 - 1980)

Abstract

Depth reconstruction is a fundamental problem in many computer vision applications. Although calculating depth is one of the oldest and most researched topics in computer vision, there are still many challenges to be resolved. Repetitive structures, different lighting conditions, reflections, oversaturated and undersaturated image areas, occlusions etc. make the stereo problem even more difficult. The main goal of developing novel stereo methods is to be robust against these challenges. Current research mainly focuses on Deep Learning (DL) approaches to automatically learn robust models. However, these models require a large amount of training data and often have difficulty in generalizing to new scenes.

In this work, we propose new structured models that consist of Convolutional Neural Networks (CNNs) together with Conditional Random Fields (CRFs) and Variational Networks (VNs) coming from discrete or continuous optimization. This enables us to get the best of both worlds: the power and expressiveness of *DL* as well as the structure and interpretability of the optimization. We use the two opposite methods specifically where they work best. For this purpose, we learn on the one hand optimal features for matching with a *CNN*, since it is unclear how optimal features can be hand-crafted. On the other hand, we use optimization to bring prior knowledge into our model and generalize it with modern, learnable elements. This allows us to explicitly prefer slanted surfaces and add global information to the problem, which is not easy with traditional *CNNs*.

In order to demonstrate the applicability and robustness of the proposed *CNN + CRF* models, we apply them to the stereo task. The experiments show that these hybrid models make it possible to significantly reduce the number of learnable parameters while at the same time achieving excellent performance in the stereo benchmarks. Of particular note are the results of the high-resolution Middlebury 2014 benchmark. Our *CNN + CRF* method is one of the few *DL*-based methods that can be used for high-resolution images and also has an excellent ratio of performance and runtime.

Since the *CNN + CRF* models naturally lead to a discrete disparity map, we propose a refinement module with a hybrid *CNN + VN* model. This makes it possible to learn a higher-order regularizer specifically for the stereo task. Our experiments show that our proposed module can significantly

reduce discretization artifacts, reduce artifacts in occlusions and, as a result, deliver the desired sub-pixel accurate disparity maps.

We furthermore tackle the problem of acquiring ground-truth data for stereo. Due to the geometric properties of the stereo problem, it is not possible to create this data manually. Instead, expensive hardware is required. To overcome this limitation, we propose a method based on self-learning to automatically generate training data for stereo. Our experiments show that we can not only successfully train *DL*-based models with our generated training data, but also significantly improve the performance of the resulting models.

Keywords: Stereo, CNN+CRF, VN, Optimization, Deep Learning, Self-Learning

Kurzfassung

Tiefeninformation ist ein wichtiger Bestandteil für viele Computer-Vision-Anwendungen. Obwohl die Berechnung von Tiefe eines der ältesten und am meisten erforschten Themen in der Bildverarbeitung ist, gibt es nach wie vor viele Herausforderungen zu lösen. Repetitive Strukturen, unterschiedliche Lichtverhältnisse, Reflexionen, über- und untergesättigte Bildbereiche, Verdeckungen usw. machen das Stereoproblem noch schwieriger. Das Hauptziel der Entwicklung neuartiger Stereomethoden ist es, gegenüber diesen Herausforderungen robust zu sein. Die aktuelle Forschung konzentriert sich hauptsächlich auf *DL*-Ansätze, um automatisch robuste Modelle zu lernen. Diese Modelle benötigen jedoch eine große Menge an Trainingsdaten und haben häufig Schwierigkeiten, auf neue Szenen zu generalisieren.

In dieser Arbeit schlagen wir Hybridmodelle vor, die aus *CNNs* zusammen mit *CRFs* und *VNs* bestehen und aus der diskreten oder kontinuierlichen Optimierung stammen. Dies ermöglicht es uns, das Beste aus beiden Welten herauszuholen: die Ausdruckskraft von *DL* sowie die Struktur und Interpretierbarkeit der Optimierung. Wir verwenden die beiden entgegengesetzten Methoden speziell dort, wo sie am besten funktionieren. Zu diesem Zweck lernen wir einerseits optimale Features für den Vergleich mit einem *CNN*, da unklar ist, wie optimale Features von Hand erzeugt werden können. Andererseits verwenden wir die Optimierung, um Vorwissen in unser Modell einzubringen und sie mit modernen, lernbaren Elementen zu verallgemeinern. Dies ermöglicht es uns, geneigte Oberflächen explizit zu bevorzugen und dem Problem globale Informationen hinzuzufügen, was mit herkömmlichen *CNNs* nicht einfach möglich ist.

Um die Anwendbarkeit und Robustheit der vorgeschlagenen *CNN* + *CRF*-Modelle zu demonstrieren, wenden wir sie auf das Stereoproblem an. Unsere Experimente zeigen, dass es diese Hybridmodelle ermöglichen, die Anzahl der lernbaren Parameter signifikant zu reduzieren und gleichzeitig eine hervorragende Leistung bei den Stereo-Benchmarks zu erzielen. Besonders hervorzuheben sind die Ergebnisse auf dem hochauflösenden Middlebury 2014-Benchmark. Unsere *CNN* + *CRF*-Methode ist eine der wenigen *DL*-basierten Methoden, die für hochauflösende Bilder verwendet werden können und außerdem ein hervorragendes Verhältnis von Leistung und

Laufzeit aufweisen.

Da die *CNN + CRF* Modelle natürlicherweise zu einer diskreten Disparitätskarte führen, schlagen wir ein Verfeinerungsmodul mit einem hybriden *CNN + VN* Modell vor. Dies ermöglicht es, einen Regularisierer höherer Ordnung speziell für das Stereoproblem zu lernen. Unsere Experimente zeigen, dass unser vorgeschlagenes Modul Diskretisierungs- und Verdeckungen-Artefakte signifikant reduzieren und als Ergebnis die gewünschten subpixelgenauen Disparitätskarten liefern kann.

Zusätzlich beschäftigen wir uns mit dem Problem der Erfassung von Referenzdaten für Stereo. Aufgrund der geometrischen Eigenschaften des Stereoproblems ist es nicht möglich, diese Daten manuell zu erstellen. Stattdessen ist teure Hardware erforderlich. Um diese Einschränkung zu überwinden, schlagen wir eine Methode vor, die auf dem Selbstlernen basiert, um automatisch Referenzdaten für Stereo zu generieren. Unsere Experimente zeigen nicht nur, dass *DL*-basierte Modelle nicht nur mit unseren generierten Referenzdaten erfolgreich trainiert werden können, sondern auch, dass die Leistung der resultierenden Modelle erheblich verbessert wird.

Keywords: Stereo, CNN+CRF, VN, Optimierung, Deep Learning, Self-Learning

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

Doing a PhD at the Institute for Computer Graphics and Vision with Prof. Pock was an excellent decision. It enabled me to learn and understand algorithms in computer vision, optimization, and machine learning in depth. Pursuing the PhD was often hard work, but it allowed me to meet great people with similar interests, discuss new ideas, and work closely together to push the limit of what is possible by a little bit. Therefore, I want to take this opportunity to thank the people who made my days great days.

First and foremost, I would like to thank my supervisor Prof. Pock. Even before I started my PhD, I was intrigued by his lectures on computer vision and optimization and the elegance of his approach to complex problems. He always guided me to the right directions during my PhD, challenged me to get the best out of myself, and inspired me with lots of great ideas and the way he approaches challenges. Thank you very much for giving me the time to develop myself further in research and teaching. I would also like to thank my second superior Andreas Geiger.

Many thanks also to all current and former members of the VLO group. I would particularly like to thank Alexander Shekhovtsov for all the insightful discussions, especially about graphical models and discrete optimization algorithms, where he is for sure a leading expert on. I would also like to thank my close collaborators Christoph Vogel, Christian Reinbacher, Gottfried Munda and Christian Sormann. Thank you for all the fruitful discussions and the countless hours we spent together as a deadline approached.

I would also like to thank Christian Reinbacher and Dominik Narnhofer, with whom I shared an office at the institute. Thank you guys for listening to me and for discussing about both research and the little everyday things. It was a great time with you. I also thank Alexander Effland and Joana Grah for proofreading this thesis.

None of this would have been possible without my family. Special thanks go to my parents who made my studies possible, helped me pursue my goals, and supported me and my decisions in everything I do. Without you I wouldn't be where I am now! Finally, I would like to thank Tanja for her support in every respect. Thank you for listening to me and my problems and for saving many days with your smile.

Thank you!

Contents

1 Introduction **1**

1.1 Machine Learning and Optimization – A Symbiosis . . . 2
1.2 Stereo . . . 2
1.3 Contribution and Outline . . . 7
1.4 Notation . . . 9

2 Related work **11**

2.1 Probability Theory . . . 11
2.1.1 Probabilities and Random Variables . . . 11
2.1.2 Probability Calculus . . . 11
2.1.3 Decision Theory . . . 12
2.2 Mathematical Preliminaries . . . 13
2.2.1 Inner Product . . . 13
2.2.2 Vector Norms . . . 14
2.2.3 Matrix Norm . . . 15
2.2.4 Adjoint Operator . . . 17
2.2.5 Properties of functions . . . 17
2.3 Optimization problems in Computer Vision and Machine Learning . . . 17
2.4 Discrete Optimization . . . 18
2.4.1 Markov Random Fields . . . 18
2.4.2 Conditional Random Fields . . . 21
2.4.3 Energy Minimization for MRF/CRF . . . 23
2.4.4 Inference Algorithms . . . 25
2.4.4.1 Dynamic Programming . . . 25
2.4.4.2 (Loopy) Belief Propagation . . . 28
2.4.4.3 Dual Decomposition . . . 41

2.4.4.4 Summary and Further Reading . . . 44
2.5 Continuous Optimization . . . 45
2.5.1 Convex Analysis . . . 45
2.5.1.1 Convex Sets . . . 45
2.5.1.2 Convex Functions . . . 46
2.5.1.3 Subdifferential . . . 47
2.5.1.4 Convex Conjugate . . . 48
2.5.1.5 Proximal Operator . . . 50
2.5.2 Convex Optimization . . . 51
2.5.2.1 Gradient Methods . . . 52
2.5.2.2 Proximal Methods . . . 54
2.5.2.3 Primal-Dual Method . . . 56
2.5.3 Variational Methods for Stereo . . . 57
2.6 Machine Learning . . . 59
2.6.1 Artificial Intelligence . . . 59
2.6.2 Machine Learning . . . 60
2.6.3 Supervised Learning . . . 61
2.6.3.1 Classification . . . 61
2.6.3.2 Regression . . . 63
2.6.4 Self-Supervised Learning . . . 63
2.6.5 Unsupervised Learning . . . 65
2.7 Deep Learning with Neural Networks . . . 65
2.7.1 Perceptron . . . 65
2.7.2 Fully Connected Neural Networks . . . 67
2.7.3 Activation Functions . . . 68
2.7.4 Convolutional Neural Networks . . . 71
2.7.5 Model Training . . . 72

3 Hybrid CNN-CRF Model for Stereo 81
3.1 Introduction . . . 82
3.2 Related Work . . . 84
3.3 CNN-CRF Model . . . 85
3.3.1 Unary CNN . . . 85
3.3.2 Correlation . . . 85
3.3.3 CRF . . . 86
3.3.4 Pairwise CNN . . . 87
3.4 Training . . . 87
3.4.1 Training Unary CNN in the Pixel-wise Model . . . 88
3.4.2 Training Joint Model . . . 89
3.4.3 Training Unary and Pairwise CNNs in Joint Model . . . 92
3.4.4 Implementation Details . . . 93

3.4.5 Training insights . . . 93
3.5 Experiments . . . 94
3.5.1 Benchmark Data Sets . . . 94
3.5.2 Performance of Individual Components . . . 94
3.5.3 Benefits of Joint Training . . . 94
3.5.4 Benchmark Test Performance . . . 96
3.5.5 Additional Experiments . . . 97
3.5.5.1 Sublabel Enhancement . . . 97
3.5.5.2 Middlebury Stereo v3 . . . 98
3.5.5.3 Kitti 2015 . . . 99
3.5.6 Timing . . . 101
3.6 Conclusion . . . 101

4 Belief Propagation Reloaded **105**
4.1 Introduction . . . 106
4.2 Related Work . . . 107
4.3 Belief Propagation . . . 108
4.4 Sweep BP-Layer . . . 110
4.4.1 Sweep BP as Dynamic Programming . . . 111
4.4.2 Other Inference Methods . . . 113
4.4.2.1 SGM . . . 114
4.4.2.2 Tree-structured DP . . . 114
4.4.2.3 TRW and TBCA . . . 114
4.5 Models . . . 116
4.5.1 Stereo . . . 117
4.5.2 Optical Flow . . . 117
4.5.3 Semantic Segmentation . . . 118
4.6 Learning . . . 118
4.7 Implementation Details . . . 119
4.7.1 Runtime Analysis . . . 119
4.7.2 Model Architecture . . . 120
4.8 Experiments . . . 121
4.8.1 Improvements brought by the BP-Layer . . . 121
4.8.2 Stereo Benchmark Performance . . . 123
4.8.3 More stereo experiments . . . 124
4.8.4 Optical Flow . . . 127
4.8.5 Semantic Segmentation . . . 129
4.9 Conclusion . . . 133

5 Learned Continuous Disparity Refinement **135**
5.1 Introduction . . . 136
5.2 Related Work . . . 137
5.3 Method . . . 139
5.4 Computing Inputs . . . 143
5.5 Learning . . . 145
5.6 Experiments . . . 146
5.6.1 Ablation Study . . . 147
5.6.2 Benchmark Performance . . . 153
5.7 Analyzing the VN . . . 153
5.7.1 Learned Filters and Activation Functions . . . 153
5.7.2 Shared Parameters . . . 155
5.7.3 VN Color Image . . . 157
5.7.4 VN Confidences . . . 158
5.8 Conclusion & Future Work . . . 160

6 Self-learning for Stereo **161**
6.1 Introduction . . . 162
6.2 Related Work . . . 163
6.3 Self-Supervised Dense Matching . . . 164
6.4 Experiments . . . 166
6.4.1 Vaihingen Dataset . . . 167
6.4.2 Cityscapes Dataset . . . 170
6.5 Conclusion & Future Work . . . 171

7 Summary and Outlook **173**
7.1 Summary . . . 173
7.2 Conclusion & Outlook . . . 174

A List of Acronyms **177**

B List of Publications **179**

Bibliography **185**

1 Introduction

Within the last years, Machine Learning (ML) and especially Deep Learning (DL) have become the most successful tools in computer vision. Many ideas, including Artificial Neural Networks (ANNs), the back propagation algorithm [105], *etc.*, have been invented already decades ago. At that time the training of *ANNs* was very tedious due to the limited computational power and the learned models generalized poorly [26]. However, the renaissance started in 2012 when these ideas have been successfully applied to large-scale problems in practice. Back then, Krizhevsky et al. [100] paved the path to large-scale *DL* by exploiting the Graphics Processing Unit (GPU) for training deep Convolutional Neural Networks (CNNs) stochastically. The stochastic training and the additional power and parallelization on *GPUs* allowed for the first time to train on huge datasets including billions of images [42]. The massive amount of training data together with the expressiveness of *CNNs* yielded a significant performance gain on the image classification task without the severe overfitting problem. Thenceforth, the whole field of computer vision moved towards using deep *CNNs* and pushed the performance to previously unimaginable areas. This is also reflected in many computer vision benchmarks, where models using *CNNs* outperform classical, non-learned models by a significant margin. For example, the best performing method on the Kitti Stereo benchmark [132], which does not use a *CNN* at all [200], is only ranked at position 199 (!).[1] This shows impressively that *DL* is an irreplaceable tool in modern computer vision. Theoretically, neural networks are universal function approximators [41, 228]. Although it is tempting to let the model learn a specific task directly, it is often a bad and naive idea. Instead, it turns out that it is advantageous to explicitly incorporate *structure* into the model. In this thesis, we provide structure to our models through optimization. This allows to explicitly incorporate prior knowledge about specific tasks, which does not have to be additionally learned by the model. Therefore, as a result we often get more lightweight models which are i) faster to train and ii) often generalize better to previously unseen data.

[1] `http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo` accessed on 17th July, 2020.

1.1 Machine Learning and Optimization – A Symbiosis

ML and optimization are tightly connected to each other. Many *ML* problems are a (large-) scale optimization problem in their core. Efficient optimization algorithms are therefore required in order to find optimal parameters for the *ML* models. We will see this connection in Section 2.5, where we are introducing important optimization algorithms. Although the algorithms presented come from the literature on convex optimization [5, 140], these algorithms are also frequently and successfully used to train machine learning models. However, *ML* and optimization are not the same as also argued by Bennett and Parrado-Hernández [9]. They rather form a symbiosis. *ML* research focuses on designing new models for specific tasks using *e.g.* tools from statistics and probability theory [64]. There, the optimization algorithms are viewed as a tool that should be easy to use. In difference, the focus in optimization is more on efficiency, speed in terms of convergence properties, robustness and theoretical guarantees without having a specific target application. This symbiosis between *ML* and optimization is also evident in research, where *ML* triggers optimization research and vice versa. On the one hand, new results from the optimization literature are often used in *ML*, because they enable to train *e.g.* more complex models efficiently, which was not possible before. On the other hand, *ML* also triggers optimization research. An important example is the training of deep *CNNs*, which poses a large-scale, nonlinear and non-convex optimization problem. Theoretical results from the literature on convex optimization are no longer valid and thus new theoretical results are required.

In this work, we show how ideas originating from optimization can be used to introduce structures into *ML* models. We investigate and propose methods to seamlessly integrate Conditional Random Field (CRF) into *DL* models in Chapters 3 and 4. This enables us to explicitly incorporate global structure in terms of pixel neighborhoods and leads to a global optimization problem. We integrate structure into our *ML* models by interpreting the layers of a CNN as the iterations of an optimization algorithm, too. This technique is called *algorithm unrolling* and guides the architecture of the resulting *ML* model. We successfully use this technique in Chapter 5 and show that we can gain interpretability of what the model learns. We explicitly exploit the provided efficiency by the optimization algorithm and transfer it to our *ML* models in both variants. Furthermore, we can also build up on mathematically sound theoretical results which yield *ML* models designed in a principled way. Therefore, optimization is not only a necessary tool for training *ML* models, but also a principled tool for designing new *DL* models.

1.2 Stereo

Solving the stereo problem is the main task we want to tackle with the methods presented in this thesis. Therein, the goal is to compute depth information from images. We will mainly focus on the canonical stereo problem, where two images are captured at the same time, but from slightly different viewpoints. This is different to the Multi-View-Stereo (MVS) problem where we have more than two views possibly captured at different times, which we will only tackle briefly in Chapter 6.

Figure 1.1: Stereo matching on rectified images. The left image shows the reference image together with a reference point/patch. The task is to find the corresponding point in the right image on the epipolar line. The green line depicts the epipolar line, which corresponds to an image row.

Let us now describe the (canonical) stereo problem formally. To this end, we consider images to be continuous functions $I : \Omega \to \mathbb{R}^C$, where $\Omega \subset \mathbb{R}^2_+$ is the domain of our images (usually a rectangle) and C is the number of channels being 1 for gray-scale images and 3 for Red-Green-Blue (RGB) color images. Note that we can always convert discrete images to continuous images by interpolation and vice versa by sampling. Then, given two rectified images $I_0, I_1 \colon \Omega \to \mathbb{R}^C$ from a calibrated camera pair, the goal is to find the displacements $d \colon \Omega \to \mathbb{R}^2$, such that

$$I_0(\mathbf{x}) = I_1(\mathbf{x} - d(\mathbf{x})), \tag{1.1}$$

where $\mathbf{x} \in \Omega$ are the pixel coordinates and $d(\mathbf{x}) = (d_1(\mathbf{x}), d_2(\mathbf{x}))^\top$. In the rectified stereo setup d_1 are the sought horizontal displacements which are referred to as *disparities* and measured in pixels. Due to the rectification and the epipolar constraint we have $d_2(\mathbf{x}) = 0$ everywhere.

In the following we will review and discuss the generic *stereo taxonomy* by Scharstein and Szeliski [171]. It consists of the four steps

1. Matching cost computation,
2. Cost (support) aggregation,
3. Disparity computation / Optimization,
4. Disparity refinement.

Additionally, we will also discuss possible features which can be used for the matching cost computation.

Matching Cost Computation Equation (1.1) is also known as the Brightness Constancy Assumption (BCA) [72]. Therein, the central assumption is that the brightness (=intensity) of a pixel remains the same when viewed from different cameras. In order to find pixels fulfilling the *BCA*,

we need to solve the pixel-wise optimization problem

$$\min_{d} \int_{\Omega} f\big(I_0(\mathbf{x}), I_1(\mathbf{x} - d(\mathbf{x}))\big) d\mathbf{x}, \tag{1.2}$$

where the matching cost function $f\colon \Omega^2 \to \mathbb{R}$ measures the similarity between a pixel in the reference image I_0 and the second image I_1, respectively. Hence, we want to find for every pixel position $\mathbf{x}$ the displacement $d(\mathbf{x})$ yielding the minimal matching cost or equivalently the maximal matching similarity. Equation (1.2) is also known as the *correspondence problem*. Note that Eq. (1.2) decouples over the individual pixels. A visualization explaining Eq. (1.2) is shown in Fig. 1.1. To this end, let us next review classical matching cost functions. Since pixel-wise matching is not robust we define the matching cost functions on patches. For the sake of simplicity, we assume here patches of a are discrete image which can be represented by a matrix of size $M \times N$ or vector of size MN, respectively. Let us denote a patch in the reference image centered at position $\mathbf{x}$ with $\mathbf{p} = (p_1, \dots, p_{MN})$ and a patch in the second image centered at the shifted position $\mathbf{x} - d(\mathbf{x})$ with $\mathbf{q} = (q_1, \dots, q_{MN})$. One of the most often used matching cost function are the Sum of Squared Differencess (SSDs) defined as

$$f_{SSD}(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_2^2 \tag{1.3}$$

where $\|\cdot\|_2^2$ is the squared ℓ_2-norm (see Section 2.2.2). This is probably the simplest matching cost and works very well in many conditions. In difference, the matching cost Sum of Absolute Differencess (SADs) is defined as

$$f_{SAD}(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1, \tag{1.4}$$

where we have replaced the ℓ_2-norm with an ℓ_1-norm which is known to be robust against outliers. Another alternative is the Normalized Cross Correlation (NCC) [110] defined as

$$f_{NCC}(\mathbf{p}, \mathbf{q}) = \frac{\sum_{i=1}^{MN} (p_i - \bar{p})(q_i - \bar{q})}{\sqrt{\sum_{i=1}^{MN} (p_i - \bar{p})^2 (q_i - \bar{q})^2}} \tag{1.5}$$

where

$$\bar{p} = \frac{1}{MN} \sum_{i=1}^{MN} p_i \qquad \bar{q} = \frac{1}{MN} \sum_{i=1}^{MN} q_i \tag{1.6}$$

are the mean of the patches $\mathbf{p}$ and $\mathbf{q}$, respectively. The *NCC* is a similarity measures where larger values reflect more similarity. Thus, we have to add a minus in Eq. (1.5) to make it a *cost* measure. In difference to the *SAD* and *SSD* the *NCC* is invariant to affine intensity changes and thus often more robust.

There are many more classical similarity measures used in stereo such as *e.g.* Mutual Information [71], *etc.* Recently, also learned matching cost functions using a *CNN* have been proposed as *e.g.* done by Kendall et al. [78]. We refer the reader to these papers for more information about these matching costs.

Local Features Brightness information, as used in Eq. (1.2), is simple to use but often ambiguous. For example, the color of the roof of the church in Fig. 1.1 is the same for all roof pixels. Thus, it is impossible to find the correct corresponding point using only brightness or color information. This problem can be mitigated by first extracting features $\varphi_0, \varphi_1 \colon \Omega \to \mathbb{R}^F$ out of the images and then using the features in the correspondence problem. Taking this into account leads to the new optimization problem

$$\min_d \int_\Omega f(\varphi_0(\mathbf{x}),\ \varphi_1(\mathbf{x} - d(\mathbf{x})))d\mathbf{x}, \tag{1.7}$$

where we compute the matching cost on the feature representations instead of on the brightness information. Note that we can still use the similarity measures shown in the previous paragraph.

It remains to answer the question of which features should be actually used in Eq. (1.7). Classical features are *e.g.* Scale Invariant Feature Transform (SIFT) features [117], the Census Transform [216] or the Rank Transform [216]. Although these features have been heavily used in classical stereo methods [171], the question of which features are the *best* for patch comparison is remaining. Unfortunately, the answer to this question is unknown. Therefore, modern approaches propose to *learn* optimal features for stereo matching with a *CNN* and show significant performance improvements compared to classical methods [132]. Learning optimal features for matching and integrating them into the stereo pipeline lies at the core of this thesis. We refer the reader to Chapters 3 to 6 for more information about the actual architecture for feature learning for the stereo task.

Cost Volume and Cost Aggregation Equipped with a matching cost and a feature representation of the two input images we can build a *cost volume*. To this end, we first define a continuous set $\mathcal{D} = [0, D]$ representing the possible disparities, which we can then use to define a cost volume $C_0 \colon \Omega \times \mathcal{D} \to \mathbb{R}$ for a specific disparity $d \colon \Omega \to \mathcal{D}$ as

$$C_0(\mathbf{x}, d) = f(\varphi_0(\mathbf{x}),\ \varphi_1(\mathbf{x} - d(\mathbf{x}))). \tag{1.8}$$

The cost volume contains all the necessary information about the stereo problem and can thus be used as an input for further processing steps. To this end, we can use the initial cost volume C_0 to compute a new cost volume, where we aggregate information over local support regions. Formally, this can be defined as

$$C(\mathbf{x}, d) = (w * C_0)(\mathbf{x}, d), \tag{1.9}$$

where we convolve a weight filter w with the cost volume C_0. We can use either 2D or 3D weights for w. 2D weights are applied at fixed disparities and fast to compute. However, they favor fronto-parallel surfaces which can be sub-optimal. In difference, 3D weights also allow for slanted surfaces, but come with a higher computational cost. The weights w can be *e.g.* computed based on a bilateral filter [189] in the color and disparity domain or using cross-based cost aggregation [222]. For more methods used for cost aggregation in the stereo setting the reader is referred to [171].

Disparity Computation Using the cost volume, we can directly compute the Winner Takes All (WTA) solution to the stereo problem using

$$d^* \in \arg\min_d \int_\Omega C(\mathbf{x}, d) d\mathbf{x}, \tag{1.10}$$

which corresponds to the solution of the *local* stereo problem, because we considered only local information in order to compute the cost volume. Figure 1.2b shows a visualization of a local stereo result computed with Eq. (1.10). It shows that local matching is not sufficient in order to get high-quality disparity maps as a result. This is especially a problem in untextured regions of the image and in occlusions. Also map uniqueness [230], *i.e.* that one pixel from the reference image can only match to exactly one pixel in the second image, is not modeled in the *WTA* solution.

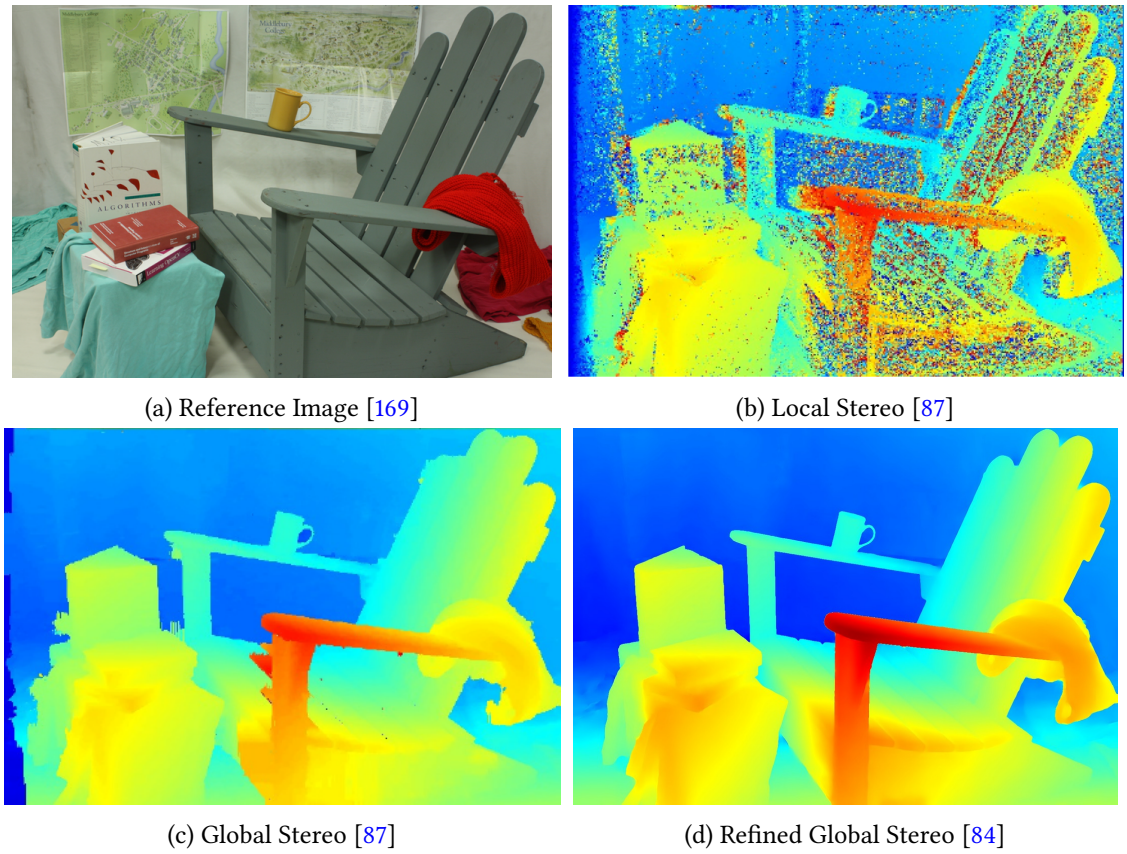
Optimization To tackle the problems of local stereo methods, we can add *global* information to the stereo problem. This results in the new global optimization problem

$$\min_d D(d) + R(d), \tag{1.11}$$

where $d\colon \Omega \rightarrow \mathbb{R}$ is the disparity map. The function D is called *data term* capturing local information measuring how well the disparity d agrees with the observed input images. The function R is called the *regularizer*. As we will show in Section 2.4.3, the regularization term has a strong connection to the *prior* in the Bayesian probability setting. It can therefore be used to inject prior knowledge about the stereo problem and prefer physically plausible solutions. A commonly used regularizer is based on a smoothness assumption defined on neighboring pixels. For the stereo problem this means that we want to avoid small and isolated regions and prefer slanted surfaces and sparse depth discontinuities. This can be seen in Fig. 1.2c, where we have eliminated almost all the noise compared to the *WTA* solution in Fig. 1.2b.

There are many possible instantiations of the optimization problem (1.11). We will use two contrary formulations in this thesis. In the first formulation Eq. (1.11) is defined by a *CRF*, which poses a discrete optimization problem (*c.f.* Chapters 3, 4 and 6). Second, we formulate the optimization in the continuous domain (*c.f.* Chapter 5).

Disparity Refinement Many stereo algorithms compute integer valued disparity maps, which could, however, introduce discretization artifacts. In order to get a sub-pixel accurate disparity map, we need to perform a disparity refinement step. This can be *e.g.* done by fitting a quadratic function to the matching cost [118, 220]. Furthermore, there are also works proposing to *learn* the refinement of disparity maps with *CNNs* [80, 84]. Figure 1.2d shows the effect of a learned stereo refinement method. We will tackle stereo refinement in Chapter 5.



(a) Reference Image [169] (b) Local Stereo [87]

(c) Global Stereo [87] (d) Refined Global Stereo [84]

Figure 1.2: Comparison of local, global and refined stereo results on the Adirondack image [169]. Disparity values are color coded from cold = far away to warm = close to the camera.

1.3 Contribution and Outline

In this thesis we are interested in combining *DL* with classical discrete and continuous optimization models/techniques. It will turn out that this allows us to get the best out of both worlds: The power and expressiveness of *DL* and the efficiency and interpretability of classical optimization.

- ***CNN+CRF* models** We consider the generic optimization problem (1.11) in the discrete setting. While many methods in the stereo literature learn features for matching cost computation [120, 220], they apply the global optimization only as a post-processing. However, the resources are not fully exploited in this setting, because it is impossible to learn a robust matching term for homogeneous regions. In contrast, this is perfectly possible for the regularizer in the optimization. We therefore show how to enable end-to-end learning of the *CNNs* together with the *CRF* via a Structured Support Vector Machine (SSVM) formulation.
- **Generic *CNN+CRF* framework** The end-to-end learning in the previous point is only possible if the *CRF* is the last layer of the model. Here, we get rid of this restriction and therefore enable to integrate *CRF* inference layers at any position in the model. Thus, we

can now build hierarchical models, where the output of the *CRF* from a coarse resolution is used as an input for the next finer resolution – everything fully end-to-end trainable. Furthermore, we introduce a generic dynamic programming building block which allows us to instantiate multiple *CRF* inference algorithms together with their gradient.

- ***CNN* + variational optimization** One disadvantage of discrete graphical models such as *CRFs* is that their output is discrete. This is unnatural for the geometrical stereo problem. Therefore, we consider here a continuous variant of the generic optimization problem defined in Eq. (1.11). Specifically, we tackle the stereo-refinement task. This allows to get rid of the discretization artifacts from the discrete solution of a *CRF* and yields sub-pixel accurate solutions.

- **Learning without labeled data** *CNNs* usually require a large amount of labeled training data. However, for stereo it is very difficult and also expensive to get labeled data because hand-labeling is clearly out of reach. We therefore propose to exploit geometrical constraints, which we can use to automatically generate training data for the stereo problem. This allows to train high-performance models also on datasets, where no ground-truth data exists.

Outline Chapter 2 gives an overview of all relevant topics to understand the later chapters describing the contributions. We have therefore grouped the related work section into Discrete Optimization, Continuous Optimization and Machine Learning. Discrete Optimization covers the basics of probabilistic graphical models such as Markov Random Fields (MRFs) and *CRFs*, relates the probabilistic interpretation to energy minimization and shows how we can perform inference in these models. These sections are especially relevant for Chapters 3, 4 and 6.

In the section Continuous Optimization, we review the most important concepts of convex analysis together with convex optimization algorithms. Many of the presented algorithms are also used in *ML* in order to find optimal parameters for the model.

Section 2.6 gives an overview of the different kinds of *ML* and shows the most important building blocks for *CNNs*. Furthermore, we discuss model training and show how the objective function can be derived from risk minimization.

1.4 Notation

We use the subscripts + and ++ to restrict a set to positive and strictly positive values. For example, we write $\mathbb{R}_+$ to denote positive real values and $\mathbb{R}_{++}$ to denote strictly positive real values, respectively. Upper-case letters such as $M, N \in \mathbb{N}_+$ are used to specify *e.g.* the number of elements of a vector or the size of an image.

We consider images to be functions $I\colon \Omega \to \mathbb{R}^C$, where $\Omega \subset \mathbb{R}_+^2$ is the domain of the function and C is the number of channels which is 3 for an *RGB*-color images. If the image size is important, we specialize the domain to $\Omega_{MN} \coloneqq \{1, \dots, M\} \times \{1, \dots, N\}$, which corresponds to a matrix of M rows and N columns.

We use bold-face lowercase letters to denote a column vector such as *e.g.* $\mathbf{x} = (x_1, \dots, x_N)$ and thus a row vector is denoted by $\mathbf{x}^\top$. Similarly, we use bold-face uppercase letters to denote matrices, *e.g.* $\mathbf{M}$. By convention we always index matrices row-major, *i.e.* the element M_{ij} is the matrix element in row i and column j. We use curly letters for general sets, *e.g.* $\mathcal{X}, \mathcal{Y}$.

The notation $\{1, \dots, N\}$ denotes a discrete set. Here, we have defined the integer values from 1 to N. Similarly, the notation $[a, b]$ denotes the closed interval from a to b, *i.e.* the values a and b are included to the interval. In difference, the notation (a, b) denotes the open interval from $a \in \mathbb{R}$ to $b \in \mathbb{R}$, where a and b are not included in the interval. Thus, $(a, b]$ would denote a half-open interval excluding a but including b.

2 Related work

2.1 Probability Theory

Probabilities allow us to incorporate uncertainty into our models and thus enable to make optimal predictions with noisy, incomplete or ambiguous data. We give a brief overview of the basics of probability theory in this section. We refer the reader to [63] for a more complete overview of probability theory.

2.1.1 Probabilities and Random Variables

A discrete random variable is denoted by capital letter X and can take values from a finite set $\mathcal{X}$ called *state space*. The realization that a random variable takes a specific value is denoted by $X = x$, indicating the event that the random variable X takes the value x. The complete probability distribution is denoted by $p(X)$ and fulfills the two properties i) $0 \leq p(x) \leq 1 \quad \forall x \in \mathcal{X}$ and ii) $\sum_{x \in \mathcal{X}} p(x) = 1$. We denote the probability of an event $X = x$ by $p(X = x)$, which is equivalent to the shortcut $p(x)$. Hence, we use the notation to distinguish between the whole probability distribution $p(X)$ and the probability of an event $p(x)$. Let X and Y be two random variables which can take values from the state spaces $\mathcal{X}$ and $\mathcal{Y}$. Then, the joint probability distribution over these two random variables is denoted by $p(X, Y)$ and $p(x, y)$ denotes the probability of the event that $X = x$ and $Y = y$. A conditional probability distribution is denoted by $p(X|Y)$, which reads as "the probability of X given Y". We can also group N random variables to one random vector of size N, *i.e.* $\mathbf{X} = (X_1, \ldots, X_N)$, which we denote by a bold face capital letter.

2.1.2 Probability Calculus

The probability framework contains two fundamental rules, the *product rule* of probability and the *sum rule* of probability. Let X and Y be random variables with their corresponding state spaces $\mathcal{X}$ and $\mathcal{Y}$. Then,the product rule of probability is defined as

$$p(X, Y) = p(Y|X)p(X), \tag{2.1}$$

where $p(X, Y)$ is the joint distribution over X and Y, $p(Y|X)$ is the conditional distribution of Y given X and $p(X)$ is the distribution over X. Note that we can similarly factorize the joint distribution as $p(X, Y) = p(X|Y)p(Y)$. The sum rule of probability is defined as

$$p(X) = \sum_{y \in \mathcal{Y}} p(X, Y = y), \tag{2.2}$$

where $p(X)$ is the distribution over X and $p(X, Y)$ is the joint distribution over X and Y. The sum-rule of probability in Eq. (2.2) shows that we can *marginalize* out one variable. Hence, marginalizing means applying the sum-rule to remove the dependence on one variable. Therefore, the resulting probability distribution $p(X)$ in Eq. (2.2) is also called *marginal distribution*. We can also marginalize over X to get a distribution only dependent on Y, *i.e.* $p(Y) = \sum_{x \in \mathcal{X}} p(X = x, Y)$.

Bayes Theorem The most fundamental equation in probability theory is *Bayes theorem* also known as *Bayes rule*. It is defined as

$$\underbrace{p(Y|X)}_{\text{Posterior}} = \frac{\overbrace{p(X|Y)}^{\text{Likelihood}} \overbrace{p(Y)}^{\text{Prior}}}{\underbrace{p(X)}_{\text{Evidence}}}, \tag{2.3}$$

which can be computed by *e.g.* applying the product rule Eq. (2.1) twice. The individual terms in Eq. (2.3) are denoted by *likelihood*, *prior*, *evidence* and *posterior*. To illustrate their purpose consider the setting where X is the random variable for the data and Y is the random variable for the labels. Then the likelihood $p(X|Y)$ describes how likely the data is given the label. The prior $p(Y)$ captures information we know in advance. For example we could encode here that some labels are more likely than others and thus inject prior knowledge. The evidence is used to normalize the resulting posterior distribution $p(Y|X)$ appropriately. The posterior defines the distribution after (= lat. post) we have evaluated the right-hand side and thus incorporates all the information necessary to make predictions given unseen data. The task of computing $p(Y|X)$ is called *probabilistic inference*.

2.1.3 Decision Theory

As shown in the previous section, the posterior distribution $p(Y|X)$ in Eq. (2.3) is a probability distribution. However, eventually we have to make a prediction which should be optimal in some sense. We will investigate two possibilities here, the Maximum-A-Posteriori (MAP) solution and the expected solution.

***MAP* solution** The *MAP* decision is given by

$$\hat{y}_{MAP} = \operatorname*{argmax}_{y \in \mathcal{Y}} p(y|x), \tag{2.4}$$

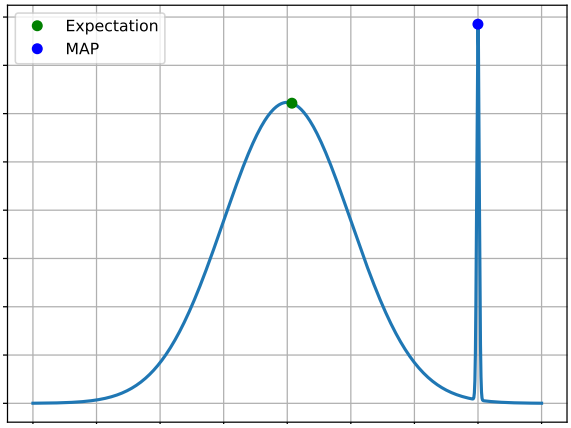


Figure 2.1: MAP vs Expectation. The *MAP* solution is uncharacteristic for this probability distribution, while the expected solution reflects the characteristic of the probability distribution.

where $\hat{y}_{MAP} \in \mathcal{Y}$ can only take values from the state space $\mathcal{Y}$ and is referred to as the maximum a-posteriori solution. Equation (2.4) tells that we should select the most probable solution. This also makes sense intuitively and it can indeed be shown that the *MAP* solution actually minimizes the misclassification rate [12]. Thus, this is the preferred decision rule for Classification problems.

Expected solution A second option is to decide for the expected solution. We can compute the expected solution with

$$\hat{y}_{\mathbb{E}} = \mathbb{E}_{Y \sim p(\cdot|x)}[Y] = \sum_{y \in \mathcal{Y}} y p(y|x), \tag{2.5}$$

where $\hat{y}_{\mathbb{E}}$ can also take values that are not in the state space $\mathcal{Y}$. This is *e.g.* beneficial if sub-classes exist and can be used to recover a sub-label accurate solution. A second property of the expected solution is shown in Fig. 2.1. It can be seen that the expected solution reflects the characteristic of the probability distribution better than the *MAP* solution. The expected solution (2.5) minimizes the expected loss [12].

2.2 Mathematical Preliminaries

Let us define the necessary mathematical preliminaries in this section. We will use these basic concepts throughout the thesis.

2.2.1 Inner Product

An inner product on a vector space $\mathbb{V} = \mathbb{R}^N$ is defined as the function $\langle \cdot, \cdot \rangle \colon \mathbb{V} \to \mathbb{R}$. An inner product fulfills the properties

(1) **Symmetry:** $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$ for any $\mathbf{x}, \mathbf{y} \in \mathbb{V}$.

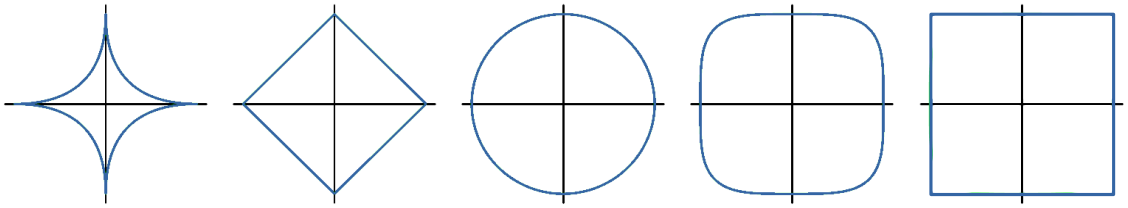


Figure 2.2: Visualization of ℓ_p-unit-balls. Note that $\ell_{0.5}$ is not a norm.

(2) **Additivity:** $\langle \mathbf{x}, \mathbf{y} + \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle$ for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{V}$.

(3) **Homogeneity:** $\langle \lambda \mathbf{x}, \mathbf{y} \rangle = \lambda \langle \mathbf{x}, \mathbf{y} \rangle$ for any $\mathbf{x}, \mathbf{y} \in \mathbb{V}$ and $\lambda \in \mathbb{R}$.

(4) **Positive definiteness:** $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$ for any $\mathbf{x}, \mathbf{y} \in \mathbb{V}$ and $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ iff $\mathbf{x} = 0$.

The inner product is given by the vector dot product on $\mathbb{V} = \mathbb{R}^N$ as

$$\langle \mathbf{x}, \mathbf{y} \rangle \coloneqq \mathbf{x}^\top \mathbf{y} = \sum_{i=1}^{N} x_i y_i, \tag{2.6}$$

and by

$$\langle \mathbf{x}, \mathbf{y} \rangle_\mathbf{M} \coloneqq \mathbf{x}^\top \mathbf{M} \mathbf{y}, \tag{2.7}$$

on a symmetric and positive definite matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$.

2.2.2 Vector Norms

A norm $\|\cdot\|$ on a vector space $\mathbb{V} = \mathbb{R}^N$ is defined as a function $\|\cdot\| : \mathbb{V} \to \mathbb{R}$ mapping from a vector space $\mathbb{V}$ to the real numbers $\mathbb{R}$. A norm fulfills the properties

(1) **Nonnegativity:** $\|\mathbf{v}\| \geq 0$ for any $\mathbf{v} \in \mathbb{V}$.

(2) **Definiteness:** $\|\mathbf{v}\| = 0$ iff $\mathbf{v} = \mathbf{0}$.

(3) **Positive Homogeneity:** $\|\lambda \mathbf{v}\| = |\lambda| \cdot \|\mathbf{v}\|$ for any $\mathbf{v} \in \mathbb{V}$ and $\lambda \in \mathbb{R}$.

(4) **Triangle Inequality:** $\|\mathbf{v} + \mathbf{w}\| \leq \|\mathbf{v}\| + \|\mathbf{w}\|$ for any $\mathbf{v}, \mathbf{w} \in \mathbb{V}$.

If property (2) is not satisfied, it is called a *semi-norm* and a vector space $\mathbb{V}$ with a norm is called a *normed vector space*. Additionally, an inner product on $\mathbb{V}$ induces a norm on $\mathbb{V}$ via $\|\mathbf{v}\| \coloneqq \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$. The $\mathbf{Q}$-induced norm or the norm in the metric $\mathbf{Q}$, respectively, is defined as

$$\|\mathbf{v}\|_\mathbf{Q} \coloneqq \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle_\mathbf{Q}} = \sqrt{\mathbf{v}^\top \mathbf{Q} \mathbf{v}}, \tag{2.8}$$

where $\mathbf{Q} \in \mathbb{R}^{N \times N}$ is a symmetric and positive definite matrix.

ℓ_p-norm Let us next investigate the ℓ_p-norm. The ℓ_p-norm is defined as

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{N} |v_i|^p\right)^{\frac{1}{p}}, \tag{2.9}$$

where $\mathbf{v} = (v_1, \ldots, v_N) \in \mathbb{V}$ is a vector and $p \in [1, \infty)$ is a parameter. Note that if $p < 1$, then Eq. (2.9) defines no longer a norm, because then the triangle inequality is no longer fulfilled. Figure 2.2 shows a visualization of different ℓ_p-norm balls. Additionally, we will define three special cases of the ℓ_p-norm in the upcoming paragraphs.

ℓ_1-norm The ℓ_1-norm is also referred to as *Manhattan* norm and can be computed by setting $p = 1$. We thus get the definition of the ℓ_1-norm

$$\|\mathbf{v}\|_1 = \sum_{i=1}^{N} |v_i|. \tag{2.10}$$

ℓ_2-norm The ℓ_2-norm is the probably most often used norm, which corresponds to our interpretation of distance in the Euclidean space. Therefore, the ℓ_2 norm is also called *Euclidean norm*. We set $p = 2$ and get the definition of the ℓ_2-norm

$$\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{N} |v_i|^2}. \tag{2.11}$$

Sometimes we will omit the explicit subscript and use the simpler notation

$$\|\mathbf{v}\| = \|\mathbf{v}\|_2. \tag{2.12}$$

ℓ_∞-norm The last important norm is the so-called *ℓ-infinity norm*. It is a special case of the ℓ_p-norm and can be computed by

$$\|\mathbf{v}\|_\infty = \max_{i=1,\ldots,N} \{|v_i|\}, \tag{2.13}$$

which is the maximal absolute component of $\mathbf{v}$.

2.2.3 Matrix Norm

Let $\|\cdot\|_p$ and $\|\cdot\|_q$ be vector norms on $\mathbb{R}^N$ and $\mathbb{R}^M$, respectively. Given a matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$, the induced matrix norm is defined as

$$\|\mathbf{A}\|_{p,q} := \max_{\mathbf{x} \in \mathbb{R}^N :\ \|\mathbf{x}\|_p \leq 1} \|\mathbf{A}\mathbf{x}\|_q = \sup_{\mathbf{x} \in \mathbb{R}^N \setminus \{0\}} \frac{\|\mathbf{A}\mathbf{x}\|_q}{\|\mathbf{x}\|_p}, \tag{2.14}$$

which immediately implies the important inequality

$$\|\mathbf{A}\mathbf{x}\|_q \leq \|\mathbf{A}\|_{p,q}\|\mathbf{x}\|_p \tag{2.15}$$

revealing the relation between vector norms and matrix norms. If $p = q$, then we use the shortcut $\|\mathbf{A}\|_p = \|\mathbf{A}\|_{p,q}$.

As an example, let us next consider the 2, 1-matrix norm given by

$$\|\mathbf{M}\|_{2,1} = \sum_{i=1}^{M} \left| \sqrt{\sum_{j=1}^{N} |M_{ij}|^2} \right|, \tag{2.16}$$

which is often used in optimization for computer vision.

1, 2 and ∞ Matrix Norms If $\|\cdot\|_p = \|\cdot\|_q = \|\cdot\|_1$, then the induced norm of a matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$ is given by

$$\|\mathbf{A}\|_1 = \max_{j=1,2,\ldots,N} \sum_{i=1}^{N} \|A_{ij}\|, \tag{2.17}$$

which is the largest absolute column sum norm. For $\|\cdot\|_p = \|\cdot\|_q = \|\cdot\|_2$ we obtain the spectral norm

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^\top \mathbf{A})} = \sigma_{\max}(\mathbf{A}), \tag{2.18}$$

which corresponds to the largest singular value of $\mathbf{A}$. If $\|\cdot\|_p = \|\cdot\|_q = \|\cdot\|_\infty$, the induced norm of a matrix $\mathbf{A}^{M\times N}$ is given by

$$\|\mathbf{A}\|_\infty = \max_{i=1,2,\ldots,M} \sum_{j=1}^{M} |A_{ij}|, \tag{2.19}$$

which is the largest absolute row sum norm.

Frobenius Norm The Frobenius norm is an example of a matrix norm, which is not an induced norm. It is defined by

$$\|\mathbf{A}\|_F := \sqrt{\sum_{i=1}^{N} \sum_{j=1}^{M} A_{ij}^2}, \tag{2.20}$$

where $\mathbf{A} \in \mathbb{R}^{M\times N}$.

Schatten Norm A class of matrix norms that are based on ℓ_p norms of the singular values $\sigma_i(\mathbf{A})$ with $i = 1, \ldots, \min\{M, N\}$ is called Schatten norm. For $p \geq 1$ it is given by

$$\|\mathbf{A}\|_{S_p} = \left(\sum_{i=1}^{\min\{M,N\}} (\sigma_i(\mathbf{A}))^p \right)^{\frac{1}{p}}. \tag{2.21}$$

2.2.4 Adjoint Operator

Let $\mathbb{V} = \mathbb{R}^N$ and $\mathbb{E} = \mathbb{R}^M$ be vector spaces and $\mathbf{A}\colon \mathbb{V} \to \mathbb{E}$ a linear transform. Then the adjoint linear transform $\mathbf{A}^*\colon \mathbb{E} \to \mathbb{V}$ is defined as

$$\langle \mathbf{A}\mathbf{x}, \mathbf{y} \rangle_{\mathbb{E}} = \langle \mathbf{x}, \mathbf{A}^*\mathbf{y} \rangle_{\mathbb{V}}, \tag{2.22}$$

where $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^M$. Since we work on $\mathbb{V} = \mathbb{R}^N$ and $\mathbb{E} = \mathbb{R}^M$, the adjoint transform is given by the usual matrix transpose $\mathbf{A}^* = \mathbf{A}^\top$, because

$$\langle \mathbf{A}\mathbf{x}, \mathbf{y} \rangle = (\mathbf{A}\mathbf{x})^\top \mathbf{y} = \mathbf{x}^\top (\mathbf{A}^\top \mathbf{y}) = \langle \mathbf{x}, \mathbf{A}^\top \mathbf{y} \rangle. \tag{2.23}$$

2.2.5 Properties of functions

Domain of a function The domain of a proper extended real-valued function $f\colon \mathbb{V} \to (-\infty, \infty]$ is written as

$$\operatorname{dom}(f) \coloneqq \{\mathbf{x} \in \mathbb{V}\colon f(\mathbf{x}) < \infty\}. \tag{2.24}$$

Lower Semi-Continuous A function $f\colon \mathbb{V} \to [-\infty, \infty]$ is lower semi-continuous at $\mathbf{x} \in \mathbb{V}$ if

$$f(\mathbf{x}) \leq \liminf_{n \to \infty} f(\mathbf{x}_n) \tag{2.25}$$

for any sequence $\{\mathbf{x}_n\}_{n \geq 1} \subset \mathbb{V}$ for which $\mathbf{x}_n \to \mathbf{x}$ when $n \to \infty$. The whole function is lower semi-continuous if it is lower semi-continuous at every point $\mathbf{x} \in \mathbb{V}$.

2.3 Optimization problems in Computer Vision and Machine Learning

In many situations in research, industry, but also in life we aim to find the "best" option out of a range of choices for a specific problem. Optimization is a discipline of mathematics which provides tools and recipes to find these best options in a structural way. An important family of optimization problems specifically suitable for practical problems arising in computer vision and machine learning can be defined in terms of *energy minimization problems*

$$\min_{\mathbf{x}} E(\mathbf{x}) \coloneqq D(\mathbf{x}) + R(\mathbf{x}), \tag{2.26}$$

where $\mathbf{x} \in \mathbb{R}^N$ is the N-dimensional optimization variable and $E\colon \mathbb{R}^N \to \mathbb{R}$ is the objective function, which is often referred to as *energy* function. It consists of a *data term* $D(\mathbf{x})$ and a *regularization term* $R(\mathbf{x})$. The data term measures thereby the compatibility between the measurements and our current choice of $\mathbf{x}$ and the regularization term is used to impose regularity and smoothness on the result. The "min" in Eq. (2.26) states that we want to find a solution having the lowest energy and thus Eq. (2.26) poses an optimization problem. Note that the result of Eq. (2.26)

is a scalar stating the energy of the optimal solution. However, sometimes we are also interested in the argument yielding minimal energy. Formally, this can be written as

$$\mathbf{x}^* \in \arg\min_{\mathbf{x}} E(\mathbf{x}) \tag{2.27}$$

where we have changed the "min" with "arg min" to get the optimizer $\mathbf{x}^*$. We generally denote optimal solutions of optimization problems with an asterisk in the super-script. We have used "element of" instead of "equality" to denote that there could be multiple optimizers yielding the same energy.

Depending on the domain and on the actual structure imposed by the function E in Eq. (2.26) different optimization algorithms are appropriate. In this thesis we generally distinguish between *discrete* (*c.f.* Section 2.4) and *continuous* (*c.f.* Section 2.5) optimization problems. Both discrete and continuous problems have different properties and thus different algorithms are required. Finding efficient algorithms for specific practical problems is the central goal in optimization. We will review the most important problems as well as suitable algorithms for solving them in the upcoming sections.

2.4 Discrete Optimization

Discrete optimization is a sub-discipline of optimization, where the optimization variable can only attain discrete values. In the computer vision community discrete optimization problems are often referred to as *labelling problems*, where the discrete values encode integer-valued labels $\{1, 2, \dots\}$. Depending on the task at hand the labels can then be interpreted as *e.g.* disparities in the stereo problem or semantic classes in the semantic segmentation problem. Due to the grid-structure of the pixel-grid in natural images, we are mainly interested in (probabilistic) undirected graphical models such as Markov Random Fields (MRFs) or Conditional Random Fields (CRFs). They can be represented by grid-graphs as shown in Fig. 2.3 and we will review them in detail in the upcoming sections. The material presented in this section is partly based on the excellent textbooks [14, 143, 168]. Note that we will not cover directed graphical models which are known as Bayes Networks here. For more information on Bayes Networks we refer the reader to *e.g.* [12, 92].

2.4.1 Markov Random Fields

MRFs are undirected probabilistic graphical models defined as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. Vertices are also often referred to as "nodes" and thus we will use both terms interchangeably in this thesis. The structure of the graph $\mathcal{G}$ thereby defines the relations between the nodes. A sample *MRF* is depicted in Fig. 2.3, where the nodes of the graph are visualized as blue circles and the edges in the graph are visualized as connecting lines. Denoting the label-set with $\mathcal{Y}$, every node in the graph represents a discrete random variable $Y_i \in \mathcal{Y}$. Thus the *MRF* actually defines a probability distribution $p(Y_1, \dots, Y_{|\mathcal{V}|})$. The graphical

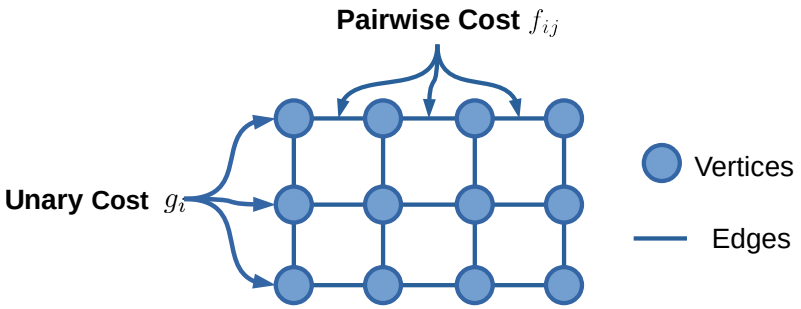


Figure 2.3: Overview of a pairwise *MRF*. Vertices (= nodes) define random variables and the edges encode conditional dependence of random variables. Unary costs are defined on every node and pairwise costs are defined on pairs of nodes visualized as edges.

visualization in Fig. 2.3 also directly reveals that an *MRF* encodes factorization properties of the underlying probability distribution p and thus shows the conditional dependence of the individual random variables Y_i. It can for example be seen that every node is only dependent on its direct neighbors (encoded by the edges), but not on all the other nodes present. This property is also referred to as the *Markov property*.

An *MRF* defines a family of joint probability distributions over a random vector $\mathbf{Y}$, obeying factorization properties given by the graph structure. Formally, it is defined via the Hammersley-Clifford theorem [37] as

$$p(\mathbf{Y}) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{Y}), \tag{2.28}$$

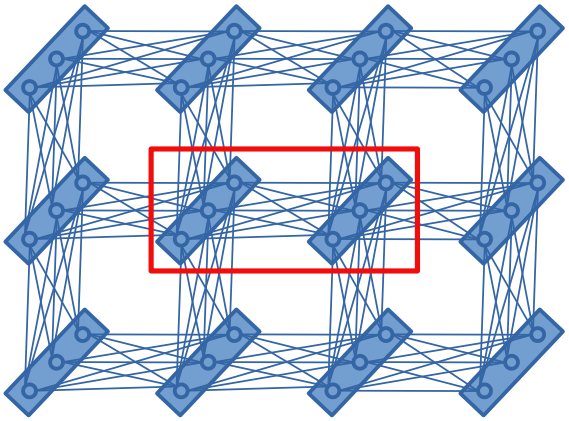
where the product is defined on the set of cliques C in the graph. A clique is a subgraph of $\mathcal{G}$ in which every node is connected with every other node. In computer vision, we work mainly with (hidden) pairwise *MRFs*. The factorization of this model can be written down in terms of unary potentials and pairwise potentials with

$$p(\mathbf{Y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{i \in \mathcal{V}} \phi_i(Y_i; x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(Y_i, Y_j), \tag{2.29}$$

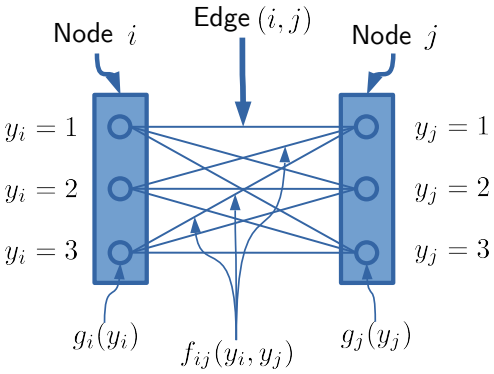
where the unary potentials $\phi_i : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ are the factors for every node i and the pairwise potentials $\psi_{ij} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ are the factors of the edges defined on pairs of nodes. Note that we index all nodes in $\mathcal{V}$ with scalar indices i and all edges in $\mathcal{E}$ with tuple indices (i, j). Thus, the index does not refer to the same node/edge in the two products. The constant $Z(\mathbf{x})$ is known as the *partition function* and used to appropriately normalize the probability distribution p. The normalization Z is given by

$$Z(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}^{|\mathcal{V}|}} \left(\prod_{i \in \mathcal{V}} \phi_i(y_i; x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(y_i, y_j) \right), \tag{2.30}$$

where the summation is defined over all possible labelings. Note, however, that it is usually



(a) Verbose *MRF*



(b) Detailed node

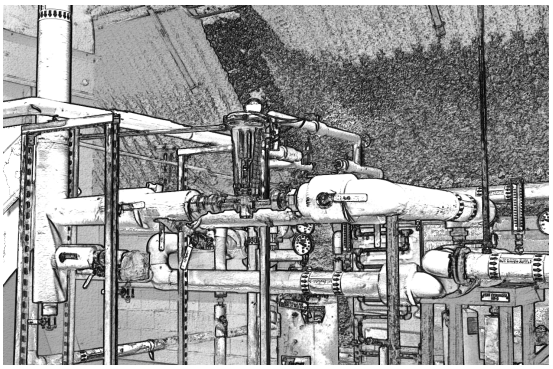
Figure 2.4: Verbose visualization of a *MRF* / *CRF*. Left: The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of a *MRF* / *CRF* showing the connection between all source and target labels. Right: A zoomed variant of the two nodes in the red rectangle describing the core parts of the *MRF* / *CRF*.

intractable to compute Z for real-world problems. To see this, we consider an *MRF* defined on the pixels of a tiny image with size 10×10. Let us further assume the task is a binary labeling problem, *i.e.* $|\mathcal{Y}| = 2$. Then the summation in Eq. (2.30) would have $2^{10 \cdot 10} = 2^{100}$ terms which is intractable to compute.

Let us now investigate the definition of an *MRF* given in Eq. (2.29) in detail. First, using Eqs. (2.29) and (2.30), we observe that an *MRF* is strongly related to Bayes' rule in Eq. (2.3). We can associate the unary potentials to the likelihood, the pairwise potentials to the prior, $Z(\mathbf{x})$ to the evidence and the left-hand side to the posterior. Let us next consider the verbose visualization showing all available labels and edges shown in Fig. 2.4a. If we compare this visualization with the formal definition in Eq. (2.29), we see that the nodes together with labels depict exactly the unary potentials ϕ_i. Here, every node can take one out of three states, *i.e.* $|\mathcal{Y}| = 3$. The edges in the graph depict the joint probability between neighboring pairs of nodes. By counting the edges between neighboring nodes, it is easy to verify that we have nine states for the joint probability. This fits perfectly to the definition of the pairwise potentials ψ_i, which are defined on the Cartesian product $\mathcal{Y} \times \mathcal{Y}$. Hence, the edges encode the jump probability from every source label to every target label. Figure 2.4b shows a detailed visualization of a node pair including the possible states.

We also want to note that an *MRF* always yields a probability distribution as a result for every node and not directly a single estimate. Depending on the inference algorithm (*c.f.* Section 2.4.4), the resulting probability distribution corresponds to the *max-marginals* or to the (true) *marginals* of every node. These resulting marginal distributions are beneficial in two regards: First, we can use tools from decision theory (*c.f.* Section 2.1.3) to make our decision based on the complete distribution and second we automatically get a confidence for our decision, since we can simply evaluate the probability of our choice.

(a) Reference image



(b) Edge weights

Figure 2.5: Exploiting information from the observation. Left: The "Pipes" images from the Middlebury [169] benchmark. Right: Edge weights extracted from the image using [87], where dark regions encode low jump costs and bright encodes high jump costs. Note that the edge weights can be directly used to encourage label jumps at strong object boundaries like *e.g.* around the pipes and discourage label jumps in homogeneous regions as *e.g.* on the pipes.

2.4.2 Conditional Random Fields

Conditional Random Fields (CRFs) [102] are an important variant of *MRFs*. Similar to the *MRF* defined in Eq. (2.29) the *CRF* also defines the posterior distribution $p(\mathbf{Y}|\mathbf{x})$. However, the factorization into the likelihood $p(\mathbf{x}|\mathbf{Y})$ and prior $p(\mathbf{Y})$ is no longer explicitly modeled [102]. Instead, the posterior distribution is directly modeled with unary and pairwise potentials. This will turn out to be beneficial in practice, since we can provide additional information to the pairwise term. Thus, we can define a *CRF* as the conditional probability distribution

$$p(\mathbf{Y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{i \in \mathcal{V}} \phi_i(y_i; x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(y_i, y_j; x_i, x_j), \tag{2.31}$$

where we have used the shorthand $p(\mathbf{Y}|\mathbf{x}) = p(\mathbf{Y}|\mathbf{X} = \mathbf{x})$ with $\mathbf{x} \in \mathcal{X}$ and added the dependence on the observation in all factors, *i.e.* ϕ_i and ψ_{ij}. We have separated the labels and the observation with a semi-colon, *i.e.* ";". In order to avoid clutter, we will not always explicitly write the dependence on $\mathbf{x}$ in the factors if it is clear from the context. Similar as in Eq. (2.30) for the *MRF* we can now compute the partition function for the *CRF* as

$$Z(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}^{|\mathcal{V}|}} \left(\prod_{i \in \mathcal{V}} \phi_i(y_i; x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(y_i, y_j; x_i, x_j) \right).$$

Let us next briefly investigate why the additional dependence on the observation is beneficial in practice. This explicit dependence on the observation allows for example to adjust the pairwise potentials based on the actual image content. In many computer vision problems we want to encourage label jumps on object edges and discourage label jumps in homogeneous regions. We

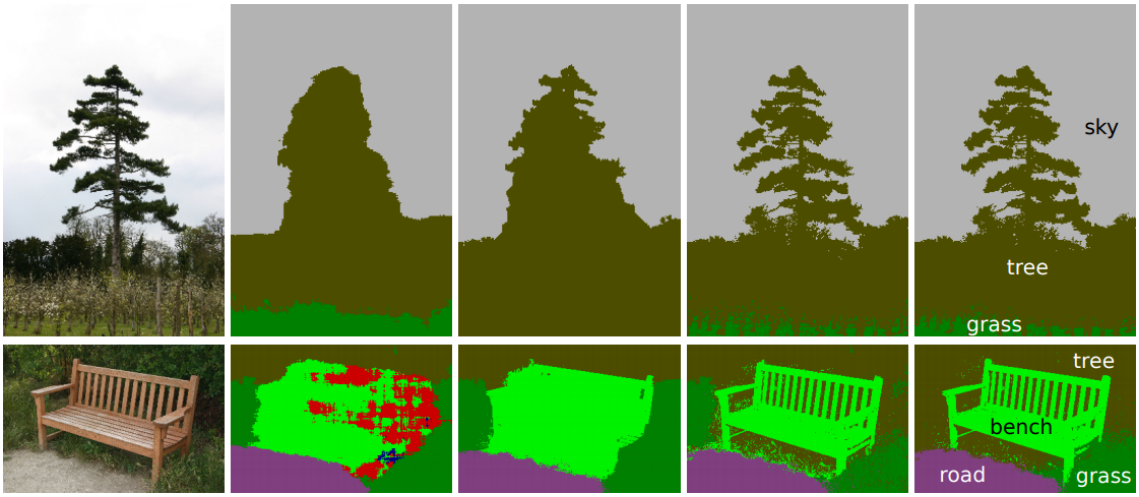


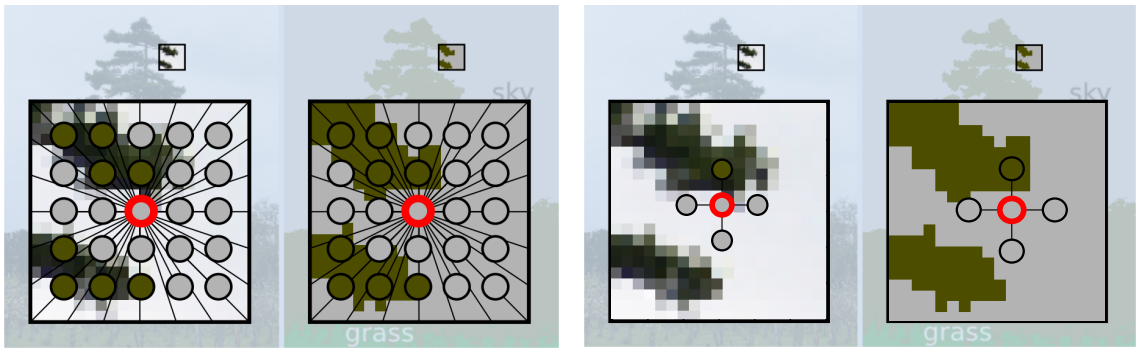
Figure 2.6: Fully connected *CRF*. From left to right: input image, unary only, P^n CRF of [91], fully connected CRF with Markov Chain Monte Carlo (MCMC) inference, fully connected CRF of [98]. Image courtesy of [98].

can use image gradients to modulate the pairwise potentials by increasing the jump probability if there is a strong gradient and decrease the jump probability if the image gradient is low. Figure 2.5 shows an exemplary edge map which was used exactly for this purpose.

CRF variants An important variant is the *fully connected CRF*. It has been mainly used for semantic segmentation problems [98, 225]. The main property of the fully connected *CRF* is that every node is connected to every other node in the graph. This is also known as dense connectivity. Dense *CRFs* therefore exhibit long range interactions. They allow for nodes to be directly dependent on spatially distant nodes and thus inject additional context as shown in Fig. 2.7. Figure 2.6 shows a comparison of a higher-order *CRF* and the fully connected *CRF* from the paper of Krähenbühl and Koltun [98]. The dense connectivity is of course the main advantage of the fully connected *CRF*. However, there is also a price to pay to keep inference tractable. The potentials are restricted to be Gaussian which are known to be non-robust against strong outliers. Nevertheless, this allows to compute the optimal solution efficiently using filtering (see [98] for details).

In this thesis, we are mainly interested in geometrical problems. Although the fully connected *CRF* has many nice properties, it implicitly encodes a fronto-parallel assumption. This is perfectly fine for semantic segmentation, but a significant limitation for stereo, where we actually want to get slanted surfaces instead. We will show in Chapters 3 and 4 that 4-connected *CRFs* are better suited for geometrical problems.

A second family of *CRFs* are *Gaussian CRFs* [196]. The advantage of Gaussian *CRFs* is that an exact solution can be computed in closed form. However, they are restricted to Gaussian potentials and thus not robust against outliers by design.



(a) Fully connected neighborhood (b) 4-connected neighborhood

Figure 2.7: Fully connected vs 4-connected neighborhood. The fully connected neighborhood has the advantage that information from distant neighbors can be used to compute the label for the current node. This is indicated by the green shaded nodes in (a). (b) shows the same region with a 4-connected neighborhood. The center node does not get direct information from any other green tree-node. Image adapted from [98].

2.4.3 Energy Minimization for MRF/CRF

We have motivated *MRFs* and *CRFs* in the probabilistic setting in Section 2.4.1 and Section 2.4.2, respectively. As we will see in Section 2.4.4, we will perform energy minimization to compute the marginal distribution $p(\mathbf{Y}|\mathbf{x})$, *i.e.* the left-hand side of Eq. (2.31). Therefore, we show here the connection between probability maximization and energy minimization.

We start from the original definition in Eq. (2.31) and get

$$p(\mathbf{Y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{i \in \mathcal{V}} \phi_i(y_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(y_i, y_j) \tag{2.32}$$

$$= \frac{1}{Z(\mathbf{x})} \prod_{i \in \mathcal{V}} \exp\left(-g_i(y_i)\right) \prod_{(i,j) \in \mathcal{E}} \exp\left(-f_{ij}(y_i, y_j)\right) \tag{2.33}$$

$$= \frac{1}{Z(\mathbf{x})} \exp\left(-\sum_{i \in \mathcal{V}} g_i(y_i)\right) \exp\left(-\sum_{(i,j) \in \mathcal{E}} f_{ij}(y_i, y_j)\right) \tag{2.34}$$

$$= \frac{1}{Z(\mathbf{x})} \exp\left(-\sum_{i \in \mathcal{V}} g_i(y_i) - \sum_{(i,j) \in \mathcal{E}} f_{ij}(y_i, y_j)\right) \tag{2.35}$$

$$= \frac{1}{Z(\mathbf{x})} \exp\left(-E(\mathbf{y}; \mathbf{x})\right), \tag{2.36}$$

where we have omitted the explicit dependence on $\mathbf{x}$ in the unary and pairwise potentials. The function

$$E(\mathbf{y}; \mathbf{x}) = \sum_{i \in \mathcal{V}} g_i(y_i) + \sum_{(i,j) \in \mathcal{E}} f_{ij}(y_i, y_j) \tag{2.37}$$

is called *energy function* and the exponential representation in Eq. (2.36) is a Gibbs[1] distribution [59] known from statistical mechanics. The term *energy* originates from mechanics and is commonly used in optimization literature as a synonym for an objective function. The functions $g_i(y_i)$ and $f_{ij}(y_i, y_j)$ correspond to the negative log-likelihood of the potential functions, *i.e.*

$$g_i(y_i) = -\log \phi_i(y_i), \qquad \phi_i(y_i) = \exp(-g_i(y_i)) \tag{2.38}$$

and

$$f_{ij}(y_i, y_j) = -\log \psi_{ij}(y_i, y_j), \qquad \psi_i(y_i, y_j) = \exp(-f_{ij}(y_i, y_j)), \tag{2.39}$$

respectively. Equipped with the result (2.36), we now show how we can compute the solution with highest probability in terms of energy minimization:

$$\begin{aligned} \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}^{|\mathcal{V}|}} p(\mathbf{y}|\mathbf{x}) &= \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}^{|\mathcal{V}|}} \frac{1}{Z(\mathbf{x})} \exp\left(-E(\mathbf{y}; \mathbf{x})\right) && (2.40) \\ &= \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}^{|\mathcal{V}|}} -E(\mathbf{y}; \mathbf{x}) && (2.41) \\ &= \operatorname*{arg\,min}_{\mathbf{y} \in \mathcal{Y}^{|\mathcal{V}|}} E(\mathbf{y}; \mathbf{x}), && (2.42) \end{aligned}$$

where we note that Eq. (2.42) is exactly an instance of the generic optimization problem introduced in Eq. (2.26). This result nicely shows the tight connection between energy minimization and probability maximization.

Relation between Probabilistic Perspective and Energy Minimization Perspective As we have shown in the previous section, the probabilistic and energy minimization perspective are closely related. However, there are also differences between them which we want to show here. The probabilistic interpretation provides in addition to the optimal solution also a confidence for this solution in terms of a probability. This is simply a result of the left hand side of Eq. (2.32) which actually defines the marginal distribution over the labels given the input, *i.e.* $p(\mathrm{Y}|\mathbf{x})$. This probability distribution not only allows to compute the *MAP* solution (shown in Eq. (2.40)) corresponding to the mode of the distribution, but we can also compute the expected value. This can be done for every pixel i as follows:

$$\mathbb{E}_{Y_i \sim p_i(\,\cdot\,|\mathrm{X}=\mathbf{x})}[Y_i] = \sum_{y \in \mathcal{Y}} y p_i(y|\mathbf{X} = \mathbf{x}), \tag{2.43}$$

which can be beneficial for some problems.

[1] Also called Boltzmann distribution

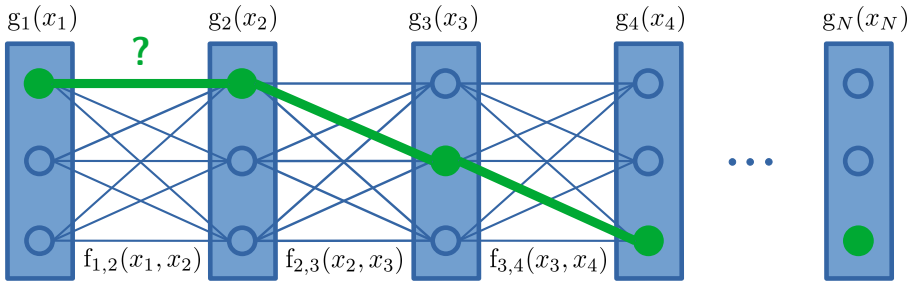


Figure 2.8: Chain sub-problem of an *MRF*. The nodes are visualized as vertical rectangles and contain here 3 circles representing the possible labels and the edges are represented as lines connecting neighboring nodes. The green path denotes the optimal path through the chain. It is the shortest path through the graph and corresponds to the *MAP* solution. Note that such a graph is also referred to as "Trellis" graph.

2.4.4 Inference Algorithms

Using the graphical models defined in Sections 2.4.1 and 2.4.2, we typically want to make predictions with these models. From now on we will apply all results to *CRFs*, but of course all presented algorithms can be also used for *MRFs*. In order to use our *CRF* to make predictions, we need to compute the marginal distribution for every node i by marginalizing out all nodes but node i. As a result we get marginal distributions $p(Y_i|\mathbf{x})$ capturing information of all other nodes given the graph structure. The task of computing this marginal distribution is referred to as *probabilistic inference*.

2.4.4.1 Dynamic Programming

Dynamic Programming (DP) is a standard algorithm invented by Bellman [7] to compute the shortest path through acyclic graphs. We derive the *DP* algorithms here on a chain graph similar as visualized in Fig. 2.8. To this end, let us assume we have given a chain graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of N nodes and $N-1$ edges connecting neighboring nodes. As usual, every node i can take labels form a label set $\mathcal{Y}$, *i.e.* $y_i \in \mathcal{Y}$. Using these definitions, we can quantify the cost of all possible paths with the energy

$$E(y_1, \ldots, y_N) = \sum_{i=1}^{N} g_i(y_i) + \sum_{i=1}^{N-1} f_{i,i+1}(y_i, y_{i+1}), \tag{2.44}$$

where the functions g_i quantify the cost of selecting one specific label for node i and the functions $f_{i,i+1}$ quantify the transition cost from node i to node $i+1$. Note also that Eq. (2.44) is just a special case of the energy shown in Eq. (2.37) used in a *CRF*. Given the energy defined in Eq. (2.44), we want to

1. compute the path through the graph yielding minimal cost and
2. the cost of the optimal path.

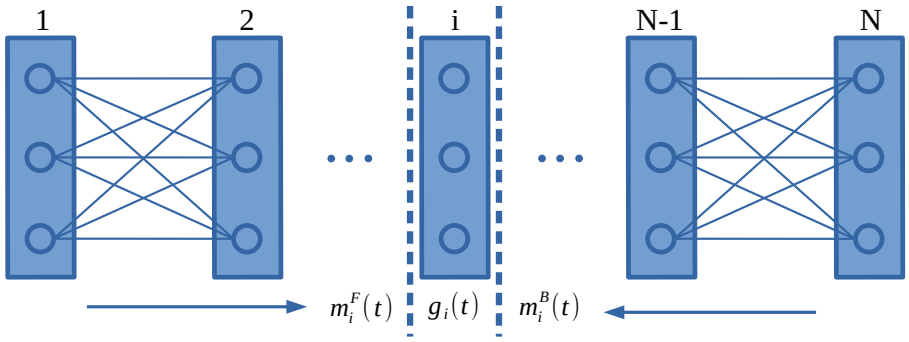


Figure 2.9: Visualization of the belief computation for node i. The nodes are enumerated from 1 to N. After the message passing the forward messages contain for every node the gathered information from node 1 to node i denoted by the arrow and the dashed vertical line at the left of node i. g_i denotes the unary costs and the backward messages hold the information from node N to i.

Thus, we have to solve the optimization problem

$$\min_{y_1,\dots,y_N} E(y_1,\dots,y_N). \tag{2.45}$$

By expanding Eq. (2.45), we see that we can split up the optimization problem into sub-problems and get

$$\min_{y_1,\dots,y_N \in \mathcal{Y}^N} E(y_1,\dots,y_N) = \min_{y_1,\dots,y_N \in \mathcal{Y}^N} \sum_{i=1}^{N} g_i(y_i) + \sum_{i=1}^{N-1} f_{i,i+1}(y_i, y_{i+1}) \tag{2.46}$$

$$= \min_{y_2,\dots,y_N \in \mathcal{Y}^{N-1}} \Big(\underbrace{\min_{y_1} g_1(y_1) + f_{1,2}(y_1, y_2)}_{m_2(y_2)} \Big) + \sum_{i=2}^{N} g_i(y_i) + \sum_{i=2}^{N-1} f_{i,i+1}(y_i, y_{i+1}) \tag{2.47}$$

$$= \min_{y_3,\dots,y_N \in \mathcal{Y}^{N-2}} \Big(\underbrace{\min_{y_2} m_2(y_2) + g_2(y_2) + f_{2,3}(y_2, y_3)}_{m_3(y_3)} \Big) + \sum_{i=3}^{N} g_i(y_i) + \sum_{i=3}^{N-1} f_{i,i+1}(y_i, y_{i+1}) \tag{2.48}$$

$$= \cdots = \min_{y_N \in \mathcal{Y}} m_N(y_N) + g_N(y_N), \tag{2.49}$$

where $m_i(y_i)$ is called the *(forward) message* sent from node $i-1$ to i. More compactly, we can write the *DP* algorithm as

$$m_{i+1}(t) = \min_{s \in \mathcal{Y}} m_i(s) + g_i(s) + f_{i,i+1}(s,t), \qquad 0 < i < N-1, \quad m_1(s) = 0, \tag{2.50}$$

where $s \in \mathcal{Y}$ is the label of the source node and $t \in \mathcal{Y}$ is the label of the target node. Applying Eq. (2.50) to a problem is also referred to as *message passing* [152] and the variables m_i are

Algorithm 1: Dynamic Programming on a Chain

Input: Unary costs g, pair-wise costs f
Result: Optimal path $\mathbf{y}^*$

// Propagate in chain direction
for $i \in \{1, 2, \dots N-1\}$ **do**
  $m_{i+1}^F(t) = \min_{s \in \mathcal{Y}} m_i^F(s) + g_i(s) + f_{i,i+1}(s,t)$; /* Eq. (2.50) */
end for

// Propagate in reverse chain direction
for $i \in \{N, N-1, \dots 2\}$ **do**
  $m_{i-1}^B(t) = \min_{s \in \mathcal{Y}} m_i^B(s) + g_i(s) + f_{i-1,i}(s,t)$; /* Eq. (2.50) */
end for

// Compute optimal path
for $i \in \mathcal{V}$ **do**
  $b_i(y_i) = g_i(y_i) + m_i^F(y_i) + m_i^B(y_i)$; /* Eq. (2.51) */
  $y_i^* = \arg\min_{y_i \in \mathcal{Y}} b_i(y_i)$; /* Eq. (2.52) */
end for
return $\mathbf{y}^*$

called *messages*. Note that the message passing also reflects the Markov property, *i.e.* that a specific node is only dependent on its direct neighbors.

In order to compute the optimal labeling we need to apply the *DP* algorithms shown in Eq. (2.50) twice, *i.e.* in chain direction and in reverse chain direction.

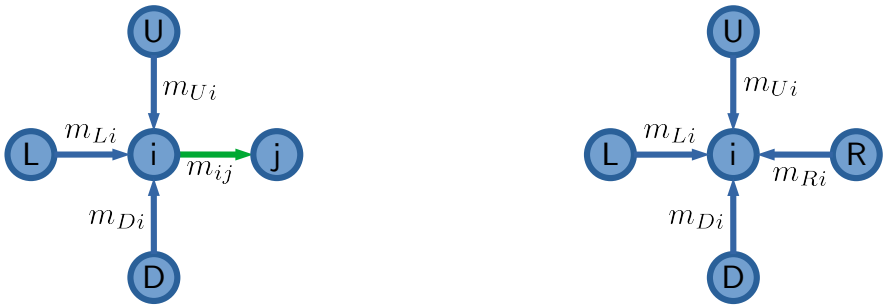
Let us denote the messages computed in chain direction as *forward messages* $m_i^F(t)$ and the messages computed in reverse chain direction as *backward messages* $m_i^B(t)$. The forward messages can be directly computed with Eq. (2.50). To compute the backward messages the chain is indexed in reverse order with indices $i \in \{N, N-1, \dots, 2\}$. Figure 2.9 shows which information is gathered by the forward and backward messages, respectively. For every node i, the forward messages contain the min-marginals until node i (without i), where the nodes $\{1, \dots, i-1\}$ have been marginalized out already. Similarly, the backward messages contain the min-marginals until node i (without i), where the nodes $\{N, \dots, i+1\}$ have been marginalized out already. Thus, in order to compute the min-marginals it remains to add the unary information to the forward and backward messages, *i.e.*

$$b_i(y_i) = g_i(y_i) + m_i^F(y_i) + m_i^B(y_i) \tag{2.51}$$

for every node i. The min-marginals are also often referred to as the more general term *beliefs*.

Next, we can use the min-marginals to compute the optimal path. This can be done by computing the element-wise Winner Takes All (WTA) solution

$$y_i^* = \arg\min_{y_i \in \mathcal{Y}} b_i(y_i), \tag{2.52}$$



(a) Message Passing (b) Belief Readout

Figure 2.10: Visualization of the update rules in the belief propagation algorithm. The green color indicates the send-direction of the message. Here, the message is sent from node i to node j. The uppercase letters L, R, U, D denote the Left, Right, Up and Down neighbors of node i.

where the asterisk in the super-script denotes the optimal labeling. Note that this variant of *DP* using the min-marginals does not require additional backtracking. All the information required to compute the optimal labeling is already captured in the min-marginals. Algorithm 1 shows the complete dynamic programming algorithm as defined in this section.

Depending on the task, the *DP* algorithm is also called *Viterbi algorithm* [198] in Hidden Markov Models or *Belief propagagtion on a chain* [168] in the probabilistic setting. The *DP* algorithm is exact for acyclic graphs such as trees and chains and the optimal solution can be computed with one forward-backward sweep through the graph. Note that the *DP* interpretation of the algorithm is different based on the task. In general, the algorithm can be used to compute the shortest path through a graph. However, this can also be interpreted as *e.g.* an optimal labeling in a classification task.

The *DP* algorithm cannot be used directly to perform inference in a *CRF* because it contains loops and is thus not an acyclic graph. Nevertheless, we will see later in Chapter 4 how the *DP* algorithm can be used as a generic building block for *CRF* inference.

2.4.4.2 (Loopy) Belief Propagation

Next we want to investigate belief propagation. We will first describe the belief propagation algorithms, max-product and sum-product belief propagation, in their most general form for grid graphs used in *CRFs* as defined in Eq. (2.31) and show how they can be used to perform inference in the *CRF*. It will turn out that belief propagation defines a whole family of algorithms on grid-graphs. Then, we will also show the relation to *DP* presented in the previous section and derive numerically stable algorithms suitable for implementation.

To get started, recall from Section 2.4.2 Conditional Random Fields that a *CRF* has nodes $i \in \mathcal{V}$, which are pair-wise connected with edges $(i, j) \in \mathcal{E}$. Every node can take labels from our label set, *i.e.* $y_i \in \mathcal{Y}$. Equipped with the notation and the setting, we can now define belief propagation.

Max-Product Belief Propagation Max-Product belief propagation consists of the two steps *message passing* and *belief readout*. The former is defined as

$$\tilde{m}_{ij}(y_j) = \max_{y_i \in \mathcal{Y}} \phi_i(y_i)\psi_{ij}(y_i, y_j) \prod_{n \in \mathcal{N}(i)\setminus j} \tilde{m}_{ni}(y_i), \tag{2.53}$$

where $\tilde{m}_{ij}$ denotes a message sent from node i to node j and $\mathcal{N}(i)$ is the set of neighboring nodes. Figure 2.10a shows a visualization of the message passing procedure. The latter, *i.e.* the belief readout, is defined as

$$b_i(y_i) = \frac{1}{\tilde{Z}_i} \phi_i(y_i) \prod_{n \in \mathcal{N}(i)} \tilde{m}_{ni}(y_i), \tag{2.54}$$

where b_i are called beliefs of node i and $\tilde{Z}_i$ is the normalization constant ensuring that the beliefs are a valid probability distribution. The belief readout can be interpreted as aggregating information from the neighboring nodes. Figure 2.10b shows a visualization of the belief readout. This specific belief propagation version shown in Eq. (2.53) is called *Max-Product Belief Propagation*. The name is simply derived from the fact that we take the **max**imum of a **product** in every message update step. The max-product algorithm can be used to compute the beliefs which correspond to *max-marginals* here. Note the similarity to the min-marginals we computed in Eq. (2.51). To get a final prediction out of the beliefs we can again simply compute the *WTA* solution, which corresponds to the *MAP* solution and is given by

$$y_i^* = \operatorname*{argmax}_{y_i \in \mathcal{Y}} b_i(y_i), \tag{2.55}$$

where the asterisk denotes the estimated solution. We cannot guarantee that we find the optimal solution for grid-graphs and it is known that inference in grid-graphs is an NP-hard problem [38]. Nevertheless, the algorithm works often very well in practice. Note that, in order to compute the *MAP* solution as done in Eq. (2.55), we do not need to compute the normalization constant Z_i because rescaling does not affect the minimizer in Eq. (2.55).

Log-Domain The belief propagation algorithm is usually not directly implemented using Eqs. (2.53) and (2.54) because the products therein cannot be computed in a numerically stable way. This is, because the functions ϕ and Ψ represent probabilities and a product of probabilities results in very small values. Propagating these small values yields tiny values and therefore the computation can become numerically unstable. To avoid this problem, the algorithm is usually implemented in the log domain. This allows to replace the products with summations, which can be implemented in a numerically stable way. Let us derive the log message updates for the

max-product belief propagation (2.53) first. We therefore compute the (negative) log messages

$$
\begin{aligned}
m_{ij}(y_j) &= -\log\left(\tilde{m}_{ij}(y_j)\right) && (2.56)\\
&= -\log\left(\max_{y_i \in \mathcal{Y}} \phi_i(y_i)\psi_{ij}(y_i, y_j) \prod_{n \in \mathcal{N}(i)\setminus j} \tilde{m}_{ni}(y_i)\right) && (2.57)\\
&= -\max_{y_i \in \mathcal{Y}} \log\left(\phi_i(y_i)\psi_{ij}(y_i, y_j) \prod_{n \in \mathcal{N}(i)\setminus j} \tilde{m}_{ni}(y_i)\right) && (2.58)\\
&= -\max_{y_i \in \mathcal{Y}} \log\left(\exp(-g_i(y_i)) \exp(-f_{ij}(y_i, y_j)) \prod_{n \in \mathcal{N}(i)\setminus j} \exp(-m_{ni}(y_i)\right) && (2.59)\\
&= -\max_{y_i \in \mathcal{Y}} \log\left(\exp\left(-g_i(y_i) - f_{ij}(y_i, y_j) - \sum_{n \in \mathcal{N}(i)\setminus j} m_{ni}(y_i)\right)\right) && (2.60)\\
&= -\max_{y_i \in \mathcal{Y}} -g_i(y_i) - f_{ij}(y_i, y_j) - \sum_{n \in \mathcal{N}(i)\setminus j} m_{ni}(y_i) && (2.61)\\
&= \min_{y_i \in \mathcal{Y}} g_i(y_i) + f_{ij}(y_i, y_j) + \sum_{n \in \mathcal{N}(i)\setminus j} m_{ni}(y_i). && (2.62)
\end{aligned}
$$

If we compare the result in Eq. (2.62) with Eq. (2.50), we see that we have derived the *DP* algorithm from a probabilistic viewpoint again. However, result (2.62) is slightly more general, because we did not specify any structure of the graph. Equation (2.62) is also referred to as *Min-Sum Belief Propagation* and Eq. (2.61) is also referred to as *Max-Sum Belief Propagation.*

The belief readout in the log domain is still missing. First, we observe that we are only interested in the *MAP* solution and thus we do not need to compute the normalization constant Z_i, because scaling does not change the result. Thus, we can rewrite Eq. (2.54) as

$$
\begin{aligned}
b_i(y_i) &= \frac{1}{\tilde{Z}_i} \phi_i(y_i) \prod_{n \in \mathcal{N}(i)} \tilde{m}_{ni}(y_i) && (2.63)\\
&\propto \phi_i(y_i) \prod_{n \in \mathcal{N}(i)} \tilde{m}_{ni}(y_i). && (2.64)
\end{aligned}
$$

Next, using result (2.64) we compute the approximate negative log beliefs with

$$
\begin{aligned}
-\log b_i(y_i) &\propto -\log\left(\phi_i(y_i) \prod_{n \in \mathcal{N}(i)} \tilde{m}_{ni}(y_i)\right) && (2.65)\\
&= -\log\left(\exp(-g_i(y_i)) \prod_{n \in \mathcal{N}(i)} \exp(-m_{ni}(y_i))\right) && (2.66)\\
&= g_i(y_i) + \sum_{n \in \mathcal{N}(i)} m_{ni}(y_i). && (2.67)
\end{aligned}
$$

Equation (2.67) corresponds to the (unnormalized) negative log likelihood. In this setting the lower the value the better the result. Therefore, we can compute the *MAP* solution using a node-wise

arg min operation, *i.e.*

$$y_i^* = \arg\min_{y_i \in \mathcal{Y}} \left\{ g_i(y_i) + \sum_{n \in \mathcal{N}(i)} m_{ni}(y_i) \right\}. \tag{2.68}$$

Implementation Details Although the log messages in Eq. (2.62) and the beliefs based on the log messages in Eq. (2.67) are fairly stable, there are still two more ingredients missing to make the whole algorithm stable. First, we need to add a normalization of the message updates. To this end, consider the min-sum message updates defined in Eq. (2.62), where the resulting message is computed by accumulating the unary cost, the pairwise cost and the neighborhood information. Depending on the length of the chain of the message passing and on the number of iterations of the algorithm, the messages will become larger and larger after every update. In order to avoid this problem, we need to normalize the messages and propagate the normalized messages instead of the original messages. Note that the normalization is a constant for every node and thus this constant will not change the final result. Therefore, we are free to choose an appropriate normalization. It turns out that it is beneficial to normalize the messages in Eq. (2.62) such that the minimum value is zero. This can be achieved by subtracting the minimum

$$z_i \coloneqq \min_{y_j \in \mathcal{Y}} m_{ij}(y_j) \tag{2.69}$$

from every component of the resulting message yielding the numerically stable message update

$$\overline{m}_{ij}(y_j) \coloneqq m_{ij}(y_j) - z_i. \tag{2.70}$$

Note that we also need to replace the original incoming messages $m_{ni}(y_i)$ by the normalized messages $\overline{m}_{ni}(y_i)$ in Eq. (2.62).

If we normalize the log messages defined in Eq. (2.62) with Eq. (2.69) and compute the result with Eq. (2.68), we eventually have a numerically stable version of the max-product belief propagation algorithm. A complete max-product algorithm applied to a grid-graph is shown in algorithms 2 and 4.

Sum-Product Belief Propagation Recall Eq. (2.2) from Section 2.1, where we have defined the sum-rule of probabilities which we can use to marginalize out specific random variables. Up to now we have been talking about "max-marginals" in Eq. (2.53) and "min-marginals" in Eq. (2.50), respectively, and not about (true) marginals. By replacing the `max` operation with a summation in the message update algorithm in Eq. (2.53), we can actually compute marginals as well.

Let us first derive this result on a chain problem containing N vertices. We know from probability theory that we can compute marginals by summing (=marginalizing) out all other variables (*c.f.* Eq. (2.2)). To derive an algorithm for our chain problem, we simply apply Eq. (2.2) to Eq. (2.31) where we are interested in computing the marginal distribution $p(y_k|\mathbf{x})$ for every

node k. This can be done with

$$p(y_k|\mathbf{x}) = \sum_{y_{-k}\in\mathcal{Y}^{N-1}} p_{-k}(y_{-k}|\mathbf{x}) \tag{2.71}$$

$$= \sum_{y_{-k}\in\mathcal{Y}^{N-1}} \left(\prod_{i=1}^{N}\phi_i(y_i)\prod_{i=1}^{N-1}\psi_{ij}(y_i, y_{i+1})\right) \tag{2.72}$$

$$= \phi_k(y_k) \sum_{y_{-k}\in\mathcal{Y}^{N-1}} \left(\prod_{i=1}^{N}\phi_i(y_i)\prod_{i=1}^{N-1}\psi_{i,i+1}(y_i, y_{i+1})\right) \tag{2.73}$$

$$= \phi_k(y_k) \underbrace{\sum_{y_{<k}\in\mathcal{Y}^{K-1}} \left(\prod_{i=1}^{K-1}\phi_i(y_i)\prod_{i=1}^{K-1}\psi_{i,i+1}(y_i, y_{i+1})\right)}_{(a)} \underbrace{\sum_{y_{>k}\in\mathcal{Y}^{N-K}} \left(\prod_{i=K+1}^{N}\phi_i(y_i)\prod_{i=K+1}^{N}\psi_{i-1,i}(y_{i-1}, y_i)\right)}_{(b)}, \tag{2.74}$$

where we have used the shortcuts $y_{-k} = \{y_1, y_2, \ldots, y_{k-1}, y_{k+1}, \ldots, y_N\}$, *i.e.* all indices except k, $y_{<k} = \{y_1, y_2, \ldots, y_{k-1}\}$, *i.e.* all indices less than k and $y_{>k} = \{y_{k+1}, y_{k+2}, \ldots, y_N\}$, *i.e.* all indices larger than k. Next we investigate the terms (a) and (b) to derive the recursive message update scheme. We can compute (a) recursively by starting from the first node with

$$(a) = \sum_{y_{1<k}\in\mathcal{Y}^{K-2}} \left[\underbrace{\left(\sum_{y_1\in\mathcal{Y}}\phi_1(y_1)\psi_{1,2}(y_1, y_2)\right)}_{\tilde{m}_2^F(y_2)} \left(\prod_{i=2}^{K-1}\phi_i(y_i)\prod_{i=2}^{K-1}\psi_{i,i+1}(y_i, y_{i+1})\right)\right] \tag{2.75}$$

$$= \sum_{y_{2<k}\in\mathcal{Y}^{K-3}} \left[\underbrace{\left(\sum_{y_2\in\mathcal{Y}}\tilde{m}_2^F(y_2)\phi_2(y_2)\psi_{2,3}(y_2, y_3)\right)}_{\tilde{m}_3^F(y_3)} \left(\prod_{i=3}^{K-1}\phi_i(y_i)\prod_{i=3}^{K-1}\psi_{i,i+1}(y_i, y_{i+1})\right)\right] \tag{2.76}$$

$$= \cdots = \sum_{y_{K-1}\in\mathcal{Y}} \tilde{m}_{K-1}^F(y_{K-1})\phi_{K-1}(y_{K-1})\psi_{K-1,K}(y_{K-1}, y_K), \tag{2.77}$$

where $y_{i<k} = \{y_i, y_{i+1}, \ldots y_{k-1}\}$. Similarly, we can compute the term (b) by starting the recursion

from the last node N going backwards with

$$(b) = \sum_{y_{N-1>K+1} \in \mathcal{Y}^{K-N-1}} \Bigg(\underbrace{\sum_{y_N \in \mathcal{Y}} \phi_N(y_N)\psi_{N-1,N}(y_{N-1}, y_N)}_{\tilde{m}^B_{N-1}(y_{N-1})} \Bigg) \Bigg(\prod_{i=K+1}^{N-1} \phi_i(y_i) \prod_{i=K+1}^{N-1} \psi_{i-1,i}(y_{i-1}, y_i) \Bigg) \tag{2.78}$$

$$= \sum_{y_{N-2>K+1} \in \mathcal{Y}^{K-N-2}} \Bigg(\underbrace{\sum_{y_{N-1} \in \mathcal{Y}} \tilde{m}^B_{N-1}(y_{N-1})\phi_{N-1}(y_{N-1})\psi_{N-2,N-1}(y_{N-2}, y_{N-1})}_{\tilde{m}^B_{N-2}(y_{N-2})} \Bigg) \cdot \tag{2.79}$$

$$\cdot \Bigg(\prod_{i=K+1}^{N-2} \phi_i(y_i) \prod_{i=K+1}^{N-2} \psi_{i-1,i}(y_{i-1}, y_i) \Bigg) \tag{2.80}$$

$$= \cdots = \sum_{y_{K+1} \in \mathcal{Y}} \tilde{m}^B_{K+1}(y_{K+1})\phi_{K+1}(y_{K+1})\psi_{K+1,K}(y_{K+1}, y_K). \tag{2.81}$$

The messages $\tilde{m}^F_i$ and $\tilde{m}^B_i$ are called forward and backward messages, respectively. They are also often referred to as *forward marginals* and *backward marginals*, respectively, because they marginalize out all preceding and succeeding nodes. Using the forward or backward marginals, respectively, we get the recursive message update equation

$$\tilde{m}_{ij}(y_j) = \sum_{y_i \in \mathcal{Y}} \phi_i(y_i)\psi_{ij}(y_i, y_j) \prod_{n \in \mathcal{N}(i) \setminus j} \tilde{m}_{ni}(y_i), \tag{2.82}$$

where the summation is used to marginalize out specific random variables (*c.f.* Eq. (2.2)). Note that we also slightly generalized Eq. (2.82) compared to the messages derived in Eqs. (2.74) and (2.81), since we now have messages from node i to node j capturing both directions and a product capturing neighborhood information from arbitrarily many directions. Using the same naming scheme as before, we see that the specific belief propagation version shown in Eq. (2.82) is called *Sum-Product Belief Propagation*, because the elementary message updates are computed as the **sum** of a **product**. The belief readout handles the left-over term $\phi_k(y_k)$ and can be computed with Eq. (2.54). The beliefs in the sum-product algorithm correspond indeed to the (true) marginals.

Log-Domain Similar as in the max-product belief propagation also the sum-product belief propagation algorithm is numerically unstable. Therefore, we can again represent the sum-product message passing in the (negative) log-domain to get a numerically stable version. This can be

done with

$$
\begin{aligned}
m_{ij}(y_j) &= -\log \tilde{m}_{ij}(t) && (2.83)\\
&= -\log\left(\sum_{y_i \in \mathcal{Y}} \phi_i(y_i)\psi_{ij}(y_i, y_j) \prod_{n \in \mathcal{N}(i)\setminus j} \tilde{m}_{ni}(y_i)\right) && (2.84)\\
&= -\log\left(\sum_{y_i \in \mathcal{Y}} \exp(-g_i(y_i)) \exp(-f_{ij}(y_i, y_j)) \prod_{n \in \mathcal{N}(i)\setminus j} \exp(-m_{ni}(y_i))\right) && (2.85)\\
&= -\log\left(\sum_{y_i \in \mathcal{Y}} \exp\Bigg(\underbrace{-g_i(y_i) - f_{ij}(y_i, y_j) - \sum_{n \in \mathcal{N}(i)\setminus j} m_{ni}(y_i)}_{-e_{ij}(y_i, y_j)}\Bigg)\right), && (2.86)
\end{aligned}
$$

where $e_{ij}(y_i, y_j)$ is the energy augmented with the messages of the neighbors and we got rid of the product.

The last missing part is the computation of the beliefs based on the (negative) log-message. We therefore insert Eq. (2.67) into Eq. (2.54) and get

$$
b_i(y_i) = \frac{\exp\left(-g_i(y_i) - \sum_{n \in \mathcal{N}} m_{ni}(y_i)\right)}{\sum_{y_i' \in \mathcal{Y}} \exp\left(-g_i(y_i') - \sum_{n \in \mathcal{N}} m_{ni}(y_i')\right)}, \tag{2.87}
$$

where the denominator corresponds to the normalization constant Z_i in Eq. (2.54) and ensures that the beliefs are a valid probability distribution.

Implementation Details In difference to the log version of the max-product algorithm (Eq. (2.62)), we end up with a summation of exponents in Eq. (2.86) which is also known to be numerically unstable if the values in the exponent are large. Luckily, we can apply the so-called *log-sum-exp trick* to Eq. (2.86). To this end, we set $x_i = -e_{ij}(y_i, y_j)$ to and get

$$
\begin{aligned}
\log\left(\sum_{i=1}^{N} \exp(x_i)\right) &= \log\left(\sum_{i=1}^{N} \exp(x_i - z + z)\right) && (2.88)\\
&= \log\left(\sum_{i=1}^{N} \exp(x_i - z) \exp(z)\right) && (2.89)\\
&= \log\left(\exp(z) \sum_{i=1}^{N} \exp(x_i - z)\right) && (2.90)\\
&= \log\left(\exp(z)\right) + \log\left(\sum_{i=1}^{N} \exp(x_i - z)\right) && (2.91)\\
&= z + \log\left(\sum_{i=1}^{N} \exp(x_i - z)\right) && (2.92)
\end{aligned}
$$

where

$$z \coloneqq \max_i x_i. \tag{2.93}$$

The transformation in the log-sum-exp trick given in Eq. (2.92) ensures that the maximal argument entering the exponential function is 0. If we apply the log-sum-exp trick given in Eq. (2.92) to the negative log-messages of the sum-product algorithm in Eq. (2.86), we get

$$m_{ij}(y_j) = z_j - \log\left(\sum_{y_i \in \mathcal{Y}_i} \exp\left(-e_{ij}(y_i, y_j) + z_j\right)\right) \tag{2.94}$$

$$= z_j - \log\left(\sum_{y_i \in \mathcal{Y}_i} \exp\left(-g_i(y_i) - f_{ij}(y_i, y_j) - \sum_{n \in \mathcal{N}(i)\setminus j} m_{ni}(y_i) + z_j\right)\right) \tag{2.95}$$

and have a numerically stable message update. However, similar as in the max-product algorithm, we need to normalize the message during propagation. Since we are free to choose an arbitrary normalization, we can simply subtract the minimum z_j and get

$$\overline{m}_{ij}(y_j) \coloneqq m_{ij}(y_j) - z_j \tag{2.96}$$

and now have a numerically stable message passing procedure. Note that Eq. (2.96) shows that we can simply omit adding the minimum z_j in Eq. (2.95). Since we need to compute this minimum already for the log-sum-exp trick, this type of normalization is beneficial compared to *e.g.* normalizing the message to have zero mean which is commonly done in the literature [143].

Also the belief readout in Eq. (2.87) can be numerically unstable. We can stabilize the readout again by computing the exponents robustly with

$$b_i(y_i) = \frac{\exp\left(-g_i(y_i) - \sum_{n \in \mathcal{N}} m_{ni}(y_i)\right)}{\sum_{y_i' \in \mathcal{Y}} \exp\left(-g_i(y_i') - \sum_{n \in \mathcal{N}} m_{ni}(y_i')\right)} \tag{2.97}$$

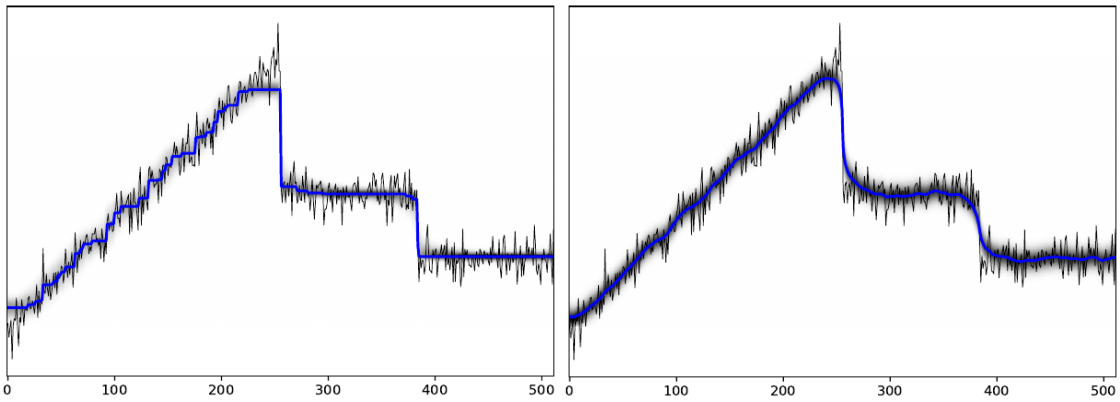
$$= \frac{\exp\left(-g_i(y_i) - \sum_{n \in \mathcal{N}} m_{ni}(y_i) - z_i + z_i\right)}{\sum_{y_i' \in \mathcal{Y}} \exp\left(-g_i(y_i') - \sum_{n \in \mathcal{N}} m_{ni}(y_i') - z_i + z_i\right)} \tag{2.98}$$

$$= \frac{\exp(z_i) \exp\left(-g_i(y_i) - \sum_{n \in \mathcal{N}} m_{ni}(y_i) - z_i\right)}{\exp(z_i) \sum_{y_i' \in \mathcal{Y}} \exp\left(-g_i(y_i') - \sum_{n \in \mathcal{N}} m_{ni}(y_i') - z_i\right)} \tag{2.99}$$

$$= \frac{\exp\left(-g_i(y_i) - \sum_{n \in \mathcal{N}} m_{ni}(y_i) - z_i\right)}{\sum_{y_i' \in \mathcal{Y}} \exp\left(-g_i(y_i') - \sum_{n \in \mathcal{N}} m_{ni}(y_i') - z_i\right)}, \tag{2.100}$$

where the constant

$$z_i = \max_{y_i \in \mathcal{Y}} \left\{-g_i(y_i) - \sum_{n \in \mathcal{N}} m_{ni}(y_i)\right\} \tag{2.101}$$



(a) Maximization (Max-Product) (b) Marginalization (Sum-Product)

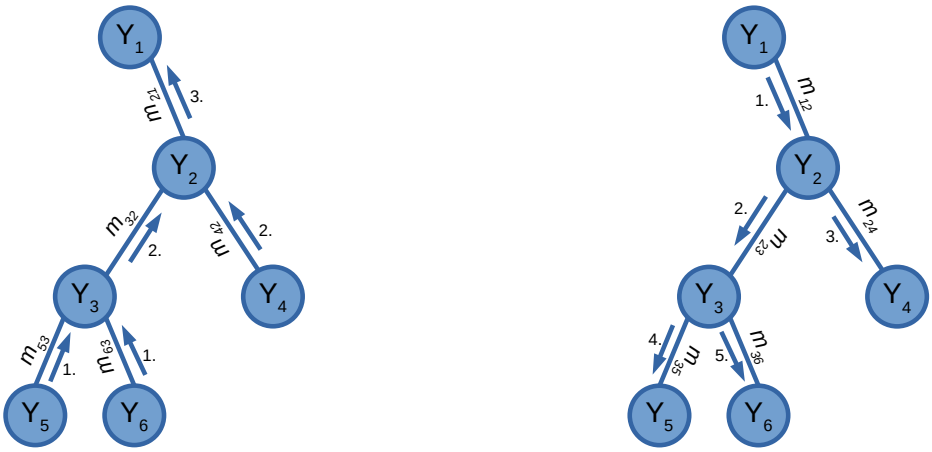
Figure 2.11: Comparison between the max-product and the sum-product algorithm. The black line shows the noisy data, the gray shading shows the max-marginals and marginals, respectively, and the blue line shows the result of the algorithms.

ensures that the maximal number entering the exponent is 0. If we now use Eqs. (2.96) and (2.100) we have a numerically stable implementation for the sum-product algorithm. The complete stable algorithm applied to a grid-graph is shown in algorithms 3 and 5.

Figure 2.11 shows a comparison of the two presented versions in this section. The result of the max-product algorithm is always a discrete label and yields sharp jumps at the discontinuities. The sum-product algorithm can also result in continuous values and therefore the result is smoother, but the jumps are often over-smoothed.

Message Update Schedules Both the max-product and sum-product algorithm are defined very generically in that they do not directly specify a message update scheme. Hence, to apply these algorithms to an actual problem, we first need to define a message update schedule.

Undirected Acyclic Graphs Let us first start with undirected acyclic graphs such as trees and chains. Undirected acyclic graphs have the property that two vertices are connected by exactly one path. In this setting Eq. (2.53) defines the complete algorithm. This can be seen through the message passing defined in Eq. (2.53), where we note that the current message is dependent on previous messages corresponding to previous nodes. Thus, in order to start the algorithm, we need to find a node which does not depend on any other node. As shown in Fig. 2.12 nodes fulfilling this requirement are leaf-nodes and the root-node. Similar as in *DP* we need to make two passes through the graph. First, we start the message passing from a leaf node and propagate the messages to the root node (visualized in Fig. 2.12a). Second, we do exactly the opposite and start at the root node and propagate the messages towards the leaf nodes (visualized in Fig. 2.12b). The beliefs can then be simply computed with Eq. (2.54). Note that the procedure described above yields the optimal solution, similar as in *DP*. An example, where belief propagation is used in practice, is the pictorial structures model [53].



(a) Leaf to root messages

(b) Root to leaf messages

Figure 2.12: Distinct ordering of message updates on a tree graph. First the messages are propagated from the leaf nodes to the root note shown in Fig. 2.12a and second the messages are propagated from the root node to the leaves shown in Fig. 2.12b. Same numbers denote that both messages are required by the receiving node.

Grid Graphs In this thesis, we are mainly interested in grid graphs containing loops as *e.g.* in *CRFs*. In difference to trees and chains, we cannot find any node not depending on any other node in grid graphs. This is simply because such a node does not exist (*c.f.* Fig. 2.3). Nevertheless, we can come up with an approximate inference algorithm by defining an update schedule. This yields an iterative algorithm which we can use to perform approximate inference in the *CRF*. Such an algorithm is referred to as *Loopy Belief Propagation* in the literature [136]. We also want to emphasize that we can actually come up with several update schemes and depending on the specific schedule we also get different loopy belief propagation algorithms with different properties.

Now we are ready to show two commonly used schedules which can be applied to grid graphs [186]. These are the *synchronous* update schedule and the *asynchronous* update schedule. Let us start with the synchronous update schedule, where all messages are updated in parallel. This algorithm is summarized for the min-sum belief propagation variant in algorithm 2 and for the sum-product belief propagation variant in algorithm 3, where we have additionally normalized the messages after each update step. The synchronous loopy belief propagation version updates all neighboring nodes using the current message states. This is easy to implement, but it can take many iterations until the algorithm converges. However, a drawback of the synchronous schedule is that information is only transported over one node in one iteration. Thus, if we have an image with a width of *e.g.* 1000 pixels, we need to perform at least 1000 iterations to ensure that information is transported over the whole width of the image.

An alternative schedule is the asynchronous schedule, where one message is updated at a time. This schedule is also known as left-right-up-down belief propagation [186], BP-M [185] or sweep belief propagation [88], because the update schedule performs sweeps over the whole

Algorithm 2: Synchronous Max-Product Loopy Belief Propagation

Input: Unary probabilities g, pair-wise probabilities f
Result: Negative log beliefs $-\log b$

```
// Message Passing
```
for $0 \leq k \leq K$ **do**
  for $i \in \mathcal{V}$ **do**
    for $d \in \{L, R, U, D\}$ **do**
      `// Compute message and normalization`
      $m_{id}^{k+1}(y_d) = \min_{y_i \in \mathcal{Y}} g_i(y_i) + f_{id}(y_i, y_d) + \sum_{n \in \mathcal{N}(i) \setminus j} \overline{m}_{ni}^{k}(y_i)$; `/* Eq. (2.62) */`
      $z_d = \min_{y_d \in \mathcal{Y}} m_{id}^{k+1}(y_d)$; `/* Eq. (2.69) */`
      `// Update normalized message for neighbor` d
      $\overline{m}_{id}^{k+1}(y_j) = m_{id}^{k+1}(y_d) - z_d$; `/* Eq. (2.70) */`
    end for
  end for
end for

```
// Belief Readout
```
for $i \in \mathcal{V}$ **do**
  $-\log b_i(y_i) = g_i(y_i) + \sum_{n \in \mathcal{N}(i)} m_{ni}(y_i)$; `/* Eq. (2.67) */`
end for
return $-\log b$

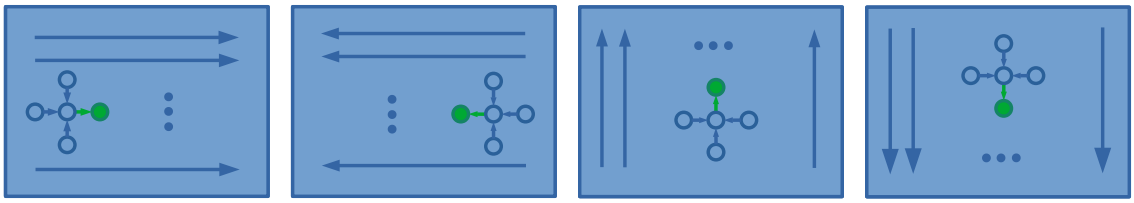


Figure 2.13: Left-Right-Up-Down Belief Propagation on a 4-connected grid-graph. The long arrows show the sweep direction of the algorithm, the green node is the node receiving the new message from the blue center node and its neighbors.

width and height of an image, respectively. Figure 2.13 shows a visualization of the sweeps for a 4-connected grid graph. These sweeps are indeed a great advantage because (i) in this setting information is transported over the whole width or height of an image in one iteration and (ii) the chain sub-problems can be solved exactly with *DP* which yields faster convergence of the message updates. A robust implementation of the max-product and sum-product algorithm is shown in algorithms 4 and 5, respectively. Similar as in the asynchronous update schedule, the messages are normalized before they are propagated through the chain. Note that from an implementation

Algorithm 3: Synchronous Sum-Product Loopy Belief Propagation

Input: Unary probabilities g, pair-wise probabilities f
Result: Beliefs $b \approx$ marginals

```
// Message Passing
```
for $0 \leq k \leq K$ **do**
  for $i \in \mathcal{V}$ **do**
    for $d \in \{L, R, U, D\}$ **do**
      `// Update message for neighbor` d
      $e^k_{id}(y_i, y_d) = g_i(y_i) + f_{id}(y_i, y_d) + \sum_{n \in \mathcal{N}(i) \setminus d} \overline{m}^k_{ni}(y_i)$; `/* Eq. (2.62) */`
      $z_d = \min_{y_i \in \mathcal{Y}} e^k_{id}(y_i, y_d)$; `/* Eq. (2.93) */`
      $\overline{m}^{k+1}_{id}(y_d) = -\log\left(\sum_{y_i \in \mathcal{Y}} \exp\left(-e^k_{id}(y_i, y_d) + z_d\right)\right)$; `/* Eq. (2.96) */`
    end for
  end for
end for

```
// Belief Readout
```
for $i \in \mathcal{V}$ **do**
  $\log b_i(y_i) = -g_i(y_i) - \sum_{n \in \mathcal{N}} m_{ni}(y_i)$; `/* Eq. (2.67) */`
  $z_i = \max_{y_i \in \mathcal{Y}_i} \log b_i(y_i)$; `/* Eq. (2.101) */`
  $b_i(y_i) = \frac{\exp\left(\log b_i(y_i) - z_i\right)}{\sum_{y'_i \in \mathcal{Y}} \exp\left(\log b_i(y'_i) - z_i\right)}$; `/* Eq. (2.100) */`
end for
return b

point of view the two different algorithms, synchronous and asynchronous belief propagation, differ only in the order of the `for` loops. In the synchronous version the outer loop is over the nodes and the inner loop is over the directions and in the asynchronous version it is exactly the opposite. There, the outer loop is over the directions and the inner loop is over the nodes resulting in sweeps through chain sub-problems.

Practical examples, where belief propagation is used on grid graphs are shown in [54, 171, 183].

Summary and Outlook We have seen the generic formulation of belief propagation and how we can apply belief propagation to undirected acyclic models and grid graphs such as *CRFs*. For the former, we can compute the exact solution with two sweeps through the graph. The situation is different for grid graphs. First, the inference problem in grid graphs is known to be NP-hard. Second, the generic belief propagation formulation defines a whole family of approximate inference algorithms, where the message update schedule defines the actual algorithm. However,

Algorithm 4: Asynchronous Max-Product Loopy Belief Propagation

Input: Unary probabilities g, pair-wise probabilities f
Result: Negative log beliefs $-\log b$

```
// Message Passing
```
for $0 \leq k \leq K$ **do**
  for $d \in \{L, R, U, D\}$ **do**
    for $i \in \mathcal{V}_d$ *on chain sub-graph in direction* d **do**
      `// Compute message and normalization`
      $m_{i,i+1}^{k+1}(y_{i+1}) = \min_{y_i \in \mathcal{Y}} g_i(y_i) + f_{i,i+1}(y_i, y_{i+1}) + \sum_{n \in \mathcal{N}(i) \setminus j} \overline{m}_{ni}^{k}(y_i)$; `/* Eq. (2.62) */`
      $z_{i+1} = \min_{y_{i+1} \in \mathcal{Y}} m_{i,i+1}^{k+1}(y_{i+1})$; `/* Eq. (2.69) */`
      `// Propagate message in direction` d
      $\overline{m}_{i,i+1}^{k+1}(y_{i+1}) = m_{i,i+1}^{k+1}(y_{i+1}) - z_{i+1}$; `/* Eq. (2.70) */`
    end for
  end for
end for

```
// Belief Readout
```
for $i \in \mathcal{V}$ **do**
  $-\log b_i(y_i) = g_i(y_i) + \sum_{n \in \mathcal{N}(i)} m_{ni}(y_i)$; `/* Eq. (2.67) */`
end for
return $-\log b$

it should be noted that independent of the message update schedule the derived belief propagation algorithms do not come with any convergence guarantees. It might for example happen that the beliefs jump between two states yielding a bad approximation of the (max-)marginals. However, especially the max-product belief propagation algorithm performs very well empirically and is thus one of the mostly used algorithms for approximate inference in grid graphs.

More information on message schedules can be found *e.g.* in [51]. Alternative message passing algorithms with convergence guarantees are *e.g.* the tree-reweighted message passing algorithm, TRW-S, by Kolmogorov [94], the generalized max-product belief propagation by Sontag et al. [182] and the max-sum diffusion algorithm by Werner [207]. Connections to the Bethe free energy are shown in *e.g.* [67, 212, 215]. An excellent overview of inference in graphical models including connections between graphical models, exponential families and variational inference is given in [202]. Meltzer et al. [130] shows a unified overview of message passing algorithms. In the next section we will derive another inference algorithm for graphical models based on dual decomposition.

Algorithm 5: Asynchronous Sum-Product Loopy Belief Propagation

Input: Unary probabilities g, pair-wise probabilities f
Result: Beliefs $b \approx$ marginals

// Message Passing
for $0 \leq k \leq K$ **do**
  for $d \in \{L, R, U, D\}$ **do**
    for $i \in \mathcal{V}_d$ *on chain sub-graph in direction* d **do**
      // Propagate message in direction d
      $e^k_{i,i+1}(y_i, y_{i+1}) = g_i(y_i) + f_{i,i+1}(y_i, y_{i+1}) + \sum_{n \in \mathcal{N}(i) \setminus i+1} \overline{m}^k_{ni}(y_i)$; /* Eq. (2.62) */
      $z_{i+1} = \min_{y_i \in \mathcal{Y}} e^k_{i,i+1}(y_i, y_{i+1})$; /* Eq. (2.93) */
      $\overline{m}^{k+1}_{i,i+1}(y_{i+1}) = -\log\left(\sum_{y_i \in \mathcal{Y}} \exp\left(-e^k_{i,i+1}(y_i, y_{i+1}) + z_{i+1}\right)\right)$; /* Eq. (2.96) */
    end for
  end for
end for

// Belief Readout
for $i \in \mathcal{V}$ **do**
  $\log b_i(y_i) = -g_i(y_i) - \sum_{n \in \mathcal{N}} m_{ni}(y_i)$; /* Eq. (2.67) */
  $z_i = \max_{y_i \in \mathcal{Y}} \log b_i(y_i)$; /* Eq. (2.101) */
  $b_i(y_i) = \frac{\exp\left(\log b_i(y_i) - z_i\right)}{\sum_{y'_i \in \mathcal{Y}} \exp\left(\log b_i(y'_i) - z_i\right)}$; /* Eq. (2.100) */
end for
return b

2.4.4.3 Dual Decomposition

The idea in dual (or Lagrangian) decomposition is to split up a difficult optimization problem into two simpler optimization problems. Consider the generic primal optimization problem

$$\min_{\mathbf{x} \in C} P(\mathbf{x}) \coloneqq f_1(\mathbf{x}) + f_2(\mathbf{x}), \tag{2.102}$$

where $C \subseteq \mathbb{R}^N$ is a convex set and f_1 and f_2 are two functions depending on the input $\mathbf{x}$. To motivate the decomposition, consider an instance of Eq. (2.102) where the composite problem $\min_{\mathbf{x}} f_1(\mathbf{x}) + f_2(\mathbf{x})$ is hard to solve, but the individual problems $\min_{\mathbf{x}} f_1(\mathbf{x})$ and $\min_{\mathbf{x}} f_2(\mathbf{x})$, respectively, are easy to solve. We can now rewrite the unconstrained original problem shown in Eq. (2.102) to the

equivalent constrained optimization problem

$$\min_{\mathbf{x}_1,\mathbf{x}_2 \in C} f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2) \tag{2.103}$$

$$\text{s.t. } \mathbf{x}_1 = \mathbf{x}_2, \tag{2.104}$$

where the equality constraint Eq. (2.104) ensures equivalence to the initial problem. Starting from Eq. (2.104) we compute the relaxation

$$\min_{\substack{\mathbf{x}_1,\mathbf{x}_2 \in C \\ \mathbf{x}_1 = \mathbf{x}_2}} f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2) = \min_{\mathbf{x}_1,\mathbf{x}_2 \in C} f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2) + \delta_{\mathbf{x}_1=\mathbf{x}_2}(\mathbf{x}_1, \mathbf{x}_2) \tag{2.105}$$

$$\geq \min_{\mathbf{x}_1,\mathbf{x}_2 \in C} \underbrace{f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2) + \langle \boldsymbol{\lambda}, \mathbf{x}_1 - \mathbf{x}_2 \rangle}_{L(\mathbf{x}_1,\mathbf{x}_2,\boldsymbol{\lambda})} \tag{2.106}$$

$$= \min_{\mathbf{x}_1 \in C} f_1(\mathbf{x}_1) + \langle \boldsymbol{\lambda}, \mathbf{x}_1 \rangle + \min_{\mathbf{x}_2} f_2(\mathbf{x}_2) - \langle \boldsymbol{\lambda}, \mathbf{x}_2 \rangle \tag{2.107}$$

which is known as the Lagrange decomposition, since we relaxed the indicator function $\delta_{\mathbf{x}_1=\mathbf{x}_2}$ by introducing a Lagrange multiplier $\boldsymbol{\lambda}$ for the equality constraint $\mathbf{x}_1 = \mathbf{x}_2$. The function $L(\mathbf{x}_1, \mathbf{x}_2, \boldsymbol{\lambda})$ is known as the Lagrangian. We see that we have actually split up the composite optimization problem into two independent optimization problems in Eq. (2.107). In order to find the best lower bound of (2.107), we need to solve the optimization problem

$$\max_{\boldsymbol{\lambda}} \min_{\mathbf{x}_1,\mathbf{x}_2 \in C} L(\mathbf{x}_1, \mathbf{x}_2, \boldsymbol{\lambda}) = \max_{\boldsymbol{\lambda}} \left(\min_{\mathbf{x}_1 \in C} f_1(\mathbf{x}_1) + \langle \boldsymbol{\lambda}, \mathbf{x}_1 \rangle + \min_{\mathbf{x}_2 \in C} f_2(\mathbf{x}_2) - \langle \boldsymbol{\lambda}, \mathbf{x}_2 \rangle \right), \tag{2.108}$$

where we need to minimize for the primal variables $\mathbf{x}_1$ and $\mathbf{x}_2$ and maximize with respect to the dual variable $\boldsymbol{\lambda}$.

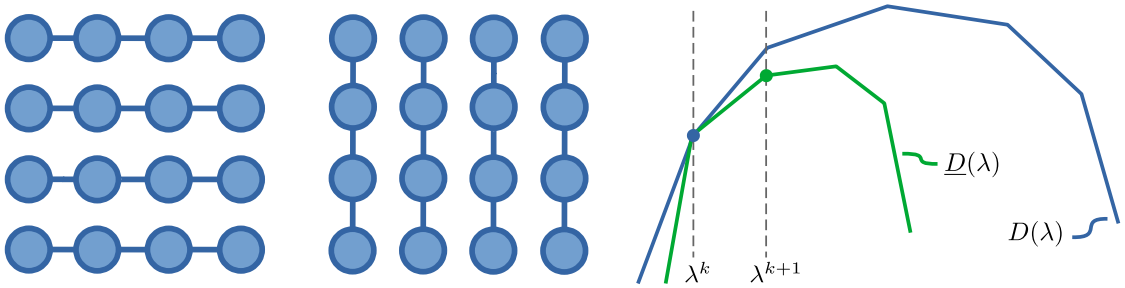
Recall our initial motivation for the decomposition. We argued thereby that the individual problems are easy to solve and the composite problem is hard to solve. The advantage of the new optimization problem in Eq. (2.107) is that we have actually achieved to decouple the two functions f_1 and f_2. These functions are now sub-problems in Eq. (2.108) which are easy to solve. Let us denote the solutions of these easy problems in the primal variables by $\mathbf{x}_1^*$ and $\mathbf{x}_2^*$. Inserting these solutions into the Lagrangian we get the dual problem

$$D(\boldsymbol{\lambda}) = f_1(\mathbf{x}_1^*) + \langle \boldsymbol{\lambda}, \mathbf{x}_1^* \rangle + f_2(\mathbf{x}_2^*) - \langle \boldsymbol{\lambda}, \mathbf{x}_2^* \rangle \tag{2.109}$$

$$= -\left(f_1^*(-\boldsymbol{\lambda}) + f_2^*(\boldsymbol{\lambda}) \right), \tag{2.110}$$

where the functions f_1^* and f_2^*, respectively, are the convex conjugate functions as defined in Eq. (2.125). The function $D(\boldsymbol{\lambda})$ is known as the *Lagrange decomposition based dual*, which is a concave function and non-smooth in general [168]. Next, we combine Eq. (2.108) and Eq. (2.110) to see that it remains to solve the maximization problem

$$\max_{\boldsymbol{\lambda}} D(\boldsymbol{\lambda}) \tag{2.111}$$



(a) Horizontal Sub-Problem (b) Vertical Sub-Problem (c) Minorant

Figure 2.14: Decomposition and minorant used in Dual Minorize-Maximize (DMM). Figs. 2.14a and 2.14b shows a decomposition of Fig. 2.3. The *DMM* algorithm exploits the fact that these sub-problems can be easily solved and parallelized on the *GPU*. Fig. 2.14c shows a minorant for the dual problem.

in the dual variable $\boldsymbol{\lambda}$. This problem can for example be solved with the sub-gradient ascent algorithm as proposed by Komodakis et al. [96]. In the next section, we will see another possibility to tackle Eq. (2.111) which can be highly parallelized on the Graphics Processing Unit (GPU).

Dual Minorize-Maximize Algorithm The Dual Minorize-Maximize (DMM) algorithm [178] brings us back to *MAP* inference problems in *CRFs*. This algorithm is a massively parallel algorithm specifically designed to be used on *GPUs*. The *DMM* algorithm operates on the Lagrange decomposition based dual defined in Eq. (2.111). To show how the algorithm works we rewrite Eq. (2.111) to

$$\max_{\boldsymbol{\lambda}} D(\boldsymbol{\lambda}) = \max_{\boldsymbol{\lambda}} -\left(f_1^*(-\boldsymbol{\lambda}) + f_2^*(\boldsymbol{\lambda})\right) \tag{2.112}$$

$$= \max_{\boldsymbol{\lambda}} \big(\underbrace{-f_1^*(-\boldsymbol{\lambda})}_{D_1(\boldsymbol{\lambda})}\big) + \big(\underbrace{-f_2^*(\boldsymbol{\lambda})}_{D_2(\boldsymbol{\lambda})}\big) \tag{2.113}$$

$$= \max_{\boldsymbol{\lambda}} D_1(\boldsymbol{\lambda}) + D_2(\boldsymbol{\lambda}), \tag{2.114}$$

where the two sub-problems correspond to a splitting of the original graph shown in Fig. 2.3 into horizontal and vertical sub-problems visualized in Figs. 2.14a and 2.14b, respectively. In every iteration of the algorithm, we need to compute a minorant of the dual functions D_1 and D_2, respectively. These minorants are lower bounds on the original problem, *i.e.*

$$\underline{D}^{\{1,2\}}(\boldsymbol{\lambda}) \leq D^{\{1,2\}}(\boldsymbol{\lambda}) \tag{2.115}$$

and exact in the current point $\boldsymbol{\lambda}^k$,*i.e.*

$$\underline{D}^{\{1,2\}}(\boldsymbol{\lambda}^k) = D^{\{1,2\}}(\boldsymbol{\lambda}^k). \tag{2.116}$$

Algorithm 6: Dual Minorize-Maximize

Input: Initial dual varaiables $\boldsymbol{\lambda}^0$
Result: Optimal dual variables $\boldsymbol{\lambda}^*$
for $k \geq 0$ **do**
  $\boldsymbol{\lambda}^{k+\frac{1}{2}} = \max_{\boldsymbol{\lambda}} \underline{D}^{1,k}(\boldsymbol{\lambda}) + D^2(\boldsymbol{\lambda})$
  $\boldsymbol{\lambda}^{k+1} = \max_{\boldsymbol{\lambda}} D^1(\boldsymbol{\lambda}) + \underline{D}^{2,k}(\boldsymbol{\lambda})$
end for
return $\boldsymbol{\lambda}^*$

Figure 2.14c shows a visualization of a suitable minorant. We are interested in maximal modular minorants $\varphi(\mathbf{x}) = \boldsymbol{\varphi}^\top \mathbf{x}$ fulfilling

$$\boldsymbol{\varphi}^\top \mathbf{x} \leq f(\mathbf{x}), \quad \forall \mathbf{x}, \tag{2.117}$$

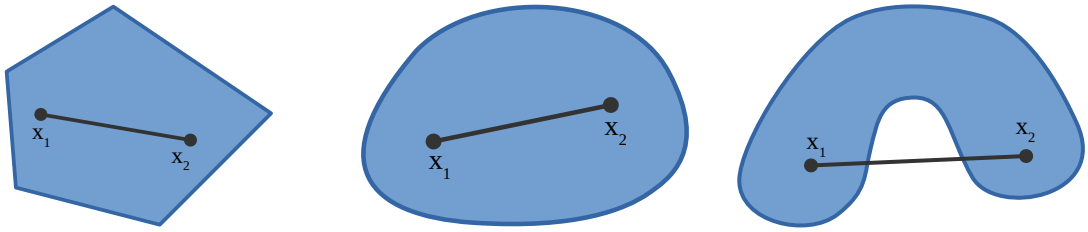
for a function f such that there is no $\boldsymbol{\varphi}'$ satisfying

$$\boldsymbol{\varphi}^\top \mathbf{x}' < \boldsymbol{\varphi}'^\top \mathbf{x}'. \tag{2.118}$$

There are different possibilities to find maximal modular minorants φ. The *DMM* algorithm computes the minorants by distributing the slacks of the chain sub-problems using a hierarchical approach. The advantage of this distribution is that the minorants can be computed very efficiently in parallel on the GPU which yields a fast algorithm in practice. Algorithm 6 shows the complete *DMM* algorithm. The reader is referred to [178] for more details.

2.4.4.4 Summary and Further Reading

We have seen an overview of different inference algorithms for graphical models, especially *CRFs*, in this section. The inference in *CRFs* is an NP-hard problem on grid-graphs and therefore, we need to make approximations to keep these models tractable. In this section we focused on inference algorithms retaining as much information as possible from the original problem. For *CRFs* this means that we tried to keep as many edges from the original grid graph as possible. This yields a family of algorithms based on *DP*, where the sub-problems consist of chains which can be solved efficiently. We did not cover further inference techniques such as the mean-field approximation, graph-cuts, LP-relaxations and sampling methods here. For more information on the mean-field approximation we refer the reader to [202], for earlier applications of mean-field in computer vision see [197, 205], for mean-field approximation combined with Convolutional Neural Networks (CNNs) see *e.g.* [31, 116, 123, 226]. The reader is referred to [19, 21, 75] for more information about graph-cuts and to [20, 186] for applications. For a good overview about graphical models in general we recommend the textbook by Nowozin and Lampert [143] and the surveys [77, 185] including solvers based on the LP-relaxation proposed by Schlesinger [172].



(a) Convex Set (b) Convex Set (c) Non-Convex Set

Figure 2.15: Convex and non-convex sets in $\mathbb{R}^2$.

2.5 Continuous Optimization

In this section, we are interested in problems arising in image processing and computer vision and how we can exploit *continuous optimization* techniques to tackle these problems. Computer vision and image processing problems are usually large-scale problems, because the optimization variable has *e.g.* the size of the image. For an image of size 1000×1000 pixels, *i.e.* "only" a mega-pixel image, we therefore have to solve a one million dimensional optimization problem. This simple example shows that we need to use efficient optimization algorithms in order to be able to actually compute solutions in practice. In this section, we are especially interested in continuous optimization algorithms which can be used in such large-scale practical problems. As we will see in Section 2.5.2, we can exploit gradient information in order to iteratively approach an optimal solution.

2.5.1 Convex Analysis

We will review the most important concepts and tools used in convex optimization in this section. They form the basis for a deep understanding of the problems and algorithms presented in the upcoming sections. The material is based on the excellent textbooks of Beck [5] and Boyd et al. [16].

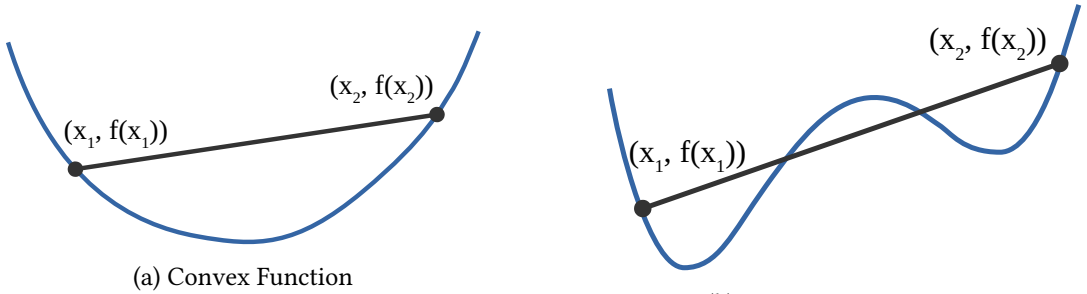
2.5.1.1 Convex Sets

A subset $C \subseteq \mathbb{R}^N$ is a *convex set* if the connecting line between any two points $\mathbf{x}_1, \mathbf{x}_2 \in C$ is also entirely contained in the set C. Formally, this can be defined as

$$\theta \mathbf{x}_1 + (1-\theta)\mathbf{x}_2 \in C \qquad \theta \in [0,1]. \tag{2.119}$$

Figure 2.15 shows two convex sets and one non-convex set. The connecting lines shown in the figure reflect exactly the definition of a convex set in Eq. (2.119). Note that also the empty set and the whole space $\mathbb{R}^N$ are convex sets.

Convexity preserving operations Sometimes it is easier to show the convexity of a set based on convexity preserving operations instead of showing it directly with Eq. (2.119). The following



(a) Convex Function

(b) Non-Convex Function

Figure 2.16: Visualization of an exemplary convex and non-convex function. The black line shows the chord between $\mathbf{x}_1$ and $\mathbf{x}_2$.

list provides some convexity preserving operations:

- The intersection of convex sets $\bigcap_{i=1}^{M} C_i$ is a convex set.
- The Minkowski sum $C_1 + C_2 = \{\mathbf{x}_1 + \mathbf{x}_2 : x_1 \in C_1,\ \mathbf{x}_2 \in C_2\}$ of two convex sets C_1, C_2 is again a convex set.
- A convex set C, which is transformed with an affine function $f = \mathbf{A}\mathbf{x} + \mathbf{b}$, is again a convex set given by $\{f(\mathbf{x}) : \mathbf{x} \in C\}$.

2.5.1.2 Convex Functions

A function $f : C \to \mathbb{R}$ on a convex set $C \subseteq \mathbb{R}^N$ is called a *convex* function if $\forall \mathbf{x}_1, \mathbf{x}_2 \in C$ and $0 \leq \theta \leq 1$ the inequality

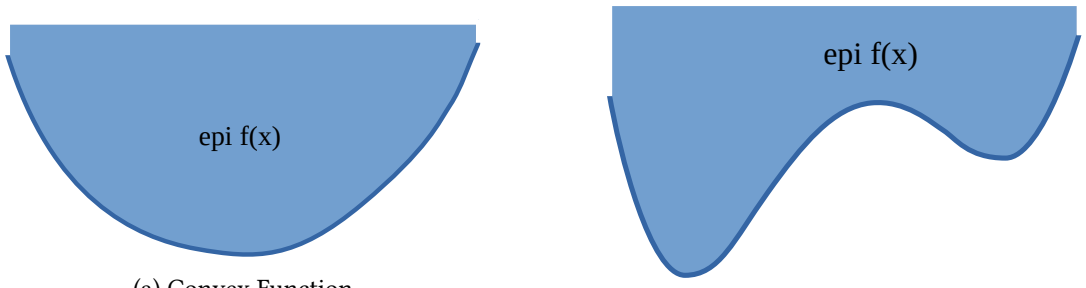
$$f(\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2) \leq \theta f(\mathbf{x}_1) + (1-\theta) f(\mathbf{x}_2) \tag{2.120}$$

holds. Equation (2.120) can be interpreted geometrically as follows. We span a chord (=line) between the two points x_1 and x_2 and if the chord is everywhere above the function, then the function is convex. If inequality (2.120) holds strictly for $0 < \theta < 1$, then the function is called *strictly* convex. You can find a visualization of this geometric interpretation in Fig. 2.16. If a function $f(\mathbf{x})$ is convex, then we call the function $-f(\mathbf{x})$ a *concave* function.

An alternative way of defining a convex function is by using the so-called *epigraph* of the function. The epigraph is the set of all points above the function $f(\mathbf{x})$. This is also reflected in the name, where the Greek word “epi” means “above”. Formally, the epigraph is defined as the set

$$\text{epi}\, f(\mathbf{x}) = \{(\mathbf{x}, t) : \mathbf{x} \in \text{dom}(f),\ t \geq f(\mathbf{x})\} \tag{2.121}$$

If the epigraph (Eq. (2.121)) of a function $f(\mathbf{x})$ is a convex set, then the function $f(\mathbf{x})$ is convex. This can also be easily seen in Fig. 2.17. Note that this figure shows exactly the same functions as used in Fig. 2.16.



(a) Convex Function

(b) Non-Convex Function

Figure 2.17: Visualization of the epigraph of an exemplary convex and non-convex function.

Convexity-preserving operations Similar as for the convex sets, we will also list some convexity-preserving operations on convex functions.

- A non-negative weighting $w_1, \ldots, w_M \geq 0$ of convex functions is again a convex function, thus $f = w_1 f_1 + w_2 f_2 + \cdots + w_N f_N$ is a convex function.
- The composition with an affine mapping is again a convex function. If $f(\mathbf{x})$ is convex, then $g(\mathbf{x}) = f(\mathbf{A}\mathbf{x} + \mathbf{b})$ is also a convex function.
- The point-wise maximum of two functions f_1 and f_2 is a convex function. If f_1 and f_2 are convex functions, then $f(\mathbf{x}) = \max\{f_1(\mathbf{x}), f_2(\mathbf{x})\}$ is convex.

2.5.1.3 Subdifferential

If we deal with non-smooth functions, a gradient can not be computed everywhere. Therefore, we introduce here a generalization of the gradient called subgradient which can then be used to define the subdifferential. A vector $\mathbf{g} \in \mathbb{R}^N$ is called a subgradient at $\mathbf{x} \in \mathbb{R}^N$, if

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \langle \mathbf{g}, \mathbf{y} - \mathbf{x} \rangle \qquad \forall \mathbf{y} \in \mathrm{dom}(f). \tag{2.122}$$

Equation (2.122) states that we are looking for a linear under-estimator of our function. Let us, as an example, take the absolute function visualized in Fig. 2.18. We cannot compute a gradient for this function at the point $\mathbf{x} = 0$ because the absolute function is a non-smooth function. However, we can compute subgradients based on the definition in Eq. (2.122). The gray lines shown in the figure are exemplary subgradients in the point $\mathbf{x} = 0$. The set of all subgradients together is called the *subdifferential*. Formally, it is defined as

$$\partial f(\mathbf{x}) \coloneqq \left\{ \mathbf{g} \in \mathbb{R}^N : f(\mathbf{y}) \geq f(\mathbf{x}) + \langle \mathbf{g}, \mathbf{y} - \mathbf{x} \rangle \ \forall \mathbf{y} \in \mathrm{dom}(f) \right\}. \tag{2.123}$$

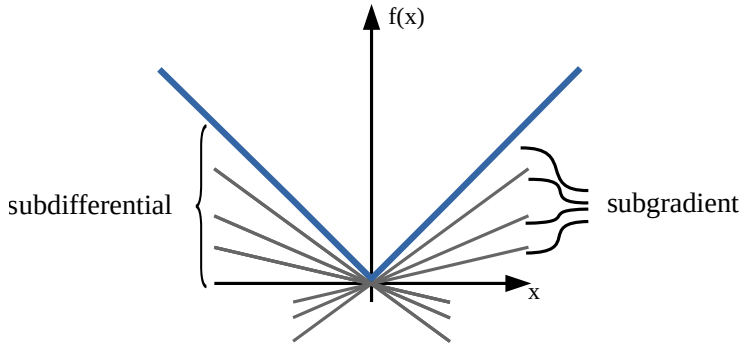


Figure 2.18: Visualization of exemplary sub-gradients (shown in gray) of the absolute function $|\mathbf{x}| = \sum_{i=1}^{N} x_i$. The set of all sub-gradients is called the sub-differential.

Coming back to our example, we can now formally compute the subdifferential with

$$\partial \|\mathbf{x}\|_1 = \begin{cases} -1 & \text{for } x_i < 0, \\ [-1, 1] & \text{for } x_i = 0, \\ 1 & \text{for } x_i > 0, \end{cases} \tag{2.124}$$

where $(x_i)_{i=1}^{N}$ are the components of the vector $\mathbf{x}$.

2.5.1.4 Convex Conjugate

Next, we investigate the dual representation of a convex function which is known as the *convex conjugate* or as the *Legendre–Fenchel transform*. It is defined as the supremum

$$f^*(\mathbf{y}) = \sup_{\mathbf{x} \in \mathbb{R}^N} \{\langle \mathbf{x}, \mathbf{y} \rangle - f(\mathbf{x})\}. \tag{2.125}$$

Geometrically, it can be computed by moving a line $\langle \mathbf{x}, \mathbf{y} \rangle$ upwards until the line touches the graph of the function $f(\mathbf{x})$. Then we need to find the largest slope such that the line is a lower bound of the function $f(\mathbf{x})$. The intersection of this line with the $\mathbf{y}$-axis then corresponds to $-f^*(\mathbf{y})$. Figure 2.19 shows a visualization of a function $f(\mathbf{x})$ with its corresponding dual function $f^*(\mathbf{y})$. The left plot shows the lines lower-bounding the function. This representation also reveals that we actually represent the function by a family of intersections of half-spaces. Note that the dual variable $\mathbf{y}$ represents the slopes of the half-spaces representing our function $f(\mathbf{x})$.

We can also apply the Legendre–Fenchel transform to the conjugate function $f^*(\mathbf{y})$ and get

$$f^{**}(\mathbf{x}) = \sup_{\mathbf{y} \in \mathbb{R}^N} \{\langle \mathbf{x}, \mathbf{y} \rangle - f^*(\mathbf{y})\}, \tag{2.126}$$

which is known as the bi-conjugate function. In general, the bi-conjugate f^{**} is the convex

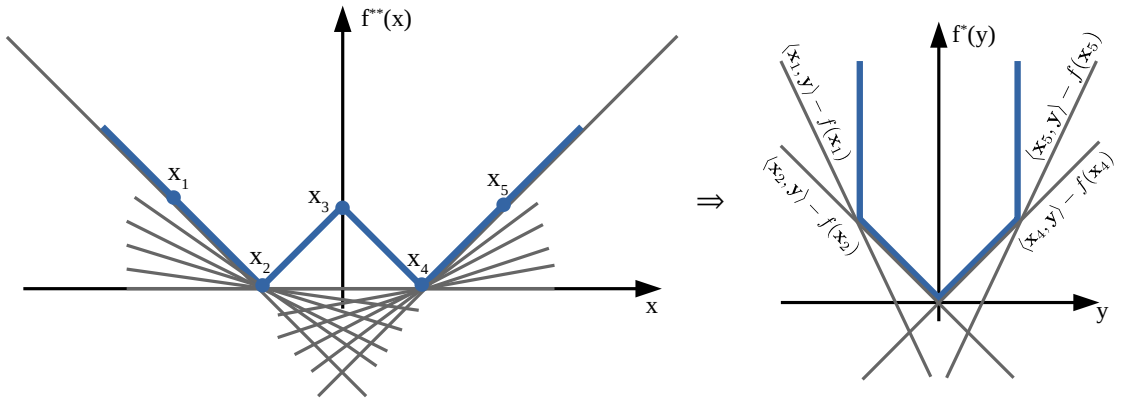


Figure 2.19: The convex conjugate. It can be computed by finding the maximal slope $\mathbf{x}$ such that the line is a lower bound of the function $f(\mathbf{x})$ as visualized in the left plot. Using these lines we can represent the convex conjugate as shown in the right plot.

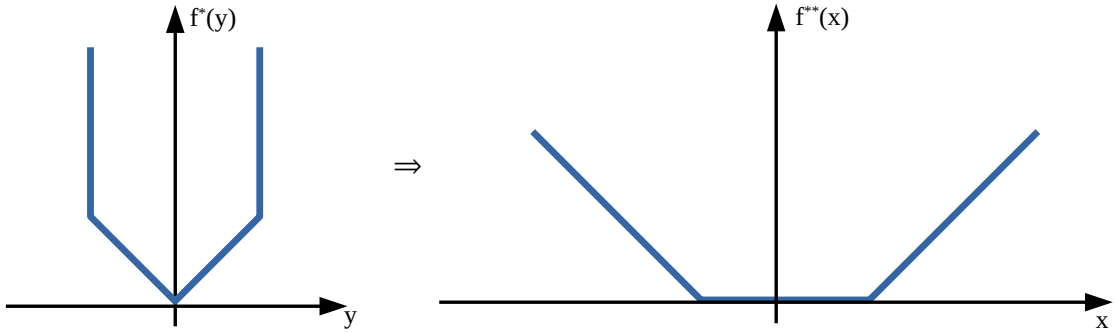


Figure 2.20: Computing the bi-conjugate from the conjugate. See Fig. 2.19 for a visualization of the function $f(x)$.

envelope of f and thus a lower bound of the original function f. Thus, the inequality

$$f^{**}(\mathbf{x}) \leq f(\mathbf{x}) \tag{2.127}$$

holds. Figures 2.19 and 2.20 show the transformation of a function to its conjugate and to its bi-conjugate. Since the original function was non-convex, we have successfully computed the convex envelope of the original function with the bi-conjugate. If the original function f is convex, then

$$f^{**}(\mathbf{x}) = f(\mathbf{x}), \tag{2.128}$$

i.e. the bi-conjugate is the same function as the original function.

Properties of the conjugate function

- The conjugate function f^* is always a convex function, because it is the maximum over a family of linear functions.

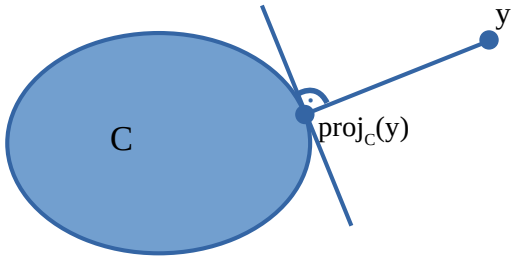


Figure 2.21: Projection of a point y onto a convex set C.

- The bi-conjugate $f^{**}(\mathbf{x})$ is equivalent to the function $f(\mathbf{x})$ if f is convex.
- The bi-conjugate $f^{**}(\mathbf{x})$ is the convex envelope of the function $f(\mathbf{x})$ if f is non-convex.

2.5.1.5 Proximal Operator

The proximal operator, often referred to as "prox", can be derived by generalizing the projection onto a convex set to convex functions. We can define the projection of a given point y onto a closed and non-empty convex set C as the constrained optimization problem

$$\text{proj}_C(\mathbf{y}) = \arg\min_{\mathbf{x}\in C} \|\mathbf{x} - \mathbf{y}\|, \tag{2.129}$$

which is also known as *Euclidean projection* or *orthogonal projection*. Figure 2.21 shows a visualization of this optimization problem. The point minimizing Eq. (2.129) is given by the orthogonal projection of the point y onto the convex set C. We can rewrite the optimization problem in Eq. (2.129) into the unconstrained optimization problem

$$\begin{aligned} \text{proj}_C(\mathbf{y}) &= \arg\min_{\mathbf{x}\in C} \|\mathbf{x} - \mathbf{y}\| && (2.130)\\ &= \arg\min_{\mathbf{x}\in C} \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2 && (2.131)\\ &= \arg\min_{\mathbf{x}\in \mathbb{R}^N} \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2 + \delta_C(\mathbf{x}), && (2.132) \end{aligned}$$

where δ_C is the indicator function of the set C defined as

$$\delta_C(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \in C, \\ +\infty & \text{else.} \end{cases} \tag{2.133}$$

Next, we generalize Eq. (2.132) to get the definition of the proximal operator. To this end, we replace the indicator function in Eq. (2.132) by a convex function f and get

$$\text{prox}_{\lambda f}(\mathbf{y}) = \arg\min_{\mathbf{x}\in \mathbb{R}^N} f(\mathbf{x}) + \frac{1}{2\lambda}\|\mathbf{x} - \mathbf{y}\|^2, \tag{2.134}$$

where $\lambda \in \mathbb{R}$. Equation (2.134) is known as the *proximal operator* with respect to a function f.

Next, we derive an alternative definition based on the optimality condition of the proximal operator which is

$$\lambda \partial f(\mathbf{x}) + \mathbf{x} - \mathbf{y} \ni \mathbf{0} \tag{2.135}$$
$$\Leftrightarrow \lambda \partial f(\mathbf{x}) + \mathbf{x} \ni \mathbf{y} \tag{2.136}$$
$$\Leftrightarrow (\mathbf{I} + \lambda \partial f)(\mathbf{x}) \ni \mathbf{y} \tag{2.137}$$
$$\Leftrightarrow \mathbf{x} = (\mathbf{I} + \lambda \partial f)^{-1}(\mathbf{y}) \tag{2.138}$$
$$\Leftrightarrow \mathbf{x} = \operatorname{prox}_{\lambda f}(\mathbf{y}), \tag{2.139}$$

where $\partial f(\mathbf{x})$ is the sub-differential of the function f. We note that the proximal operator can also be used with non-smooth convex functions.

An important result, known as *Moreau's Identity* relates the proximal operators of f and f^* and is given by

$$\mathbf{x} = \operatorname{prox}_{\lambda f}(\mathbf{x}) + \lambda \operatorname{prox}_{\lambda^{-1} f^*}\left(\frac{\mathbf{x}}{\lambda}\right). \tag{2.140}$$

The proximal operator is *e.g.* used in Proximal Methods and in the Primal-Dual Method. We will see later in Chapter 5 how we can also use the proximal operator to *e.g.* project to the positive half-space.

2.5.2 Convex Optimization

Convex optimization is an important sub-discipline in optimization. This field studies optimization problems of the form

$$\min_{x \in \mathbb{R}^N} f(\mathbf{x}) \tag{2.141}$$
$$\text{s.t. } f_i(\mathbf{x}) \leq 0, \ i = 1, \ldots, M, \tag{2.142}$$

where $f \colon \mathbb{R}^N \to \mathbb{R}$ is a convex, but possibly non-smooth function and the functions $f_i \colon \mathbb{R}^N \to \mathbb{R}$ are convex constraint functions. Since Eq. (2.142) is a minimization problem, we want to find an $\mathbf{x}^*$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x}. \tag{2.143}$$

The argument $\mathbf{x}^*$ is referred to as the minimizer of f. The convexity of the domain and the function yields the beneficial property that any local minimum is also a global minimum. Furthermore, if a minimum exists and the function is strictly convex, then the global optimum is also unique. These properties allow to analyse convex optimization problems and we can come up with algorithms that are guaranteed to converge towards the optimal solution.

Here we are mainly interested in the algorithms which we can use to optimize large scale optimization problems with possibly millions of dimensions. The following optimization algorithms exploit gradient information of the function which should be optimized. While the algorithms are

Algorithm 7: Gradient Method

Input: Choose $\mathbf{x}^0 \in \mathbb{R}^N$, step-size $\alpha^k > 0$
Result: Optimal $\mathbf{x}$
for $k \geq 0$ **do**
  $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha^k \nabla f(\mathbf{x}^k)$
end for

Algorithm 8: Sub-Gradient Method

Input: Choose $\mathbf{x}^0 \in \mathbb{R}^N$, step-size $\alpha^k > 0$
Result: Optimal $\mathbf{x}$
for $k \geq 0$ **do**
  Compute $d(\mathbf{x}^k) \in \partial f(\mathbf{x}^k)$
  $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha^k d(\mathbf{x}^k)$
end for

guaranteed to converge to the global optimum if the function is convex, they converge to a local optimum or a saddle point in the non-convex case. We start with the most basic gradient descent algorithms and move on to proximal algorithms until we arrive at the powerful primal-dual algorithm.

2.5.2.1 Gradient Methods

We consider here the generic optimization defined in Eq. (2.142) and assume the function to be convex and smooth functions. Some of the algorithms can also be applied to non-smooth functions. We will note this in the respective paragraphs.

(Sub-) Gradient Descent The simplest method which we can use for optimization is referred to as *gradient method*, *steepest descent method* or *gradient descent*. All three names refer to the same optimization algorithm. As the name already indicates the key idea in the gradient method is to iteratively perform steps in the direction of steepest descent. The steepest descent direction for a continuously differentiable function is given as the negative gradient of the function

$$d(\mathbf{x}^k) = -\nabla f(\mathbf{x}^k), \tag{2.144}$$

where we call d a *descent direction*. Algorithm 7 shows a listing of the complete algorithm. If the function f has a Lipschitz-continuous gradient, *i.e.*

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\| \tag{2.145}$$

for a finite constant L, then it can be shown that the gradient method converges with a constant step-size $\alpha^k \in (0, 2/L)$ with rate $O(1/k)$.

Algorithm 9: Heavy Ball Method [155]

Input: Choose $\mathbf{x}^0 \in \mathbb{R}^N$, set $\mathbf{x}^{-1} = \mathbf{x}^0$, $\alpha^k, \beta^k > 0$
Result: Optimal $\mathbf{x}$
for $k \geq 0$ **do**
  $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha^k \nabla f(\mathbf{x}^k) + \beta^k (\mathbf{x}^k - \mathbf{x}^{k-1})$
end for

Algorithm 10: Nesterov Accelerated Gradient Method

Input: Choose $\mathbf{x}^0 \in \mathbb{R}^N$, set $\mathbf{x}^{-1} = \mathbf{x}^0$, $\alpha^k, \beta^k > 0$
Result: Optimal $\mathbf{x}$
for $k \geq 0$ **do**
  $\mathbf{y}^k = \mathbf{x}^k + \beta^k (\mathbf{x}^k - \mathbf{x}^{k-1})$
  $\mathbf{x}^{k+1} = \mathbf{y}^k - \alpha^k \nabla f(\mathbf{y}^k)$
end for

In case the function f is non-smooth, the descent direction is given by a sub-gradient (*c.f.* Section 2.5.1.3)

$$d(\mathbf{x}^k) \in \partial f(\mathbf{x}^k) \tag{2.146}$$

which is used in the sub-gradient method shown in algorithm 8. Since non-smooth functions are more difficult to optimize, the convergence rate of the sub-gradient method is only $O(1/\sqrt{k})$. We note, that the sub-gradient method is often used for *learning* optimal parameters in deep *CNNs* because many activation functions are not differentiable in all points, *e.g.* ReLU is only sub-differentiable in the point 0.

Heavy Ball Method Polyak [155] proposed the *Heavy Ball Method*, also known as *gradient descent with momentum*. This algorithm achieves a better convergence rate of $O\left(1/k^2\right)$. It is motivated from a physical point of view, where the model adds friction to a body moving in a potential field. It can be derived from a second-order Ordinary Differential Equation (ODE) representing the motion of this body in the potential field. The acceleration term in the *ODE* yields the additional term $\mathbf{x}^k - \mathbf{x}^{k-1}$ which is referred to as *momentum*. Intuitively, if the ball rolls down a steep curve and approaches a flat region, then the acceleration (=momentum) in the ball is used to make larger steps towards the optimum also in flat regions. This is the reason for the name "Heavy Ball". Algorithm 9 shows a listing of the method.

Nesterov Accelerated Gradient Method Next, we present the optimal first-order method of Nesterov [138] which achieves, similar as the Heavy Ball Method, a convergence rate of $O\left(1/k^2\right)$. This algorithm first performs an extrapolation step and then computes the new descent direction using the gradient at the extrapolated position. Using the Lipschitz constant L, we can compute the step-size as $\alpha^k = 1/L$. The proof in Nesterov's paper [138] shows that the optimal value for β^k

is given by

$$\beta^k = \frac{t_k - 1}{t_{k+1}}, \tag{2.147}$$

where one needs to be careful with the used indices, since we subtract 1 from t_k in the enumerator and the denominator uses the next iterate t_{k+1} in Eq. (2.147). The variables t_k can be computed by

$$t^{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2}. \tag{2.148}$$

A simpler yet good approximation to Eq. (2.147) is given by

$$\beta^k = \frac{k - 1}{k + 2}, \tag{2.149}$$

as also shown in [138]. We provide a complete listing of the method in algorithm 10.

Nesterov showed in [140] that the convergence rate of $O\left(1/k^2\right)$ is the lower bound for any first-order method on smooth, convex functions. Therefore, both the heavy ball method and the accelerated gradient method are *optimal methods*.

Machine Learning The algorithms presented above come with strong convergence guarantees if applied to convex functions. However, these algorithms are not only applicable to convex problems. Actually, the presented algorithms are also implemented in current deep learning frameworks such as PyTorch, Tensorflow, *etc.* and are often used to find the optimal parameters for Machine Learning (ML) models. They are heavily used in training non-linear and non-convex *CNNs* and also work very well in this setting. While the convergence guarantees do not hold anymore when applied to non-convex functions, the trend is similar in practice, *i.e.* the heavy ball method and Nesterov's accelerated gradient method usually converge faster than gradient descent.

2.5.2.2 Proximal Methods

Proximal methods are a family of algorithms which make use of the Proximal Operator in the algorithm. The advantage of proximal methods is that they can also handle non-smooth objective functions via the proximal map. In this section, we assume that the optimization problem at hand is a composite function

$$\min_{\mathbf{x} \in \mathbb{R}^N} f(\mathbf{x}) := g(\mathbf{x}) + h(\mathbf{x}), \tag{2.150}$$

where the function g is continuously differentiable and convex and the function h is convex, but possibly non-smooth.

Let us first start with the proximal gradient method. This method can be seen as a generalization of the gradient method presented in Section 2.5.2.1. The proximal gradient method performs an explicit gradient step on the differentiable function g and a proximal step (=implicit gradient step) on the non-differentiable function h. Although the proximal gradient method can handle

Algorithm 11: Proximal Gradient Method

Input: Choose $\mathbf{x}^0 \in \mathrm{dom}(f)$, step-size $\alpha^k > 0$
Result: Optimal $\mathbf{x}$
for $k \geq 0$ **do**
  $\mathbf{x}^{k+1} = \mathrm{prox}_{\alpha^k h}\left(\mathbf{x}^k - \alpha^k \nabla g(\mathbf{x}^k)\right)$
end for

Algorithm 12: Accelerated Proximal Gradient Method

Input: Choose $\mathbf{x}^0 \in \mathrm{dom}(f)$, set $\mathbf{x}^{-1} = \mathbf{x}^0$, $\alpha^k, \beta^k > 0$
Result: Optimal $\mathbf{x}$
for $k \geq 0$ **do**
  $\mathbf{y}^k = \mathbf{x}^k + \beta^k(\mathbf{x}^k - \mathbf{x}^{k-1})$
  $\mathbf{x}^{k+1} = \mathrm{prox}_{\alpha^k h}\left(\mathbf{y}^k - \alpha^k \nabla g(\mathbf{y}^k)\right)$
end for

non-differentiable functions, it can be proven that the convergence rate of the algorithm is $O(1/k)$. This is notable because this is the same rate of convergence as for the gradient method which can only handle smooth functions. For comparison, a sub-gradient method which can handle non-differentiable functions only has a convergence rate of $O(1/\sqrt{k})$. Thus, the proximal operator significantly improves the rate of convergence for non-smooth functions. Again, if we know the Lipschitz constant L of the gradient (*c.f.* Eq. (2.145)) we can set the constant step-size to $\alpha = \alpha^k \in (0, 2/L)$. The complete algorithm is shown in algorithm 11.

Now, the natural question arises, whether we can improve this convergence rate by adding momentum, similarly as we did in Section 2.5.2.1 for smooth functions. It actually turns out that this is possible. The resulting method is then called *accelerated proximal gradient method* and shown in algorithm 12. The only change we need to make is to additionally conduct the extrapolation step before performing the explicit gradient step on the continuously differentiable function g and the proximal step on the non-differentiable function h. It can be shown that this algorithm converges with rate $O(1/k^2)$ which is the best convergence rate we can achieve.

We also want to note that the proximal methods described above are special instances of the generic Proximal Point Algorithm (PPA)

$$\mathbf{x}^{k+1} = \mathrm{prox}_{\alpha^k f}(\mathbf{x}^k). \tag{2.151}$$

Variants The proximal methods presented in this section perform an explicit gradient step on the differentiable function g and an implicit (= proximal) step on the non-differentiable function h. Note, however, that it is sometimes beneficial to perform the gradient step on the non-differentiable function and the proximal step on the differentiable function. The method is then called *proximal sub-gradient method*, because we can only compute sub-gradients on the non-differentiable function.

Algorithm 13: Primal-Dual Algorithm

Input: Choose $(\mathbf{x}^0, \mathbf{y}^0) \in \mathbb{R}^N \times \mathbb{R}^M$, set $\bar{\mathbf{x}}^0 = \mathbf{x}^0$, $\tau, \sigma > 0$, $\theta \in [0, 1]$
Result: Optimal primal point $\mathbf{x}$ and dual point $\mathbf{y}$

for $k \geq 0$ **do**
  $\mathbf{y}^{k+1} = \text{prox}_{\sigma f^*}\left(\mathbf{y}^k + \sigma \mathbf{K}\bar{\mathbf{x}}^k\right)$ `// Gradient ascent on the dual`
  $\mathbf{x}^{k+1} = \text{prox}_{\tau g}\left(\mathbf{x}^k - \tau \mathbf{K}^* \mathbf{y}^{k+1}\right)$ `// Gradient descent on the primal`
  $\bar{\mathbf{x}}^{k+1} = \mathbf{x}^{k+1} + \theta(\mathbf{x}^{k+1} - \mathbf{x}^k)$ `// Over-relaxation`
end for

Similar as in the non-proximal algorithms in Section 2.5.2.1, the proximal methods here are written in their most generic form. It turns out that the optimal value for β^k can be computed with Eqs. (2.147) and (2.148) which we already used in Section 2.5.2.1. If we use these rules to compute β^k, we have actually instantiated the famous Fast Iterative Shrinkage and Thresholding Algorithm (FISTA) [6].

2.5.2.3 Primal-Dual Method

In this section we introduce a class of problems which can be solved efficiently with the *primal-dual algorithm* proposed by Chambolle and Pock [27]. The general optimization problem which we consider here is of the form

$$\min_{\mathbf{x} \in \mathbb{R}^N} f(\mathbf{x}) \coloneqq g(\mathbf{x}) + h(\mathbf{K}\mathbf{x}), \tag{2.152}$$

where $f\colon \mathbb{R}^N \to \mathbb{R} \cup \{\infty\}$ is a composite function consisting of a convex and continuously differentiable function $g\colon \mathbb{R}^N \to \mathbb{R} \cup \{\infty\}$ and a convex, but possibly non-smooth, function $h\colon \mathbb{R}^M \to \mathbb{R} \cup \{\infty\}$ and $\mathbf{K}$ is a linear operator. The optimization problem in Eq. (2.152) can be transferred to the primal-dual problem

$$\min_{\mathbf{x} \in \mathbb{R}^N} \max_{\mathbf{y} \in \mathbb{R}^M} \langle \mathbf{K}\mathbf{x}, \mathbf{y} \rangle + g(\mathbf{x}) - h^*(\mathbf{y}), \tag{2.153}$$

where $\mathbf{y} \in \mathbb{R}^M$ is the dual variable and $h^*\colon \mathbb{R}^M \to \mathbb{R} \cup \{\infty\}$ is the convex conjugate (*c.f.* Section 2.5.1.4) of the function h. If we solve the minimization problem, we get the dual formulation of the problem yielding

$$\max_{\mathbf{y} \in \mathbb{R}^M} - \left(g^*(-\mathbf{K}^*\mathbf{y}) + h^*(\mathbf{y})\right), \tag{2.154}$$

where the function $g^*\colon \mathbb{R}^N \to \mathbb{R} \cup \{\infty\}$ is the convex conjugate of the function g and $\mathbf{K}^*$ is the adjoint operator as defined in Section 2.2.4.

The primal-dual algorithm iterates the steps i) proximal gradient ascent on the dual problem, ii) proximal gradient descent on the primal problem and iii) over-relaxation. The respective step-sizes

for the primal and dual steps need to be larger than zero, *i.e.* $\tau > 0$ and $\sigma > 0$, and satisfy

$$\tau\sigma L^2 \leq 1, \tag{2.155}$$

where L is the Lipschitz constant of the gradient of h, which can be computed with

$$L = \|\mathbf{K}\|, \tag{2.156}$$

where $\|\mathbf{K}\|$ is the operator norm which corresponds to the largest singular value of $\mathbf{K}$ in this case. A complete listing is shown in algorithm 13. By investigating the algorithm in detail, one can observe that it is necessary to be able to compute the norm of the operator $\mathbf{K}$ in a reasonable time and that the proximal operator needs to be computable easily enough. This means, either we can come up with a closed form solution for the proximal operator, or we can find the solution within a few iterations of an auxiliary optimization problem. Chambolle and Pock [27] applied this algorithm to a variety of computer vision problems, including denoising, super-resolution, deconvolution and motion estimation. Furthermore, the authors give formal proofs about the convergence rate which is $O(1/k)$ for convex problems and $O(1/k^2)$ for strongly convex problems. An extended primal-dual algorithm with preconditioning is shown in [153]. Preconditioning speeds up convergence of the algorithm if the matrix $\mathbf{K}$ is *e.g.* inappropriately scaled.

2.5.3 Variational Methods for Stereo

We are interested in correspondence problems such as stereo and optical flow as described in Section 1.2. The task is to find a 1D in stereo or a 2D displacement in optical flow, respectively. More precisely, given two images $f_1, f_2 \colon \Omega \to \mathbb{R}^C$ with C channels, we are seeking a displacement field $d \colon \Omega \to \mathbb{R}^2$, such that $f_0(\mathbf{x} + d(\mathbf{x})) = f_1(\mathbf{x})$ for every pixel position $\mathbf{x} = (x_1, x_2)$. We refer to image f_0 as the *reference image* and to f_1 as the *second image*. For stereo we have $d(\mathbf{x}) = (d_1(\mathbf{x}), 0)^\top$ to model the 1D displacement and for optical flow we allow a 2D displacement by setting $d(\mathbf{x}) = (d_1(\mathbf{x}), d_2(\mathbf{x}))^\top$.

We start here the formal description of the correspondence problems with the continuous problem formulation and show later how this formulation can be discretized to be applicable to the discrete pixels of real-world images. We can define the optimization problem for the correspondence problem as

$$\min_d \int_\Omega |\nabla d| \, d\mathbf{x} + \int_\Omega |f_0(\mathbf{x} - d(\mathbf{x})) - f_1(\mathbf{x})| \, d\mathbf{x}, \tag{2.157}$$

where the first term is the *total variation* of the function $d \in W^{1,1}(\Omega)$ modeling the displacement and the second term, *i.e.* the data-term, models the Brightness Constancy Assumption (BCA) [72, 118] between f_0 and f_1. The *BCA* states that a pixel and its corresponding displaced pixel have the same gray-scale value or intensity, respectively. If we can find the correct displacement d for every pixel, then the data-term vanishes. However, the *BCA* cannot be used as shown in Eq. (2.157), because it is non-convex and non-linear in the unknown d. We can compute a linear approximation

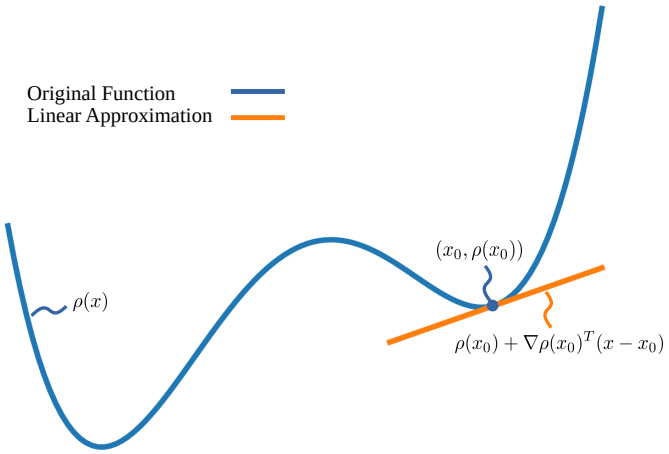


Figure 2.22: first-order Taylor approximation (=linear approximation) of a function $\rho(x)$, shown in blue, in the point x_0. The linear approximation is shown in orange.

to the *BCA* by using a first-order Taylor series expansion. Using the shortcut

$$\rho(d) = f_0(\mathbf{x} - d(\mathbf{x})) - f_1(\mathbf{x}), \tag{2.158}$$

we can compute the linearized *BCA* with

$$\rho(d) \approx \hat{\rho}(d) = \rho(d_0) + \nabla\rho(d_0)^\top (d - d_0), \tag{2.159}$$

where

$$\nabla\rho(d_0) = \begin{pmatrix} \frac{\partial\rho(d_0)}{\partial x_1} \\ \frac{\partial\rho(d_0)}{\partial x_2} \end{pmatrix}, \tag{2.160}$$

i.e. the vector of partial derivatives. The linearized *BCA* shown in Eq. (2.159) yields the famous Optical Flow Constraint (OFC) [72, 118]. Figure 2.22 shows a graphical interpretation of the original function (2.158) and its linear approximation (2.159) in 1D.

We can now plug the linearization (2.159) into Eq. (2.157) and get

$$\min_d \int_\Omega |\nabla d| \, d\mathbf{x} + \int_\Omega |\hat{\rho}(d)| \, d\mathbf{x}. \tag{2.161}$$

Equation (2.161) is now convex and can thus be optimized efficiently using the Primal-Dual Method [27] shown in Section 2.5.2.3.

Note that the optimization problem in Eq. (2.161) is entirely defined on the continuous domain Ω. In order to apply Eq. (2.161) to real-world images, we have to discretize it. A natural discretization is for example given by the discrete pixel-grid itself. We thus discretize the continuous domain Ω to a discrete domain of size $M \times N$ which we denote by Ω_{MN}.

In the discrete domain the reference and second image are defined as functions $f_0, f_1 : \Omega_{MN} \rightarrow$

$\mathbb{R}^C$, where C is again the number of channels in the image. We can define the discrete gradient operator ∇d with standard finite differences with Neumann boundary conditions as

$$(\nabla d)_{i,j,1} = \begin{cases} d_{i,j+1} - d_{i,j} & \text{if } j < N \\ 0 & \text{else,} \end{cases} \qquad (\nabla d)_{i,j,2} = \begin{cases} d_{i+1,j} - d_{i,j} & \text{if } i < M \\ 0 & \text{else,} \end{cases} \tag{2.162}$$

where $(\nabla d)_{\cdot,\cdot,1}$ is the derivative in x_1-direction and $(\nabla d)_{\cdot,\cdot,2}$ is the derivative in x_2-direction, respectively. The discrete gradient (2.162) is therefore a mapping $\nabla : \mathbb{R}^{M\times N\times P} \to \mathbb{R}^{M\times N\times 2P}$. We can now state the discrete version of Eq. (2.161) which is given by

$$\min_{d\in\mathbb{R}^{M\times N}} \|\nabla d\|_{2,1} + \lambda \|\hat{\rho}(d)\|_1, \tag{2.163}$$

where

$$\|\nabla d\|_{2,1} = \sum_{i,j} \sqrt{\sum_k |(\nabla d)_{i,j,k}|^2}, \tag{2.164}$$

is the discrete total variation and

$$\|\hat{\rho}(d)\|_1 = \sum_{i,j} |(\hat{\rho}(d))_{i,j}| \tag{2.165}$$

is the ℓ_1-norm of the vector $\hat{\rho}(d)$. Note that the ℓ_1-norm ensures robustness against strong outliers. However, as shown in Fig. 2.22, the linearization in Eq. (2.159) is only locally a good approximation to the true function in Eq. (2.158). Therefore, the optimization Eq. (2.163) is usually embedded into a coarse-to-fine warping scheme as shown in [24]. The warping ensures that the initial point d_0 is always close to the current point d in Eq. (2.159).

2.6 Machine Learning

Machine Learning (ML) is together with optimization (*c.f.* Sections 2.4 and 2.5) the main tool we use throughout this thesis. The material of this section is based on the textbooks by Murphy [137], Russell and Norvig [167], Shalev-Shwartz and Ben-David [176].

We start in the upcoming section with Artificial Intelligence (AI) and move then on towards the data driven specializations *ML* and Deep Learning (DL). Figure 2.23 shows a high-level overview of these three terms and sets them into relation.

2.6.1 Artificial Intelligence

The term Artificial Intelligence (AI) was first used by the computer scientists John McCarthy et al. [127] in 1955. Back then, they wrote a proposal to organize a summer research project on *Artificial Intelligence* in the year 1956 in Dartmouth. Together with selected scientists, the goal was to investigate how machines can be programmed such that they can solve problems, which was thought to be reserved for humans only. Hence, the term *AI* only judges the behavior of

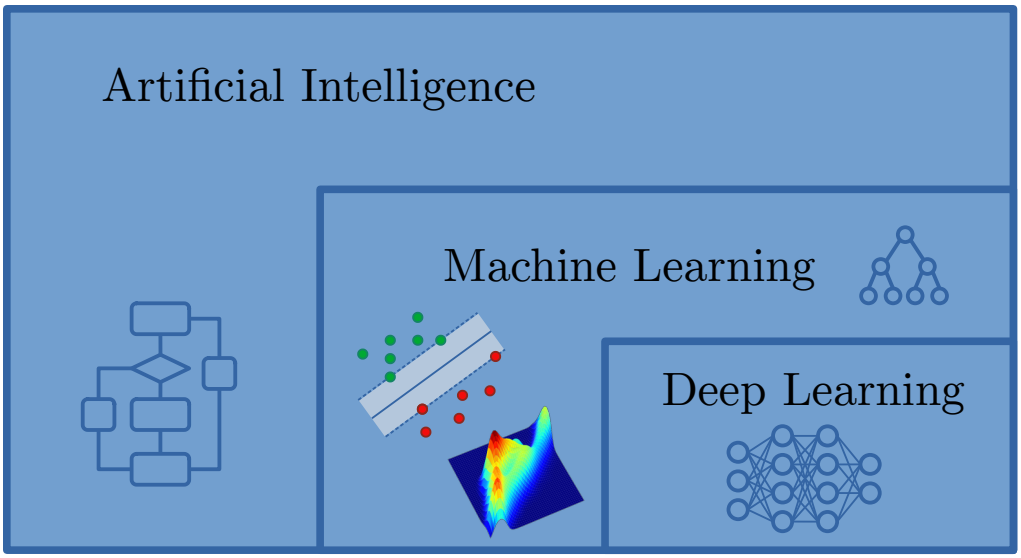


Figure 2.23: *AI*, *ML* and *DL* in relation.

a machine, but it is indifferent as to how the machine acquired its behavior. For example, an industrial robot which is programmed to automatically sort items on a conveyor belt can be considered as a machine with *AI* because it mimics human behavior. The fact that the robot is just programmed and not intelligent in terms of human intelligence is not important here.

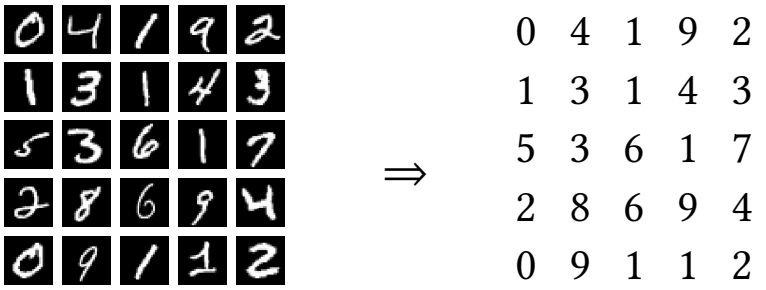
Although, in 1956 the technical capabilities have been very limited compared to today's high-end computing infrastructure, the proposal contained many terms which are common building blocks in current state-of-the-art models. Examples are the terms *Neural Nets*, *Self-Improvement* and *Randomness*. As we will describe in more detail later, Artificial Neural Network are nowadays one of the main driving forces in *AI*. Self-Improvement became an independent research area, where models try to automatically learn from their own failures and randomness is a core part during training of state-of-the-art models.

2.6.2 Machine Learning

In general, Machine Learning (ML) defines data-driven methods to automatically detect patterns in data. This usage of *data* is the main difference compared to general *AI*. We can formally define a machine learning model as

$$f : \mathcal{X} \rightarrow \mathcal{Y}, \quad \mathbf{x} \mapsto \mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta}), \tag{2.166}$$

where $\mathcal{X} \subset \mathbb{R}^N$ is the input domain, $\mathcal{Y} \subset \mathbb{R}^M$ is the output domain, $\mathbf{x} \in \mathcal{X}$ is the input, $\mathbf{y} \in \mathcal{Y}$ is the output and $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ are the learnable parameters with $\boldsymbol{\Theta}$ being the set of all learnable parameters. In the simplest case, the input and output are vectors. Fig. 2.24 shows such an example, where the input vector $\mathbf{x}$ is an image of a handwritten digit and the output $\mathbf{y}$ is the digit interpreted as a number. For this specific example the input domain $\mathcal{X} = \{\mathbf{x}^{(i)} \in \Omega\}_{i=1}^{N}$ is the set of all N images



(a) Input images $\mathbf{x} \in \mathcal{X}$ (b) Output digits $\mathbf{y} \in \mathcal{Y}$

Figure 2.24: An input image $\mathbf{x}$ is processed by a machine learning method to compute the digit $\mathbf{y}$ visible on the image. Sample images taken from the MNIST handwritten digit database [106].

and the output domain $\mathcal{Y} = \{0, 1, \ldots, 9\}$ defines the set of possible outcomes representing the digits from 0 to 9.

Regardless of the actual model f, the question remains how *well* parameters $\boldsymbol{\theta}$ should be calculated for the model. Depending on this, we can categorize machine learning approaches into *supervised learning* (*c.f.* Section 2.6.3), *unsupervised learning* and *reinforcement learning*. These approaches are distinguished by the used data during training. If *ground-truth* data is available, then we are in the supervised setting (*c.f.* Fig. 2.25b). In difference, the term unsupervised learning indicates, that we *do not* have labels for our data. Reinforcement learning is mainly used in agent systems, where the agent tries to learn a strategy in order to maximize some reward. Besides these main categories, there are also variants such as *semi-supervised learning*, *weakly supervised learning* or *self-supervised learning* (*c.f.* Section 2.6.4). The names suggest that these variants are neither fully supervised nor unsupervised. Figure 2.25a shows an example with different ways of supervision. Figure 2.25b shows the pixel-wise annotation used in (full) supervised learning and Figs. 2.25c and 2.25d show two examples of weak supervision; bounding boxes and scribbles.

2.6.3 Supervised Learning

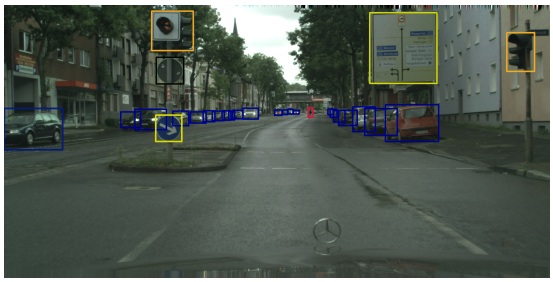
Due to its superior performance, supervised learning is the most common type of *ML*. In order to perform supervised learning we need a labeled dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$, where $\mathbf{x}^{(i)} \in \mathcal{X}$ is the input and $\mathbf{y}^{(i)} \in \mathcal{Y}$ is the desired output. Thus, the dataset contains corresponding input-output pairs, which we can use to train our models. Based on the output domain $\mathcal{Y}$, we distinguish between *classification* and *regression*.

2.6.3.1 Classification

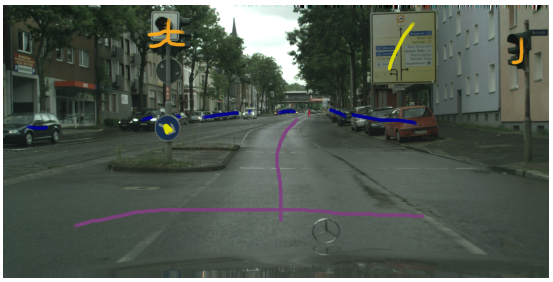
We call a machine learning problem a *classification* problem, if the output domain $\mathcal{Y}$ is defined by a discrete set. The classification problem is referred to as a *binary* classification problem if $|\mathcal{Y}| = 2$ and as a *multi-class* classification problem if $|\mathcal{Y}| > 2$. The following list shows some exemplary classification problems together with their input and output domains.

(a) Input Image

(b) Full Annotation



(c) Bounding Boxes



(d) Scribbles

Figure 2.25: Full and weak supervision. (a) shows the input image and (b) the corresponding full pixel-wise annotation. (c) and (d) show weak annotations, *i.e.* instead of the pixel-wise annotation only bounding-boxes or scribbles, respectively. Images taken from the Cityscapes dataset [39]

- **Handwritten digit recognition** Given an image, the task is to identify the digit shown on this image. Thus, the input domain $\mathcal{X} = \{\mathbf{x}^{(i)} \in \Omega\}_{i=1}^{N}$ corresponds to the set of training images and the output domain $\mathcal{Y} = \{0, \dots, 9\}$ corresponds to the possible digits. Figure 2.24 shows the exemplary input images together with their ground-truth output labels.

- **Semantic Image Segmentation** Given an input image, we are interested in a pixel-wise segmentation into the categories visible in the image pixels. Thus, the input domain $\mathcal{X} = \{\mathbf{x}^{(i)} \in \Omega\}_{i=1}^{N}$ corresponds to the training images and the output domain is for every pixel *e.g.* $\mathcal{Y} = \{$*car, street, traffic sign, vegetation, sky,* $\dots\}$. Figure 2.25b shows a semantic image segmentation, where different colors encode different semantic labels.

- **Stereo** Given two rectified images, the task is to compute for every pixel in the first image the corresponding pixel in the second image. Thus, we want to compute a 1-dimensional displacement for every pixel. Therefore, the input domain is given by $\mathcal{X} = \{(\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}) : \mathbf{x}_{\{1,2\}}^{(i)} \in \Omega\}_{i=1}^{N}$ and the output domain is for every pixel the set of all possible displacements $\mathcal{Y} = \{0, 1, \dots, D\}$, where D is the maximal displacement. Figure 2.26 shows the input images with the corresponding disparity map of the "Adirondack" from the Middlebury dataset [169].

(a) I_0 (b) I_1 (c) Disparity Map

Figure 2.26: The stereo problem. Given two images from a calibrated camera the task is to compute a dense disparity map capturing the displacement between I_0 and I_1 for every pixel. The disparity in (c) is color-coded ranging from far away (cold) to near (warm). Images from the Middlebury 2014 Stereo Benchmark [169].

We can instantiate Eq. (2.166) for a classification problem using a *discriminative* approach with

$$\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}), \tag{2.167}$$

where $\boldsymbol{\theta} \in \Theta$ are the parameters of our model, $\mathbf{x} \in \mathcal{X}$ is the input image and $\mathbf{y} \in \mathcal{Y}$ is the finite and discrete output. In difference to the discriminative approach, where we directly model the posterior distribution, we model the joint distribution $p(\mathbf{X}, \mathbf{Y}) = p(\mathbf{X}|\mathbf{Y})p(\mathbf{Y})$ in a *generative* approach.

2.6.3.2 Regression

Regression is the second category in machine learning. The difference compared to classification is that the output domain is a continuous domain such as $\mathcal{Y} = \mathbb{R}^N$. For example, in geometric problems such as stereo and optical flow, the final goal is to compute a continuous disparity map or flow field, respectively. Fig. 2.27 shows a visual comparison for the stereo task. The CNN-CRF method [85] predicts a discrete result, while the VN [84] computes a continuous and thus sub-pixel accurate solution.

2.6.4 Self-Supervised Learning

A machine learning approach is referred to as self-supervised learning, if the training data is labeled automatically. The training itself is then conducted using supervised learning as described in the previous Section 2.6.3.

One common principle for automatic training data generation is *reconstruction*. Thereby, some known parts of an image are hidden and the learning task is formulated to reconstruct the missing part. Recent works following this pattern are *e.g.* image inpainting [150], where the model needs to fill in the missing part of an image. In cross-channel prediction the task is to predict color from luminance and vice versa as done by [224]. Another example is colorization [223], where the task is to predict the color image given a grayscale image.

Figure 2.27: Comparison between discrete and continuous stereo. The disparity maps are shown using a high-frequency visualization to highlight the disparity changes. Images from [85] and [84].

The second principle we want to briefly describe here is label generation through commonsense reasoning. Commonsense thereby refers to the ability to set everyday things into relation. This could be *e.g.* jigsaw puzzles where the task is to align parts of an image [141], rotating images to some canonical representation [61] or context prediction [43]. Note that the task in the examples above is actually often only a proxy task for which training data can be automatically generated using the image data itself. For example, in the jigsaw puzzle, the model needs to learn the relation and the context of objects (proxy task) and thus our model needs to implicitly learn *e.g.* semantic information to distinguish between a foreground object and the background. Therefore, self-supervised learning approaches can also be used to learn embeddings for semantic segmentation without actual semantic supervision. Due to these implicitly learned high-level representations, the above self-supervised learning models fall also in the category called *representation learning* [8], since the learned features can represent *e.g.* some specific semantic class.

A third principle is based on automatic label generation. This approach can be used in geometrical problems such as stereo and optical flow. The geometry in these problems can thereby be directly used to get labels and/or supervision. This can be for example done by computing two disparity maps d_0 and d_1 using I_0 and I_1 as a reference image, respectively. Using both disparity maps, the left-right-consistency check [57]

$$|d_0(\mathbf{x}) + d_1(\mathbf{x} + d_0(\mathbf{x}))| < \varepsilon, \tag{2.168}$$

where ε is a hyper-parameter, can be used to determine whether the predictions from the model are plausible or not. We in [86] and Zhou et al. [227] exploited the left-right-consistency to detect reliable predictions of the model and used them for training the model itself. We refer the reader

to Chapter 6 for more details on this variant.

The examples above clearly show that self-supervised learning gains more and more importance. Until now the methods can not yet compete with their fully supervised trained pendants in terms of final performance. However, the gap between self-supervised and fully supervised trained methods became smaller in recent years.

2.6.5 Unsupervised Learning

For completeness, we also briefly mention unsupervised learning here. In unsupervised learning the training dataset does not contain any target labels. Hence, classical unsupervised learning tasks are clustering, density estimation and dimensionality reduction as *e.g.* shown in [12]. One example of an unsupervised learning algorithm is the K-means clustering algorithm [12]. The reader is referred to classical text books [12] for further information on classical unsupervised learning.

The objective function of an unsupervised problem does not use any target labels. However, we can argue that the supervision is done directly by the data itself which classifies the K-means algorithm also as a self-supervised algorithm. Again, we can conclude that unsupervised learning and self-supervised learning are closely related to each other.

2.7 Deep Learning with Neural Networks

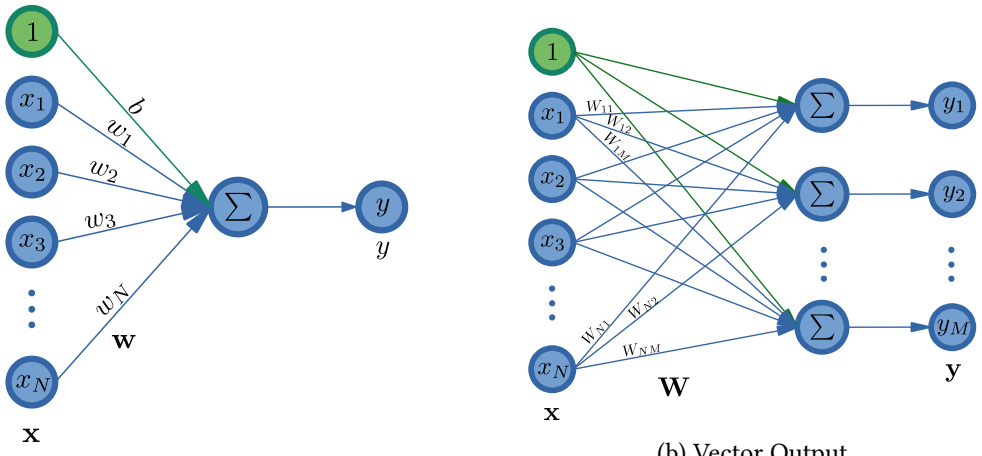
Artificial Neural Networks (ANNs) or short neural networks are nowadays the most successful machine learning models. They are successfully applied to a wide variety of tasks including, but not limited to, image classification [100], stereo [220], optical flow [47], semantic segmentation [116], natural language processing [184]. Although the basic concepts – the perceptron [164], non-linear activations and the backpropagation algorithm [105] – have been invented many years ago, the success story of neural networks truly started only in 2012. Krizhevsky et al. [100] showed in their seminal work on image classification how to utilize the *GPU* to train a *CNN* with a huge dataset containing millions of images. This was the starting point of the modern deep learning era.

We will review the most relevant building blocks of neural networks in the upcoming sections. To this end, we will start with the basic perceptron, proceed to neural networks and show the basic principles to train neural networks.

2.7.1 Perceptron

The perceptron [164] can be considered as a main building block of neural networks. Given an input vector $\tilde{\mathbf{x}} \in \mathbb{R}^N$, the perceptron is a linear model

$$y(\tilde{\mathbf{x}}; \tilde{\mathbf{w}}, b) = \sum_{i=1}^{N} \tilde{w}_i \tilde{x}_i + b, \tag{2.169}$$



(a) Scalar Output

(b) Vector Output

Figure 2.28: Perceptron for scalar output / vector output. The output y / $\mathbf{y}$ is a linear combination of learnable weights $\mathbf{w}$ / $\mathbf{W}$ and and the inputs $\mathbf{x}$. The bias parameter b is captured through the homogenous representation. See text for details.

where $\tilde{\mathbf{w}} \in \mathbb{R}^N$ are the learnable weights and $b \in \mathbb{R}$ is called the bias. Figure 2.28a shows a visualization of a perceptron. The elements $\tilde{x}_1, \ldots, \tilde{x}_N$ are called (artificial) *input neurons* and y is called the *output neuron*. Using *homogeneous coordinates*, we can express the addition in Eq. (2.169) as a vector-vector product and get the concise notation

$$y(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}, \tag{2.170}$$

where $\mathbf{x} \in \mathbb{R}^{N+1}$ is the input $\tilde{\mathbf{x}}$ extended with the homogeneous coordinate

$$\mathbf{x} = \begin{pmatrix} 1 \\ \tilde{\mathbf{x}} \end{pmatrix} = \begin{pmatrix} 1 \\ \tilde{x}_1 \\ \tilde{x}_2 \\ \vdots \\ \tilde{x}_N \end{pmatrix}, \tag{2.171}$$

and $\mathbf{w} \in \mathbb{R}^{N+1}$ is the weight vector including the bias parameter, *i.e.*

$$\mathbf{w} = \begin{pmatrix} b \\ \tilde{\mathbf{w}} \end{pmatrix} = \begin{pmatrix} b \\ \tilde{w}_1 \\ \tilde{w}_2 \\ \vdots \\ \tilde{w}_N \end{pmatrix}. \tag{2.172}$$

We can easily extend the perceptron in order to predict an output vector $\mathbf{y} \in \mathbb{R}^M$ instead of a

scalar. To this end, we replace the weight vector $\mathbf{w}$ by a weight matrix $\mathbf{W} \in \mathbb{R}^{M \times N}$:

$$\mathbf{W} = \begin{pmatrix} \mathbf{w}_0^\top \\ \mathbf{w}_1^\top \\ \vdots \\ \mathbf{w}_M^\top \end{pmatrix} = \begin{pmatrix} b_1 & w_{11} & \dots & w_{1N} \\ b_2 & w_{21} & & \\ \vdots & & \ddots & \vdots \\ b_M & & \dots & w_{MN} \end{pmatrix}. \tag{2.173}$$

This allows to map the N-dimensional input to an M-dimensional output. Both versions of the perceptron are shown in Fig. 2.28.

A perceptron can be used for modeling logical operations such as AND, OR and NOT [164]. However, as shown by Minsky and Papert [133], the perceptron cannot model the XOR operation, because XOR is not linearly separable. In the next section we overcome this problem by introducing multi-layer perceptrons which are often also referred to as fully connected neural networks.

2.7.2 Fully Connected Neural Networks

Fully connected neural networks can be constructed by stacking multiple perceptrons together. The term *fully connected* indicates that each neuron of a layer is connected with each neuron of the next layer. Equivalently, this means that the learnable weight matrices are dense matrices. Neural networks are also often called multi-layer perceptrons, since the perceptron is the main building block.

Let us now describe a fully connected neural network more formally. Note that, similar as in the previous section, we use homogeneous coordinates in the following. To this end, consider the multi-layer model

$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{W}^{(1)}\mathbf{x}, \tag{2.174}$$

where $\mathbf{x}$ is the input and $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ are the learnable weights for the first and second layer, respectively. However, if we investigate Eq. (2.174), we see that we actually did not gain anything. We can easily come up with a new matrix $\mathbf{W}$ computed as the product of $\mathbf{W}^{(2)}$ and $\mathbf{W}^{(1)}$ and see that we have just invented the perceptron again:

$$\tilde{\mathbf{y}} = \mathbf{W}\mathbf{x} = \mathbf{W}^{(2)}\mathbf{W}^{(1)}\mathbf{x} = \mathbf{y}. \tag{2.175}$$

The problem in Eq. (2.175) is that we can always merge multiple *linear* layers into one *linear* layer. The key idea to avoid this problem is the introduction of a *non-linearity*, also-called *activation function*, between the individual layers. This change makes the whole model non-linear and thus it is no longer possible to collapse multiple layers into an equivalent single layer representation. Let us denote the element-wise activation function for the l-th layer as $\sigma^{(l)} : \mathbb{R} \to \mathbb{R}$. Then we can define an L-layer neural network with layer indices $l \in \{1, \dots, L\}$ as

$$\begin{cases} \mathbf{z}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{a}^{(l)} \\ \mathbf{a}^{(l)} = \sigma^{(l)}(\mathbf{z}^{(l)}), \end{cases} \tag{2.176}$$

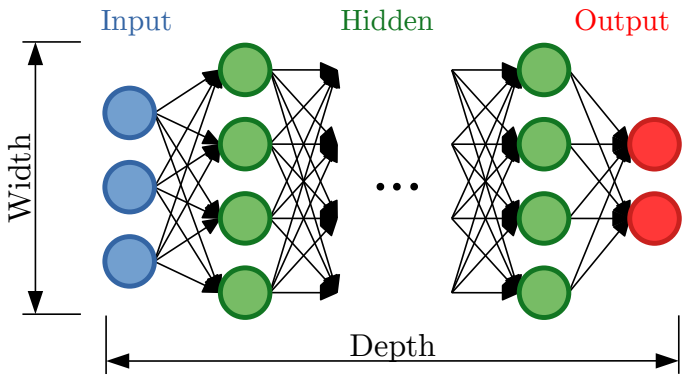


Figure 2.29: A multi-layer perceptron (=neural network) with hidden layers. The input neurons are shown in blue, hidden neurons are shown in green and the output neurons Average shown in red.

where $\mathbf{z}_i^{(l)}$ and $\mathbf{a}_i^{(l)}$ are the output and the activation of the i-th neuron in the l-th layer, respectively, and $\mathbf{W}^{(l)}$ is the homogeneous, learnable weight matrix for the l-th layer. The activation of the 0^{th}-layer is the input, *i.e.* $\mathbf{a}^{(0)} = \mathbf{x}$ and the output is simply the activation of the last layer, *i.e.* $\mathbf{y} = \mathbf{a}^{(L)}$.

A visualization of a fully connected neural network is shown in Fig. 2.29, where the *hidden layers* are the layers between the input and the output layer. If there is more than one hidden layer, then the network is called a *deep* neural network. Note that we did not explicitly visualize the activation functions in Fig. 2.29.

Properties of a neural network are *e.g.* the depth and the width of the network. The depth refers to the number of layers L in the network and the width refers to the number of neurons in the layers (*c.f.* Fig. 2.29).

So far, the activation function σ is just abstractly defined. We will look at different activation functions commonly used in deep learning in the next section.

2.7.3 Activation Functions

We have seen in Section 2.7.2 that stacking linear layers without activation functions together does not increase model complexity because we can always design an equivalent single-layer perceptron as shown in Eq. (2.175). Thus, the second main ingredient of neural networks are the non-linear activation functions. Due to the non-linearity the layers cannot be collapsed into an equivalent single-layer network. Thus, we can increase the complexity arbitrarily by designing *deep* neural networks. In this section, we will briefly show the most important activation functions. Note that this list is not complete. For a more complete list of activation functions we recommend the reader to consult the documentation of a recent machine learning framework such as PyTorch.[2] Usually, they contain an up-to-date list of activation functions, which is not trivial, because activation functions are actively researched.

[2] `https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity`

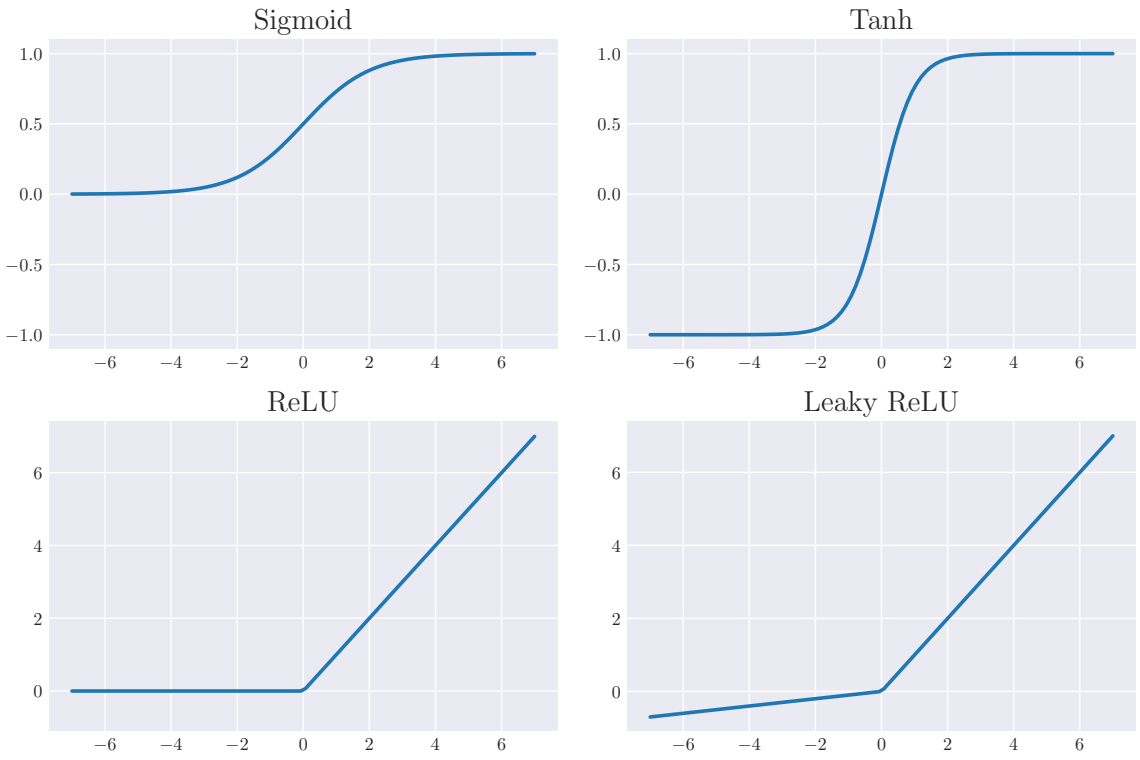


Figure 2.30: Common activation functions.

Classical activation functions The logistic "sigmoid" function (see Fig. 2.30 top-left) and the "tanh" activation function (see Fig. 2.30 top-right) are representatives of classical activation functions. The sigmoid function is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.177}$$

where $x \in \mathbb{R}$ is a scalar input. The sigmoid function squashes the input to the interval $[0, 1]$, and therefore the output of $\sigma(x)$ can be interpreted as a probability. We can also generalize the sigmoid function to the "softmax" function, which can be used to convert an arbitrary vector $\mathbf{x} \in \mathbb{R}^N$ to a probability distribution. This can be done with the softmax function defined as

$$\sigma(\mathbf{x})_i = \frac{e^{-x_i}}{\sum_{j=1}^{N} e^{-x_j}}, \tag{2.178}$$

where $\sigma(\mathbf{x}) \in \Delta^N$, *i.e.* the output is in the N-dimensional simplex and thus can be interpreted as a probability.

As visualized in Fig. 2.30 (top-left) the sigmoid function is not symmetric around the x-axis. In difference, the tanh activation function *is* symmetric around the x-axis (see Fig. 2.30 (top-right)).

The tanh function is defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{2.179}$$

where $x \in \mathbb{R}$ is a scalar input, which is squashed to the interval $[-1, 1]$. *E.g.* LeCun et al. [108] argue that learning works better if the data is centered around zero. This fits perfectly to the definition of the tanh activation function.

These classical activation functions have been the gold standard for a long time. However, modern deep learning papers [62, 65, 121] have shown the problems of the sigmoid and tanh activation function and introduced modern activation functions. We will investigate the most important ones in the next paragraph.

Modern activation functions The currently most often used activation function is the rectified linear unit, usually just abbreviated as ReLU. In difference to the sigmoid and tanh function, the ReLU is a non-saturating function, *i.e.* $\lim_{x\to\infty} \mathrm{ReLU}(x) = \infty$. This can be seen in Fig. 2.30 (bottom-left) and in the definition

$$\mathrm{ReLU}(x) = \max(0, x), \tag{2.180}$$

where $x \in \mathbb{R}$. This non-saturation is beneficial during model training (Section 2.7.5), because the gradient is always non-zero for $x > 0$. However, the ReLU can suffer from the so-called *dying ReLU problem*. To see this, consider a neural network with one hidden layer activated with the ReLU function. If the input vector $\mathbf{x} \in \mathbb{R}^N$ contains only negative entries, *i.e.* $(x_i < 0)_{i=1}^N$, then the gradient of this layer w.r.t. its input is always zero. A zero gradient discontinues learning and therefore the model gets stuck. However, it should also be noted that the shutdown of some neurons can be very useful, *i.e.* when some features should be ignored during further processing.

The ReLU activation function is the basis for many more activation functions such as LeakyReLU [122], the parameterized ReLU (PReLU) [65], the exponential linear unit (ELU) [36], the scaled exponential linear unit (SELU) [83], *etc.* The LeakyReLU function was invented to avoid the dying ReLU problem by using a small slope for the negative interval as well. It is defined as

$$\mathrm{LeakyReLU}(x; \alpha) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}, \tag{2.181}$$

where $x \in \mathbb{R}$ is the input and $\alpha = 0.01$. The LeakyReLU function is plotted in Fig. 2.30 (bottom-right). While the parameter α is fixed in LeakyReLU, in the PReLU α is a learnable parameter.

The modern activation functions are usually used in Deep Neural Network (DNN) [66, 180]. The main reason is that the training usually converges faster due to the better gradient flow. However, the classical activations are still used as well. A standard use-case for the sigmoid or softmax function is for example the conversion of some output signal to a probability distribution. This is for example useful in classification problems.

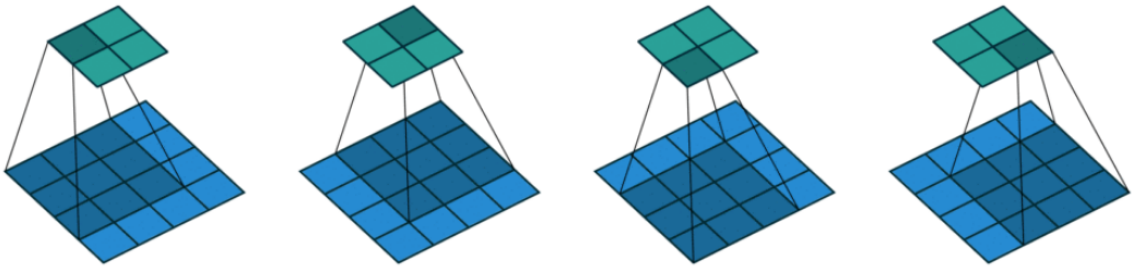


Figure 2.31: Visualization of a convolution. The blue image is convolved with the gray filter and results in the green image. Image courtesy of [188].

2.7.4 Convolutional Neural Networks

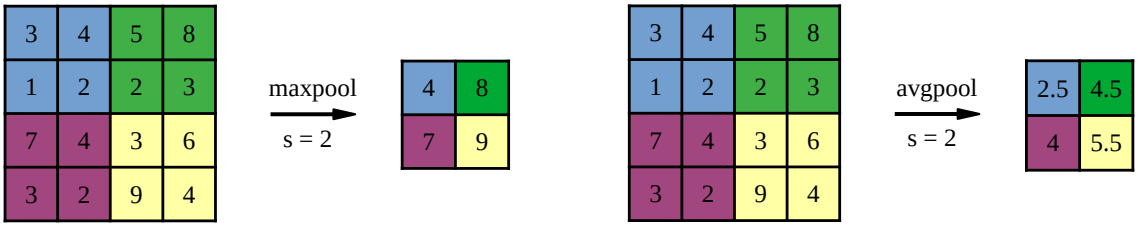
When we work with images, we usually want to share the parameters over the complete image [107]. This is also encouraged by the well-known fact that images are locally very similar. To take advantage of this self-similarity we replace the dense matrix-vector product with a convolution. A convolution is translation invariant and thus very well suited to be applied on images. Formally, it is defined as

$$(f * g)(x, y) = \sum_{i=1}^{M'} \sum_{j=1}^{N'} f(x - i, y - j) g(i, j), \tag{2.182}$$

where f is an image with size $M \times N$ and g is a filter kernel with size $M' \times N'$. In the context of neural networks usually the filter function g covers only a small local region of the image f. Intuitively, Eq. (2.182) states that a local window g is slid over the input image f. For each position in the input image the content in the local window defined by the filter is multiplied and summed up. In other words, the two signals are convolved with each other. This can be interpreted as filtering the image f with the filter g. Figure 2.31 shows a convolution of an image with 4×4 pixels and a filter with 3×3 pixels.

Convolution Layer In Eq. (2.182) we have seen the formal definition of a convolution. We will clarify in this section how convolutions are actually used in *CNNs*. Before we detail the learnable parameters of the convolution operation we are going to map Eq. (2.182) to Eq. (2.176). Therefore, we show how a convolution can be represented in terms of a matrix-vector multiplication. This is exactly the same representation as we have been using for fully connected neural networks shown in Eq. (2.176). To this end, assume we are given an input image $f \colon \Omega \to \mathbb{R}$ and a convolution kernel $g \colon \Omega \to \mathbb{R}$. We assume the image has size $M \times N$ and the convolution kernel has size $U \times V$.

Then, a convolution as defined in Eq. (2.182) can also be implemented using a matrix-vector product. The first option is to represent the convolution kernel as a flattened vector $\mathbf{g} \in \mathbb{R}^{UV}$ and the matrix $\mathbf{F} \in \mathbb{R}^{MN \times UV}$ can be constructed using the input image. Then, the convolution can be performed by taking the product $\mathbf{Fg}$. In the second option we exchange the roles. Here, the vector $\mathbf{f} \in \mathbb{R}^{MN}$ represents the flattened input image and we construct a highly sparse matrix



(a) Max-Pooling (b) Average-Pooling

Figure 2.32: Pooling operations with stride $s = 2$. The pooling operation computes the maximum (Fig. 2.32a) or the average (Fig. 2.32b) over the colored windows.

$\mathbf{G} \in \mathbb{R}^{MN \times MN}$ which represents the convolution kernel. This yields the vector-matrix product $\mathbf{Gf}$.

The last representation reveals that a convolution layer can be seen as a special instance of a fully connected layer. The difference is that a convolution layer has sparse and shared weights while the fully connected layer has dense non-shared weights. This representation is beneficial for computing the gradient w.r.t. the convolution kernel as we will show in Section 2.7.5.

The convolution layer as described above is the simplest form of a convolution used in *CNNs*. There exist many variants such as depth-wise separable convolutions [35], dilated convolutions [214], *etc.*, which are not covered here in detail.

Pooling Layer The second important layer in *CNNs* is the pooling layer. As the name already suggests, this layer is used to condense information in order to get a more compact representation. Let us consider the pooling operation in 2D. We distinguish between two different pooling operations, max-pooling and average pooling. Given an image $f \colon \Omega \subset \mathbb{N} \rightarrow \mathbb{R}$. Then we can define max-pooling as

$$\text{maxpool(f)}|_{\Omega_i} \coloneqq \max_{x \in \Omega_i} f(x) \tag{2.183}$$

and average pooling as

$$\text{avgpool(f)}|_{\Omega_i} \coloneqq \frac{1}{|\Omega_i|} \sum_{x \in \Omega_i} f(x), \tag{2.184}$$

where Ω_i are sub-regions of the image such that $\Omega = \bigcup_{i=1}^{N} \Omega_i$. Figures 2.32a and 2.32b show a visualization of max pooling and average pooling where the sub-regions have a size of 2×2.

Pooling operations are mainly used to reduce the spatial dimensions of feature maps. The interpretation of max pooling is *e.g.* that only the feature with the highest response survives, because this feature might contain most information. The pooling operations are also related to strided convolutions, where the convolution is only applied on predefined pixel positions.

2.7.5 Model Training

Successfully applying a machine learning model to a specific tasks includes the three components i) training data, ii) model architecture and iii) model training. We will investigate these three parts in the upcoming paragraphs.

Training Data Since *ML* and especially deep learning are data-driven approaches, the training data is of special importance. From a theoretical *ML* perspective the training data set is sampled from the true underlying joint distribution of inputs and outputs $p(\mathbf{X}, \mathbf{Y})$, where the capital letters denote random vectors. Thus, the better the training data $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$ represents the true distribution $p(\mathbf{X}, \mathbf{Y})$, the better the final model will perform. Let us consider the following gedankenexperiment in order to demonstrate that more training data is better in general. Imagine a tiny gray-scale image with 4×4 pixels. If every pixel can select one out of 256^3 gray values, then there exist $16^{256} \approx 10^{308}$ (!) different gray-scale images with size 4×4. This huge number is even more impressive if we set it into relation with the number of atoms in the universe, which is estimated to be around 10^{80} and thus tremendously smaller than 10^{308}. Coming back to the training data in *ML* this tells us basically that (i) there will never be enough training data available and (ii) more data represents $p(\mathbf{X}, \mathbf{Y})$ better and is thus beneficial for training.

Model Architecture In deep learning, the model architecture is usually designed by domain experts. However, nowadays some architectures are considered *standard* architectures in the deep learning community. These standard models can be directly used as *e.g.* general purpose feature extractors in composite models. Examples of standard architectures are *e.g.* the `ResNet` family [66], the `U-Nets` [116, 163] or the `VGG-Net` [180]. There are also some attempts to automate the process of finding better architectures. Such approaches fall into the research area of Neural Architecture Search (NAS) which is a sub-field of Automated Machine Learning (AutoML). The idea of automatically finding better architectures is interesting. However, especially in the context of deep learning *NAS* approaches are extremely costly in terms of computations. The computation time is usually measured in *GPU*-days and it often takes hundreds or even thousands of *GPU*-days until a suitable architecture is found [208].

Model Training Finding the best parameters for a given model is at the core of every machine learning algorithm. The goal of model training is to find parameters for a model[4] which explains the given training data best. Machine learning theory and mathematical optimization provide us with tools which we can use to find better parameters for our models. We can quantify the *risk* of our model $f : \mathcal{X} \to \mathcal{Y}$ by the expectation of the loss function ℓ over the *data-generating distribution*:

$$R(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{X},\mathbf{Y})\sim p(\cdot,\cdot)}[\ell(f(\mathbf{X};\boldsymbol{\theta}), \mathbf{Y})], \tag{2.185}$$

where $\boldsymbol{\theta}$ are the parameters of the model. The quantity R in Eq. (2.185) is also known as the *true risk*. However, in practice we can hardly ever compute the expectation in the true risk, because we do not know the distribution p. Instead, as an approximation we can compute the empirical risk

$$\hat{R}(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{X},\mathbf{Y})\sim \hat{p}(\cdot,\cdot)}[\ell(f(\mathbf{X};\boldsymbol{\theta}), \mathbf{Y})], \tag{2.186}$$

[3] Standard images are quantized with 8 bit per pixel $\Rightarrow 2^8 = 256$ different values.
[4] The word "model" here means neural networks and *CNNs*, respectively.

where we use the hat to denote an estimated quantity.

$$\hat{p}(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} \delta_{\mathbf{x}=\mathbf{x}_i} \delta_{\mathbf{y}=\mathbf{y}_i} \tag{2.187}$$

is the empirical distribution defined by the underlying dataset and $\hat{R}$ is the empirical risk. Expanding the expectation in Eq. (2.186) we get

$$\mathcal{L}(\boldsymbol{\theta}) \coloneqq \hat{R}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \ell\big(f\big(\mathbf{x}^{(i)}; \boldsymbol{\theta}\big), \mathbf{y}^{(i)}\big), \tag{2.188}$$

where we have inserted the dataset $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$ into $\hat{p}(\cdot, \cdot) = p(\mathcal{D})$ and defined the total loss $\mathcal{L}$ to be the empirical risk. Equation (2.188) can be computed and thus our goal is to find the optimal parameters $\boldsymbol{\theta}^*$ by solving the optimization problem

$$\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}(\boldsymbol{\theta}) \coloneqq \frac{1}{N} \sum_{i=1}^{N} \ell\big(h\big(\mathbf{x}^{(i)}; \boldsymbol{\theta}\big), \mathbf{y}^{(i)}\big) \right\}, \tag{2.189}$$

which is known as Empirical Risk Minimization (ERM). *ERM* is the most central task in every machine learning model.

Let us next investigate how we can approach the *ERM* problem defined in Eq. (2.189). To this end, we first note that Eq. (2.189) is actually an instance of the continuous optimization problem in Eq. (2.142). As we have seen in Section 2.5.2 we can use gradient-based optimization methods such as *e.g.* the gradient method defined in algorithm 7 to iteratively compute the optimizer $\boldsymbol{\theta}^*$. Note that $\mathcal{L}(\boldsymbol{\theta})$ is a non-convex function in the parameters $\boldsymbol{\theta}$ and thus we cannot guarantee to find the global optimum of the function. Instead, we will only find a local minimum or a saddle point. However, it turns out that the parameters found with gradient-based methods work very well in practice. Gradient-based methods require, as the name already suggests, a gradient to update the parameters. Therefore, we need to compute

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \tag{2.190}$$

where again $\boldsymbol{\theta}$ represents all the parameters of our network. Let us now consider a convolutional neural network with L layers. To this end, we reuse the formal definition given in Eq. (2.176) and

expand it to

$$\mathbf{y} = \mathbf{a}^{(L)} = \sigma^{(L)}(\mathbf{W}^{(L)} \quad \cdots \quad \overbrace{\sigma^{(2)}(\mathbf{W}^{2} \overbrace{\sigma^{(1)}(\underbrace{\mathbf{W}^{(1)}\mathbf{x}}_{\mathbf{z}^{(1)}})}^{\mathbf{a}^{(1)}})}^{\mathbf{a}^{(2)}}), \tag{2.191}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\mathbf{z}^{(2)}}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\mathbf{z}^{(L)}}$$

where we have used the homogeneous representation for $\mathbf{W}^{(l)}$ and $\mathbf{x}$ in order to implicitly handle the bias parameter. We name $\mathbf{z}^{(l)}$ and $\mathbf{a}^{(l)}$ the output and the activation of the l-th layer, respectively. The parameters of our network defined in Eq. (2.191) are the weight and bias parameters of all layers captured by $\boldsymbol{\theta} = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}\}$. Thus, we can rewrite Eq. (2.190) to

$$\nabla_{\{\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(L)}\}} L(\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}), \tag{2.192}$$

which enables us to split up the computation of the gradients. To ease the notation, we define

$$\nabla_{\mathbf{W}^{(l)}} L(\mathbf{W}^{(l)}) \coloneqq \frac{\partial L}{\partial \mathbf{W}^{(l)}} = \begin{pmatrix} \frac{\partial L}{\partial W_{11}^{(l)}} & \frac{\partial L}{\partial W_{12}^{(l)}} & \cdots & \frac{\partial L}{\partial W_{1N}^{(l)}} \\ \frac{\partial L}{\partial W_{21}^{(l)}} & \ddots & & \vdots \\ \vdots & & & \\ \frac{\partial L}{\partial W_{M1}^{(l)}} & \cdots & & \frac{\partial L}{\partial W_{MN}^{(l)}} \end{pmatrix}. \tag{2.193}$$

Next, equipped with the notation defined in Eq. (2.193) we can compute the gradient for the parameter blocks $\mathbf{W}^{(l)}$ using the chain rule, *i.e.*

$$\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \frac{\partial L}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}} \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{a}^{(L-1)}} \quad \cdots \quad \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}}. \tag{2.194}$$

Due to the structure of a neural network as defined in Eq. (2.191) we have only three different terms in Eq. (2.194) for which we must compute the derivatives. These derivatives are given by

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \sigma'^{(l)}(\mathbf{z}^{(l)}) \qquad \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{a}^{(l-1)}} = \mathbf{W}^{(l)^\top} \qquad \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}} = \mathbf{a}^{(l-1)^\top}, \tag{2.195}$$

where l is the layer index. The chain rule in Equation (2.194) reveal also that the gradient is computed by propagating the error from the output to the input. If we also compute the gradients in reverse order, *i.e.* $\{\mathbf{W}^{(L)}, \ldots, \mathbf{W}^{(1)}\}$, we observe that we can reuse intermediate results. This allows us to define an efficient algorithm for computing the gradients. We therefore define the

Algorithm 14: Backpropagation Algorithm

Input: Outputs $\mathbf{z}^{(l)}$, activations $\mathbf{a}^{(l)}$, parameters $\left\{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}\right\}$

Result: Gradients of parameters $\left\{\frac{\partial L}{\partial \mathbf{W}^{(1)}}, \ldots, \frac{\partial L}{\partial \mathbf{W}^{(L)}}\right\}$

for $l = \{L, L-1, \ldots, 1\}$ **do**
  /* Compute current delta */
  if $l == L$ **then**
    $\boldsymbol{\delta} = \sigma'^{(L)}(\mathbf{z}^{(L)}) \odot \frac{\partial L}{\partial \mathbf{a}^{(L)}}$
  else
    $\boldsymbol{\delta} = \sigma'^{(l)}(\mathbf{z}^{(l)}) \odot \mathbf{W}^{(l+1)^\top} \boldsymbol{\delta}^{(l+1)}$
  end if
  /* Compute gradient for layer l */
  $\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}\, \mathbf{a}^{(l-1)^\top}$
end for

intermediate results $\boldsymbol{\delta}$ recursively as

$$\boldsymbol{\delta}^{(l)} = \sigma'^{(l)}(\mathbf{z}^{(l)}) \odot \begin{cases} \frac{\partial L}{\partial \mathbf{a}^{(L)}} & \text{if } l = L, \\ \mathbf{W}^{(l+1)^\top} \boldsymbol{\delta}^{(l+1)} & \text{else,} \end{cases} \tag{2.196}$$

where the derivative of the activation function σ is point-wise applied to the output $\mathbf{z}$ of the same layer l and $\odot$ denotes the point-wise multiplication operation. In order to compute the sought gradients, we use Eq. (2.196) and get

$$\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l)} \mathbf{a}^{(l-1)^\top}. \tag{2.197}$$

Note that by definition of Eq. (2.196) the gradients are propagated *backwards* through the network. We actually derived the backpropagation (backprop) algorithm [105] in Equations (2.196) and (2.197) and now understand that this algorithm is simply an efficient way to compute the gradients of the parameters of neural networks by intelligently reusing intermediate results. Algorithm 14 shows the complete algorithm.

Now, we have an algorithm to efficiently compute the gradient in Eq. (2.190) and can come back to conduct the optimization. Regardless of the used optimization algorithm, we need to specify a step-width often referred to as *learning rate* in the machine learning literature. Unfortunately, finding an appropriate learning rate is a difficult problem in its own. Additionally, since neural networks are non-convex and non-linear functions, there are no theoretical guidelines available on how to select a good learning rate. Thus, finding an appropriate learning rate is usually done by a machine learning expert. On one hand the non-convexity of neural networks enables to train very powerful models, but on the other hand any theoretical guarantees are lost. Another difficulty of non-convex functions is to find a "good" *initialization* for the parameters. Depending on the initialization, the optimization algorithm will end up in a different stationary point. But of

Algorithm 15: Adam Optimizer as presented in [81]

Input: step size α
Input: Exponential decay rates for moment estimates $\beta_1, \beta_2 \in [0, 1)$
Input: Stochastic objective function $f(\theta)$
Input: Initial parameters θ_0
Result: Parameters θ_t

/* β_1^t and β_2^t denotes β_1 and β_2 to the power of t. */
$m_0 = 0$; /* Initialize 1^{st} moment vector */
$v_0 = 0$; /* Initialize 2^{nd} moment vector */
$t = 0$; /* Initialize timestep */
while θ_t *not converged* **do**
  $t = t + 1$
  $g_t = \nabla_\theta f_t(\theta_{t-1})$; /* Get gradient at timestep t */
  $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$; /* Update biased first moment estimate */
  $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$; /* Update biased second raw moment estimate */
  $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$; /* Compute bias-corrected first moment estimate */
  $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$; /* Compute bias-corrected second raw moment estimate */
  $\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$; /* update parameters */
end while

course, we want to end up in a "good" local minimum after the optimization. The term "good" is here vaguely defined. This is simply, because there are no precise theoretical measurements which could be used to define good and bad. Instead "good" means here that our model performs as expected after the training. Despite of the problem with the learning rate and the initialization, the literature shows that it is often possible in practice to handle all these unknowns. However, this also shows that from a scientific point of view there are still a lot of theoretical questions open.

The Adam optimizer The Adaptive Momentum (Adam) optimizer [81] is a relatively new optimization algorithm specifically designed to be used in deep learning problems. The main advantage compared to classical optimization algorithms shown in Section 2.5.2 is the adaptive weighting of the learning rate. To this end, the algorithm adaptively estimates the first and second moment of the gradient. The first momentum corresponds to the mean of the gradient g and the second momentum is the uncentered[5] variance of the gradient given by

$$m = \mathbb{E}[g] = \frac{1}{T}\sum_{t=1}^{T} g_t \qquad v = \mathbb{E}[g^2] = \frac{1}{T}\sum_{t=1}^{T} g_t^2, \tag{2.198}$$

[5]Not to be confused with the variance which is centered around the mean m, *i.e.* $v = \mathbb{E}[(g - m)^2]$.

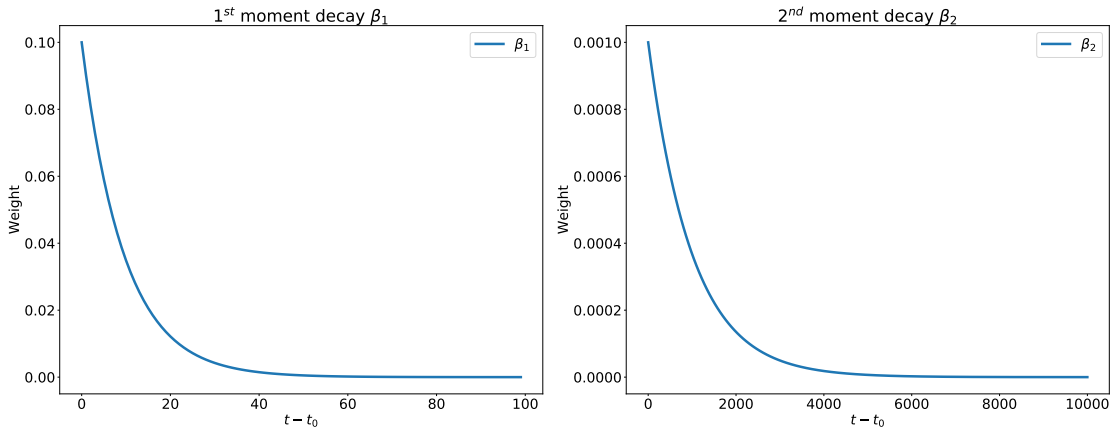


Figure 2.33: Exponential decay in the *Adam* optimizer.

where g_t and g_t^2 are the gradient and the squared gradient at time step t, respectively. Equation (2.198) can be recursively computed in an online setting with

$$m_t = \frac{1}{t} \cdot \begin{cases} g_1 & \text{if } t = 1, \\ g_t + m_{t-1} \cdot (t-1) & \text{if } t > 1, \end{cases} \tag{2.199}$$

where t is the current time step. However, the importance of every sample in Eqs. (2.198) and (2.199) is uniformly distributed. In the learning setting, we might want to increase the importance of recent samples and decrease the importance of old samples. Kingma and Ba [81] therefore suggest to use an exponentially decaying moving average which enables to make the moments adaptive to changes in the gradient. An exponentially moving average is defined as

$$m_t = \begin{cases} g_1 & \text{if } t = 1, \\ \beta g_t + (1-\beta) m_{t-1} & \text{if } t > 1. \end{cases} \tag{2.200}$$

Figure 2.33 shows the effect of the exponential decay at time step t of a sample drawn at time step t_0 using the default values $\beta_1 = 0.9$ for the first moment and $\beta_2 = 0.999$ for the second moment, respectively.

The descent direction at time step t is then computed by

$$d_t = \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}, \tag{2.201}$$

where $\hat{m}_t$ and $\hat{v}_t$ denotes the bias corrected first and second raw moments, respectively, and ε is a small hyper-parameter to avoid a division by zero. Hence, the descent direction used in the *Adam* optimizer is given by the variance-normalized, exponentially smoothed mean gradient. The smoothing and normalization ensures that the magnitude of d_t does not fluctuate significantly.

Thus, also the magnitude of the actual update step, $\alpha \cdot d_t$, is similar in all time steps. In difference, the actual step size in traditional optimization algorithms (*c.f.* Section 2.5.2) is directly influenced by the gradient, *i.e.* $\alpha \cdot g_t$. Thus, in general the *Adam* optimizer is more stable in practice compared to traditional optimization algorithms. This is especially important in the early stage in training when the randomly initialized parameters can cause large gradients or in the presence of strong outliers.

Another advantage of the *Adam* optimizer is the tracking of the running first and second moments for every (scalar) element individually. To understand this advantage we consider training a *CNN* with multiple hidden layers. The magnitude of the gradient can be significantly different in different layers. In the classical setting the maximal learning rate is limited by the largest gradient magnitude. Contrary, since the *Adam* optimizer performs element-wise rescaling the effective update will be similar for *all* layers. Thus, if we compare multiple learning algorithms for training *CNNs* in practice, the *Adam* optimizer yields often the best results although the theoretical convergence rate is even worse to the convergence of the gradient method. The very good performance makes the *Adam* optimizer often the number one choice for training *CNNs*.

3

Hybrid CNN-CRF Model for Stereo

Deep Learning (DL) based models have been successfully applied to many problems in computer vision. This is especially true for tasks where it is sufficient to predict only a single value for the entire image. Back in 2016, when we were working on the paper presented in this section, the community started to apply DL to geometric problems such as stereo. The seminal work of Žbontar and LeCun [220] therefore suggested learning the features for matching using Convolutional Neural Networks (CNNs). They replaced the hand-crafted features with the learned features, integrated them into the stereo pipeline and achieved a significant performance gain on several stereo benchmarks. However, the process relied still on a chain of hand-crafted post-processing steps. The authors used e.g. *a Conditional Random Field (CRF) during post-processing, but it was not possible to train the matching part jointly with the CRF in an end-to-end fashion. That was exactly the motivation for this paper. If we are able to enable end-to-end training with hybrid CNN + CRF models for stereo, we can i) avoid all hand-crafted post-processing steps that make the entire model learnable, ii) use much smaller networks while achieving a comparable performance and iii) integrate prior knowledge directly through the CRF and thus gain interpretability of the entire model.*

This work was presented at CVPR 2017 in Honolulu, Hawaii.

Contents

3.1 Introduction **82**
3.2 Related Work **84**
3.3 CNN-CRF Model **85**
3.4 Training **87**
3.5 Experiments **94**
3.6 Conclusion **101**

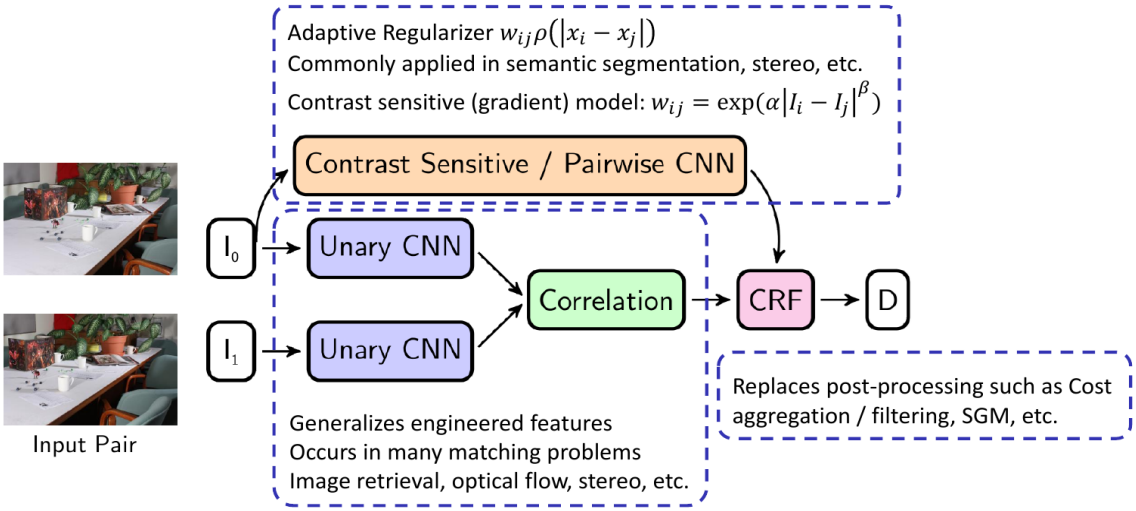


Figure 3.1: Architecture: A convolutional neural network, which we call *Unary-CNN* computes features of the two images for each pixel. The features are compared using a *Correlation* layer. The resulting matching cost volume becomes the unary cost of the *CRF*. The pairwise costs of the CRF are parametrized by edge weights, which can either follow a usual contrast sensitive model or estimated by the *Pairwise-CNN*.

3.1 Introduction

Stereo matching is a fundamental low-level vision problem. It is an ill-posed inverse problem, asking to reconstruct the depth from a pair of images. This requires robustness to all kinds of visual nuisances as well as a good prior model of the 3D environment. Prior to deep neural network data-driven approaches, progress had been made using global optimization techniques [93, 103, 160, 171, 209] featuring robust surface models and occlusion mechanisms. Typically, these methods had to rely on engineered cost matching and involved choosing a number of parameters experimentally.

Recent deep CNN models for stereo [33, 119, 220] learn from data to be robust to illumination changes, occlusions, reflections, noise, *etc.* A deep and possibly multi-scale architecture is used to leverage the local matching to a global one. However, also deep CNN models for stereo rely a lot on post-processing, combining a set of filters and optimization-like heuristics, to produce final accurate results.

In this work we combine CNNs with a discrete optimization model for stereo. This allows complex local matching costs and parametrized geometric priors to be put together in a global optimization approach and to be learned end-to-end from the data. Even though our model contains CNNs, it is still easily interpretable. This property allows us to shed more light on the learning our network performs. We start from a CRF formulation and replace all hand-crafted terms with learned ones.

We propose a hybrid CNN-CRF model illustrated in Fig. 3.1. Our *Unary-CNN* computes local features of both images which are then compared in a fixed correlation metric. Our *Pairwise-CNN* can additionally estimate contrast-sensitive pairwise costs in order to encourage or discourage

label jumps. Using the learned unary and pairwise costs, the CRF tries to find a joint solution optimizing the total sum of all unary and pairwise costs in a 4-connected graph. This model generalizes existing engineered approaches in stereo as well as augment existing fully learned ones. The *Unary-CNN* straightforwardly generalizes manually designed matching costs such as those based on differences of colors, sampling-insensitive variants [11], local binary patterns (*e.g.*, Census transform [216]), *etc.* The *Pairwise-CNN* generalizes a contrast-sensitive regularizer [18], which is the best practice in MRF/CRF models for segmentation and stereo.

To perform inference in the CRF model we apply the fast method of [177], which improves over heuristic approaches combining multiple post-processing steps as used in [33, 119, 220]. We deliberately chose not to use any post-processing in order to show that most of the performance gain through post-processing can be covered by a well-trained CRF model. While previously, methods based on LP-relaxation were considered prohibitively expensive for stereo, [177] reports a near real-time performance, which makes this choice definitely faster than a full deep architecture [220] and competitive in speed with inference heuristics such as SGM [68], MGM [52], *etc.*

We can train the complete model shown in Fig. 3.1 using the structured support vector machine (SSVM) formulation and propagating its subgradient through the networks. Training a non-linear CNN+CRF model of this scale is a challenging problem that has not been addressed before. Since at test time the inference is applied to complete images, we train it on complete images as well. This is in contrast to the works [119, 218, 220] which sample patches for training. The SSVM approach optimizes the inference performance on complete images of the training set more directly. While with the maximum likelihood it is important to sample hard negative examples (hard mining) [179], the SSVM determines labellings that are hard to separate as the most violated constraints.

We observed that the hybrid CNN+CRF network performs very well already with shallow CNN models, such as 3-7 layers. With the CRF layer the generalization gap is much smaller (less overfitting) than without. Therefore a hybrid model can achieve a competitive performance using much fewer parameters than the state of the art. This leads to a more compact model and a better utilization of the training data.

We report competitive performance on benchmarks using a shallow hybrid model. Qualitative results demonstrate that our model is often able to delineate object boundaries accurately and it is also often robust to occlusions, although our CRF did not include explicit occlusion modeling.

Contribution We propose a hybrid CNN+CRF model for stereo, which utilizes the expressiveness of CNNs to compute good unary- as well as pairwise-costs and uses the CRF to easily integrate long-range interactions. We propose an efficient approach to train our CNN+CRF model. The trained hybrid model is shown to be fast and yields competitive results on challenging datasets. We do not use any kind of post-processing.

3.2 Related Work

CNNs for Stereo Most related to our work are CNN matching networks for stereo proposed by [33, 119] and the *fast* version of [220]. They use similar architectures with a siamese network [22] performing feature extraction from both images and matching them using a fixed correlation function (product layer). Parts of our model (see Fig. 3.1) denoted as *Unary-CNN* and *Correlation* closely follow these works. However, while [33, 119, 220] train by sampling matching and non-matching image patches, following the line of work on more general matching / image retrieval, we train from complete images. Only in this setting it is possible to extend to a full end-to-end training of a model that includes a CRF (or any other global post-processing) optimizing specifically for the best performance in the dense matching. The *accurate* model of [220] implements the comparison of features by a fully connected NN, which is more accurate than their *fast* model but significantly slower. All these methods make an extensive use of post-processing steps that are not jointly-trainable with the CNN: [220] applies cost cross aggregation, semi-global matching, subpixel enhancement, median and bilateral filtering; [119] uses window-based cost aggregation, semi-global matching, left-right consistency check, subpixel refinement, median filtering, bilateral filtering and slanted plane fitting; [33] uses semi-global matching, left-right consistency check, disparity propagation and median-filtering. Experiments in [119] comparing bare networks without post-processing show that their fixed correlation network outperforms the *accurate* version of [220].

CNN Matching General purpose matching networks are also related to our work. [218] used a matching CNN for patch matching, [48] used it for optical flow and [126] used it for stereo, optical flow and scene flow. Variants of networks [48, 126] have been proposed that include a correlation layer explicitly; however, it is then used as a stack of features and followed by up-convolutions regressing the dense matching. Overall, these networks have a significantly larger number of parameters and require a lot of additional synthetic training data.

Joint Training (CNN+CRF training) End-to-end training of CNNs and CRFs is helpful in many applications. The fully connected CRF [99], performing well in semantic segmentation, was trained jointly in [30, 225] by unrolling iterations of the inference method (mean field) and backpropagating through them. Unfortunately, this model does not seem to be suitable for stereo because typical solutions contain slanted surfaces and not piece-wise constant ones (the filtering in [99] propagates information in fronto-parallel planes). Instead simple heuristics based on dynamic programming such as SGM [68] / MGM [52] are typically used in engineered stereo methods as post-processing. However they suffer from various artifacts as shown in [52]. A trained inference model, even a relatively simple one, such as dynamic programming on a tree [156], can become very competitive. Scharstein [170] and Pal et al. [146] have considered training CRF models for stereo, linear in parameters. To the best of our knowledge, training of inference techniques with CNNs has not yet been demonstrated for stereo. We believe the reason for that is the relatively slow inference for models over pixels with hundreds of labels. Employing the method proposed in [177], which is a variant of a LP-relaxation on the GPU, allows us to

overcome this limitation. In order to train this method we need to look at a suitable learning formulation. Specifically, methods approximating marginals are typically trained with variants of approximate maximum likelihood [1, 82, 113, 142, 146, 170]. Inference techniques whose iteration can be differentiated can be unrolled and trained directly by gradient descent [115, 144, 145, 159, 173, 190, 225]. Inference methods based on LP relaxation can be trained discriminatively, using a structured SVM approach [29, 56, 97, 192], where parameters of the model are optimized jointly with dual variables of the relaxation (blended learning and inference). We discuss the difficulty of applying this technique in our setting (memory and time) and show that instead performing stochastic approximate subgradient descent is more feasible and practically efficient.

3.3 CNN-CRF Model

In this section we describe the individual blocks of our model (Fig. 3.1) and how they connect.

We consider the standard rectified stereo setup, in which epipolar lines correspond to image rows. Given the left and right images I^0 and I^1, the left image is considered as the *reference* image and for each pixel we seek to find a matching pixel of I^1 at a range of possible disparities. The disparity of a pixel $i \in \Omega = \text{dom}(I)^0$ is represented by a discrete label $x_i \in \mathcal{L} = \{0, \dots L-1\}$.

The *Unary-CNN* extracts dense image features for I^0 and I^1 respectively, denoted as $\phi^0 = \phi(I^0; \theta_1)$ and $\phi^1 = \phi(I^1; \theta_1)$. Both instances of the *Unary-CNN* in Fig. 3.1 share the parameters θ_1. For each pixel, these extracted features are then correlated at all possible disparities to form a correlation-volume (a matching confidence volume) $p \colon \Omega \times \mathcal{L} \to [0, 1]$. The confidence $p_i(x_i)$ is interpreted as how well a window around pixel i in the first image I^0 matches to the window around pixel $i + x_i$ in the second image I^1. Additionally, the reference image I^0 is used to estimate contrast-sensitive edge weights either using a predefined model based on gradients, or using a trainable pairwise CNN. The correlation volume together with the pairwise weights are then fused by the CRF inference, optimizing the total cost.

3.3.1 Unary CNN

We use 3 or 7 layers in the *Unary-CNN* and 100 filters in each layer. The filter size of the first layer is (3×3) and the filter size of all other layers is (2×2). We use the tanh activation function after all convolutional layers. Using tanh i) makes training easier, *i.e.*, there is no need for intermediate (batch-)normalization layers and ii) keeps the output of the correlation-layer bounded. Related works [3, 23] have also found that tanh performs better than ReLU for patch matching with correlation.

3.3.2 Correlation

The cross-correlation of features ϕ^0 and ϕ^1 extracted from the left and right image, respectively, is computed as

$$p_i(k) = \frac{e^{\langle \phi_i^0, \phi_{i+k}^1 \rangle}}{\sum_{j \in \mathcal{L}} e^{\langle \phi_i^0, \phi_{i+j}^1 \rangle}} \quad \forall i \in \Omega, \forall k \in \mathcal{L}. \tag{3.1}$$

Hence, the correlation layer outputs the softmax normalized scalar products of corresponding feature vectors. In practice, the normalization fixes the scale of our unary-costs which helps to train the joint network. Since the correlation function is homogeneous for all disparities, a model trained with some fixed number of disparities can be applied at test time with a different number of disparities. The *pixel-wise independent estimate* of the best matching disparity

$$x_i \in \arg\max_k \; p_i(k) \tag{3.2}$$

is used for the purpose of comparison with the full model.

3.3.3 CRF

The CRF model optimizes the total cost of complete disparity labelings,

$$\min_{x \in \mathcal{X}} \big(f(x) := \sum_{i \in \mathcal{V}} f_i(x_i) + \sum_{ij \in \mathcal{E}} f_{ij}(x_i, x_j)\big). \tag{3.3}$$

where $\mathcal{V}$ is the set of all nodes in the graph, *i.e.*, the pixels, $\mathcal{E}$ is the set of all edges and $\mathcal{X} = \mathcal{L}^{\mathcal{V}}$ is the space of labelings. Unary terms $f_i \colon \mathcal{L} \to \mathbb{R}$ are set as $f_i(k) = -p_i(k)$, the matching costs. The pairwise terms $f_{ij} \colon \mathcal{L} \times \mathcal{L} \to \mathbb{R}$ implement the following model:

$$f_{ij}(x_i, x_j) = w_{ij}\rho(|x_i - x_j|; P_1, P_2). \tag{3.4}$$

The weights w_{ij} may be set either as manually defined contrast-sensitive weights [17]:

$$w_{ij} = \exp(-\alpha|I_i - I_j|^{\beta}) \qquad \forall ij \in E, \tag{3.5}$$

allowing cheaper disparity jumps across strong image gradients, or using the learned model of the *Pairwise-CNN*. The function ρ is a robust penalty function defined as

$$\rho(|x_i - x_j|) = \begin{cases} 0 & \text{if } |x_i - x_j| = 0, \\ P_1 & \text{if } |x_i - x_j| = 1, \\ P_2 & \text{otherwise,} \end{cases} \tag{3.6}$$

popular in stereo [69]. Cost P_1 penalizes small disparity deviation of one pixel representing smooth surfaces and P_2 penalizes larger jumps representing depth discontinuities. We use only pairwise-interactions on a 4-connected grid.

Inference Although the direct solution of (3.3) is intractable [111], there are a number of methods to perform approximate inference [29, 94] as well as related heuristics designed specifically for stereo such as [52, 69]. We apply our dual minorize-maximize method (`Dual_MM`) [177], which is sound because it is based on LP-relaxation, similar to TRW-S [94], and massively parallel, allowing a fast GPU implementation.

We give a brief description of Dual_MM, which will also be needed when considering training. Let f denote the concatenated *cost vector* of all unary and pairwise terms f_i, f_{ij}. The method starts from a decomposition of f into horizontal and vertical chains, $f = f^1 + f^2$ (namely, f^1 includes all horizontal edges and all unary terms and f^2 all vertical edges and zero unary terms). The value of the minimum in (3.3) is lower bounded by

$$\max_{\lambda} \big(D(\lambda) := \min_{x^1}(f^1 + \lambda)(x^1) + \min_{x^2}(f^2 - \lambda)(x^2)\big), \tag{3.7}$$

where λ is the vector of Lagrange multipliers corresponding to the constraint $x^1 = x^2$. The bound $D(\lambda) \leq$ (3.3) holds for any λ, however it is tightest for the optimal λ maximizing the sum in the brackets. The Dual_MM algorithm performs iterations towards this optimum by alternatively updating λ considering at a time either all vertical or horizontal chains, processed in parallel. Each update monotonously increases the lower bound (3.7). The final solution is obtained as

$$x_i \in \arg\min_{k}(f_i^1 + \lambda_i)(k), \tag{3.8}$$

i.e., similar to (3.2), but for the reparametrized costs $f^1 + \lambda$. If the inference has converged and the minimizer x_i in (3.8) is unique for all i, then x is the optimal solution to the energy minimization (3.3) [96, 206].

3.3.4 Pairwise CNN

In order to estimate edge weights with a pairwise CNN, we use a 3-layer network. We use 64 filters with size (3×3) and the tanh activation function in the first two layers to extract some suitable features. The third layer maps the features of pixel i to weights $(w_{ij} \mid ij \in E)$ corresponding to the two edge orientations, where we use the absolute value function as activation. This ensures that the pairwise costs are always larger than 0 and that our *Pairwise-CNN* has the ability to scale the output freely. In practice this is desirable because it allows us to automatically learn the optimal trade-off between data-fidelity and regularization. The parameters of this network will be denoted as θ_2. The weights w can be stored as a 2-channel image (one channel per orientation). They generalize over the manually defined contrast-sensitive weights defined in (3.5) in the pairwise-terms f_{ij} (3.4). Intuitively, this means the pairwise network can learn to apply the weights w adaptively based on the image content in a wider neighborhood. The values P_1, P_2 remain as global parameters. Fig. 3.2 shows an example output of the *Pairwise-CNN*.

3.4 Training

One major goal of this work is the end-to-end training of the complete model in Fig. 3.1. For the purpose of comparison of different components we train 3 types of models, of increasing generality:

- Pixel-wise *Unary-CNN*: model in which CRF interactions are set to zero and *Pairwise-CNN*

(a) Learned (b) Fixed

Figure 3.2: Learned vs fixed pairwise costs: Visualization of the pairwise costs between two neighboring pixels in horizontal direction using the learned *Pairwise-CNN* (left) and a fixed edge-function (right). Dark pixels indicate a low cost for changing the label and bright pixels indicate a high cost for a label-switch. Note, how the dark pixels follow object outlines (where depth discontinuities are likely) and how texture-edges tend to be suppressed (*e.g.*, on the floor) in the learned version.

is switched off.

- Joint *Unary-CNN* +CRF model in which the *Pairwise-CNN* is fixed to replicate exactly the contrast-sensitive model (3.5). Trained parameters are: *Unary-CNN* and global parameters P_1, P_2.

- Joint model with trained *Unary-CNN* and *Pairwise-CNN* (=complete model). Trained Parameters are: *Unary-CNN*, *Pairwise-CNN* and global parameters P_1, P_2.

3.4.1 Training Unary CNN in the Pixel-wise Model

For the purpose of comparison, we train our *Unary-CNN* in a pixel-wise mode, similarly to [33, 119, 220]. For this purpose we set the CRF interactions to zero (*e.g.*, by letting $P_1 = P_2 = 0$), in which case the resulting decision degenerates to the pixel-wise independent argmax decision rule Eq. (3.2). Training such models can be formulated in different ways, using gradient of the likelihood / cross-entropy [119, 219], reweighed regression [33] or hinge loss [219]. Following [119, 219] we train parameters of the *Unary-CNN* θ_1 using the cross-entropy loss,

$$\min_{\theta_1} \sum_{i \in \Omega} \sum_{k \in \mathcal{X}} p_i^{gt}(k) \log p_i(k; \theta_1), \tag{3.9}$$

where $p_i^{gt}(k)$ is the one-hot encoding of the ground-truth disparity for the i-th pixel.

3.4.2 Training Joint Model

We apply the structured support vector machine formulation, also known as the maximum margin Markov network [187, 192], in a non-linear setting. After giving a short overview of the SSVM approach we discuss the problem of learning when no exact inference is possible. We argue that the blended learning and inference approach of [29, 97] is not feasible for models of our size. We then discuss the proposed training scheme approximating a subgradient of a fixed number of iterations of `Dual_MM`.

SSVM Assume that we have a training sample consisting of an input image pair $I = (I^0, I^1)$ and the true disparity x^*. Let x be a disparity prediction that we make. We consider an additive loss function

$$l(x, x^*) = \sum_i l_i(x_i, x_i^*), \tag{3.10}$$

where the pixel loss l_i is taken to be $l_i(x_i, x_i^*) = \min(|x_i - x_i^*|, \tau)$, appropriate in stereo reconstruction. The empirical risk is the sum of losses (3.10) over a sample of several image pairs, however for our purpose it is sufficient to consider only a single image pair. When the inference is performed by the CRF *i.e.*, the disparity estimate x is the minimizer of (3.3), training the optimal parameters $\theta = (\theta_1, \theta_2, P_1, P_2)$ can be formulated in the form of a *bilevel optimization*:

$$\min_{\theta} l(x, x^*) \tag{3.11a}$$

$$\text{s.t. } x \in \arg\min_{x \in \mathcal{X}} f(x; \theta). \tag{3.11b}$$

Observe that any $x \in \arg\min f(x)$ in (3.11b) necessarily satisfies $f(x) \leq f(x^*)$. Therefore, for any $\gamma > 0$, the scaled loss $\gamma l(x, x^*)$ can be upper-bounded by

$$\max_{x:\ f(x) \leq f(x^*)} \gamma l(x, x^*) \tag{3.12a}$$

$$\leq \max_{x:\ f(x) \leq f(x^*)} [f(x^*) - f(x) + \gamma l(x, x^*)] \tag{3.12b}$$

$$\leq \max_{x} [f(x^*) - f(x) + \gamma l(x, x^*)]\,. \tag{3.12c}$$

A subgradient of (3.12c) w.r.t. $(f_i \mid i \in \mathcal{V})$ can be chosen as

$$\delta(x^*) - \delta(\bar{x}), \tag{3.13}$$

where $\delta(x)_i$ is a vector in $\mathbb{R}^{\mathcal{L}}$ with components $(\llbracket x_i = k \rrbracket \mid k \in \mathcal{L})$, *i.e.* the 1-hot encoding of x_i, and $\bar{x}$ is a (generally non-unique) solution to the *loss augmented inference* problem

$$\bar{x} \in \arg\min_{x} \left[\bar{f}(x) := f(x) - \gamma l(x, x^*)\right]. \tag{3.14}$$

Figure 3.3 shows a visualization of the gradient derived in (3.13). In the case of an additive loss

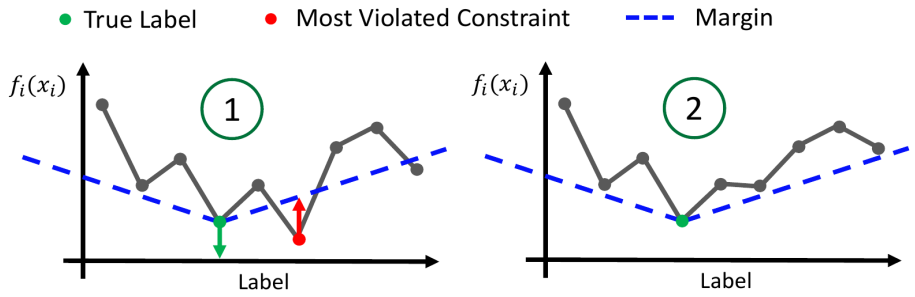


Figure 3.3: Visualization of the gradient resulting from the SSVM. (1) We need to find the most violated constraint $\bar{x}$ by solving the loss-augmented inference problem (3.14). Applying a gradient step is equivalent to decreasing the energy of the ground-truth label x^* and increasing the energy of the most violated constraint.

function, problem (3.14) is of the same type as (3.3) with adjusted unary terms.

We facilitate the intuition of why the SSVM chooses the most violated constraint by rewriting the hinge loss (3.12c) in the form

$$\min\{\xi \in \mathbb{R} \mid (\forall x) \;\; \xi \geq f(x^*) - f(x) + \gamma l(x, x^*)\}, \tag{3.15}$$

which reveals the large margin separation property: the constraint in (3.15) tries to ensure that the training solution x^* is better than all other solutions by a margin $\gamma l(x, x^*)$ and the most violated constraint sets the value of slack ξ. The parameter γ thus controls the margin: a large margin may be beneficial for better generalization with limited data. Finding the most violated constraint in (3.15) is exactly the loss-augmented problem (3.14).

SSVM with Relaxed Inference An obstacle in the above approach is that we cannot solve the loss-augmented inference (3.14) exactly. However, having a method solving its convex relaxation, we can integrate it as follows. Applying the decomposition approach to (3.14) yields a lower bound on the minimization: (3.14) $\geq$

$$\bar{D}(\lambda) := \min_{x^1}(\bar{f}^1 + \lambda)(x^1) + \min_{x^2}(\bar{f}^2 - \lambda)(x^2) \tag{3.16}$$

for all λ. Lower bounding (3.14) like this results in an upper-bound of the loss $\gamma l(x, x^*)$ and the hinge loss (3.12a):

$$\gamma l(x, x^*) \leq (3.12a) \leq f(x^*) - \bar{D}(\lambda). \tag{3.17}$$

The bound is valid for any λ and is tightened by maximizing $D(\lambda)$ in λ. The learning problem on the other hand minimizes the loss in θ. Tightening the bound in λ and minimizing the loss in θ

can be written as a joint problem

$$\min_{\theta,\lambda} f(x^*;\theta) - \bar{D}(\lambda;\theta). \tag{3.18}$$

Using this formulation we do not need to find an optimal λ at once; it is sufficient to make a step towards minimizing it. This approach is known as blended learning and inference [29, 97]. It is disadvantageous for our purpose for two reasons: i) at the test time we are going to use a fixed number of iterations instead of optimal λ ii) joint optimization in θ and λ in this fashion will be slower and iii) it is not feasible to store intermediate λ for each image in the training set as λ has the size of a unary cost volume.

Approximate Subgradient We are interested in a subgradient of (3.17) after a fixed number of iterations of the inference method, *i.e.*, training the unrolled inference. A suboptimal λ (after a fixed number of iterations) will generally vary when the CNN parameters θ and thus the CRF costs f are varied. While we do not fully backtrack a subgradient of λ (which would involve backtracking dynamic programming and recursive subdivision in `Dual_MM`) we can still inspect its structure and relate the subgradient of the approximate inference to that of the exact inference.

Proposition 1. *Let $\bar{x}^1$ and $\bar{x}^2$ be minimizers of horizontal and vertical chain subproblems in* (3.16) *for a given λ. Let $\Omega_{\neq}$ be a subset of nodes for which $\bar{x}^1_i \neq \bar{x}^2_i$. Then a subgradient g of the loss upper bound* (3.17) *w.r.t. $f_{\mathcal{V}} = (f_i \mid i \in \mathcal{V})$ has the following expression in components*

$$\begin{aligned} g_i(k) = &\big(\delta(x^*) - \delta(\bar{x}^1)\big)_i(k) \\ &+ \sum_{j \in \Omega_{\neq}} \big(J_{ij}(k, \bar{x}^2_i) - J_{ij}(k, \bar{x}^1_i)\big), \end{aligned} \tag{3.19}$$

where $J_{ij}(k,l)$ is a sub-Jacobian (matching $\frac{d\lambda_j(l)}{df_i(k)}$ for a subset of directions $df_i(k)$).

Proof. The loss upper bound (3.17) involves the minimum over x^1, x^2 as well as many minima inside the dynamic programming defining λ. A subgradient can be obtained by fixing particular minimizers in all these steps and evaluating the gradient of the resulting function. It follows that a subgradient of the point-wise minimum of $(\bar{f}^1 + \lambda)(x^1) + (\bar{f}^2 - \lambda)(x^2)$ over x^1, x^2 can be chosen as $g =$

$$\nabla_{f_{\mathcal{V}}}(\bar{f}^1(\bar{x}^1) + \bar{f}^2(\bar{x}^2)) + \nabla_{\lambda}(\lambda(\bar{x}^1) - \lambda(\bar{x}^2))J, \tag{3.20}$$

where $J_{i,j}(k,l)$ is a sub-Jacobian matching $\frac{d\lambda_j(l)}{df_i(k)}$ for the directions $df_{\mathcal{V}}$ such that $\lambda(f + df_{\mathcal{V}})$ has the same minimizers inside dynamic programming as $\lambda(f)$.

In the first part of the expression (3.20), the pairwise components and the loss $l(\bar{x}^1, x^*)$ do not depend on f_i and may be dropped, leaving only $(\nabla_{f_{\mathcal{V}}} \sum_{j \in \mathcal{V}} f_j(\bar{x}^1_j))_i = \delta(\bar{x}^1)_i$.

Let h denote the second expression in (3.20). Its component $h_i(k)$ expands as

$$h_i(k) = \sum_{j\in\mathcal{V}} \sum_{l\in\mathcal{L}} \frac{\partial}{\partial\lambda_j(l)} (\lambda_j(\bar{x}_j^1) - \lambda_j(\bar{x}_j^2)) J_{ij}(k,l) \tag{3.21a}$$

$$= \sum_{j\in\Omega_{\neq}} \sum_{l\in\mathcal{L}} ([\![\bar{x}_j^1{=}l]\!] - [\![\bar{x}_j^2 = l]\!]) J_{ij}(k,l) \tag{3.21b}$$

$$= \sum_{j\in\Omega_{\neq}} (J_{ij}(k,x_j^1) - J_{ij}(k,x_j^2)). \tag{3.21c}$$

□

Our intuition to neglect the sum (3.21c) is as follows. We expect that variation of f_i for a pixel i far enough from $j \in \Omega_{\neq}$ will not have a significant effect on λ_j and thus J_{ij} will be small over $\Omega_{\neq}$.

We conjecture that when the set $\Omega_{\neq}$ is small, for many nodes the contribution of the sum in (3.19) will be also small, while the first part in (3.19) matches the subgradient with exact inference (3.13).

Proposition 2. *For training the abbreviate inference with dual decomposition such as* `Dual_MM`*, we calculate the minimizer* $\bar{x}^1$ *after a fixed number of iterations and approximate the subgradient as* $\delta(x^*) - \delta(\bar{x}^1)$.

The assumption for the learning to succeed is to eventually have most of the pixels in agreement. The inference method works towards this by adjusting λ such that the constraints $x_i^1 = x_i^2$ are satisfied. We may expect in practice that if the data is not too ambiguous this constraint will be met for a large number of pixels already after a fixed number of iterations. A good initialization of unary costs, such as those learned using the pixel-wise only method can help to improve the initial agreement and to stabilize the method.

3.4.3 Training Unary and Pairwise CNNs in Joint Model

To make the pairwise interactions trainable, we need to compute a subgradient w.r.t. w_{ij}, P_1, P_2. We will compute it similarly to the unary terms assuming exact inference, and then just replace the exact minimizer $\bar{x}$ with an approximate $\bar{x}^1$. A subgradient of (3.12c) is obtained by choosing a minimizer $\bar{x}$ and evaluating the gradient of the minimized expression. Components of the later are given by

$$\tfrac{\partial}{\partial w_{ij}} = \rho(|x_i^*{-}x_j^*|; P_{1,2}) - \rho(|\bar{x}_i - \bar{x}_j|; P_{1,2}), \tag{3.22a}$$

$$\tfrac{\partial}{\partial P_1} = \textstyle\sum_{ij} w_{ij} ([\![|x_i^*{-}x_j^*| = 1]\!] - [\![|\bar{x}_i{-}\bar{x}_j| = 1]\!]), \tag{3.22b}$$

$$\tfrac{\partial}{\partial P_2} = \textstyle\sum_{ij} w_{ij} ([\![|x_i^*{-}x_j^*| > 1]\!] - [\![|\bar{x}_i{-}\bar{x}_j| > 1]\!]). \tag{3.22c}$$

We thus obtain an end-to-end trainable model without any hand-crafted parameters, except for the hyper-parameters controlling the training itself.

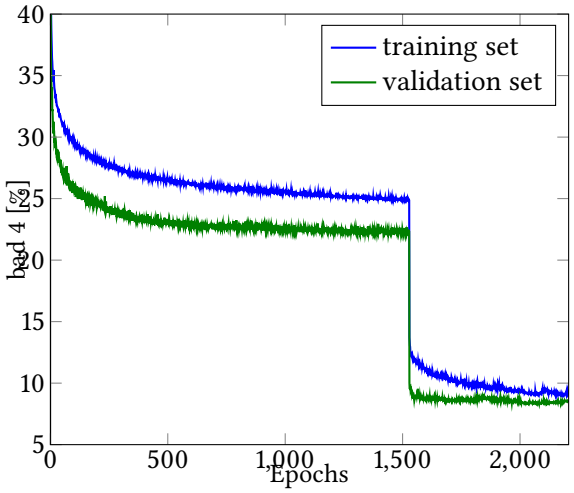


Figure 3.4: Performance w.r.t. the real objective for key complexity steps of our model during training.

3.4.4 Implementation Details

We trained our models using Theano [10] with stochastic gradient descent and momentum. For training the model without pairwise costs we set the learning rate to $1{\times}10^{-2}$, for all other models we set the learning rate to $1{\times}10^{-6}$. Before feeding a sample into our model we normalize it such that it has zero-mean and unit-variance. We additionally correct the rectification for Middlebury samples. Our full model is trained gradually. We start by training the models with lower complexity and continue by training more complex models, where we reuse previously trained parameters and initialize new parameters randomly. Since we use full RGB images for training, we have to take care of occlusions as well as invalid pixels, which we mask out during training. Additionally, we implemented the forward pass using C++/CUDA in order to make use of our trained models in a real-time environment in a streaming setting. We achieve 3-4 frames per second with our fully trained 3-layer model using an input-size of 640×480

3.4.5 Training insights

We train our full joint model gradually as explained in Section 3.4 in the main paper. To give more insights on how the joint training evolves until we get our final parameters, we show a training plot in Fig. 3.4. This plot shows the evolution of the *bad4* error on the Middlebury dataset for our 7-layer model. We can identify three key steps during the training procedure. (A) shows the training of our *Unary-CNN* using ML Section 3.4.1. In (B) we add the CRF with contrast-sensitive weights with an optimal choice of parameters $(\alpha, \beta, P_1, P_2)$. Finally, in (C) we jointly optimization the complete model Sections 3.4.2 and 3.4.3. Observe that the gap between training and validation errors is significantly smaller in (C).

3.5 Experiments

In this section we test different variants of our proposed method. In order not to confuse the reader, we use the following naming convention: *CNNx* is the argmax output of a network trained as described in Section 3.4.1; *CNNx+CRF* is the same network with `Dual_MM` as post-processing; *CNNx+CRF+Joint* is the jointly trained network described in Section 3.4.2 and *CNNx+CRF+Joint+PW* is the fully trained method described in Section 3.4.3. x represents the number of layers in the CNN.

3.5.1 Benchmark Data Sets

We use two stereo benchmark datasets for our experiments: Kitti 2015 [132] and Middlebury V3 [169]. Both benchmarks hold out the **test** set, where the ground truth is not accessible to authors. We call examples with ground truth available that can be used for training/validation the **design** set and split it randomly into 80% **training** set and 20% **validation** set. This way we obtain 160 + 40 examples for Kitti and 122 + 31 examples for Middlebury (including additionally provided images with different lightings, exposures and perfectly/imperfectly rectified stereo-pairs). The used error metric in all experiments is the percent of pixels with a disparity difference above x pixels (*badx*).

3.5.2 Performance of Individual Components

In this experiment we measure the performance improvement when going from *CNNx* to the full jointly trained model. Since ground-truth of the test data is not available to us, this comparison is conducted on the complete design set. The results are shown in Table 3.2. This experiment demonstrates that an optimization or post-processing is necessary, since the direct output of all tested CNNs (after a simple point-wise minimum search in the cost volume) contains too many outliers to be used directly. A qualitative comparison on one of the training images of Middlebury is depicted in Table 3.1. One can observe that the quality of the CNN-only method largely depends on the number of layers, whereas the CNN+CRF versions achieve good results even for a shallow CNN. Table 3.3 additionally shows the error metrics *bad*$\{2,3,4\}$ on the design set of Kitti, because these error metrics cannot be found online.

3.5.3 Benefits of Joint Training

In this experiment, we compare our method to two recently proposed stereo matching methods based on CNNs, the *MC-CNN* by Zbontar and LeCun [220] and the *Content-CNN* by Luo et al. [119]. To allow a fair comparison of the methods, we disable all engineered post-processing steps of [119, 220]. We then unify the post-processing step by adding our CRF on top of the CNN outputs. We evaluate on the whole design set since we do not know the train/test split of the different methods. In favor of the compared methods, we individually tune the parameters P_1, P_2, α, β of the CRF for each method using grid search. The results are shown in Table 3.2. While the raw output

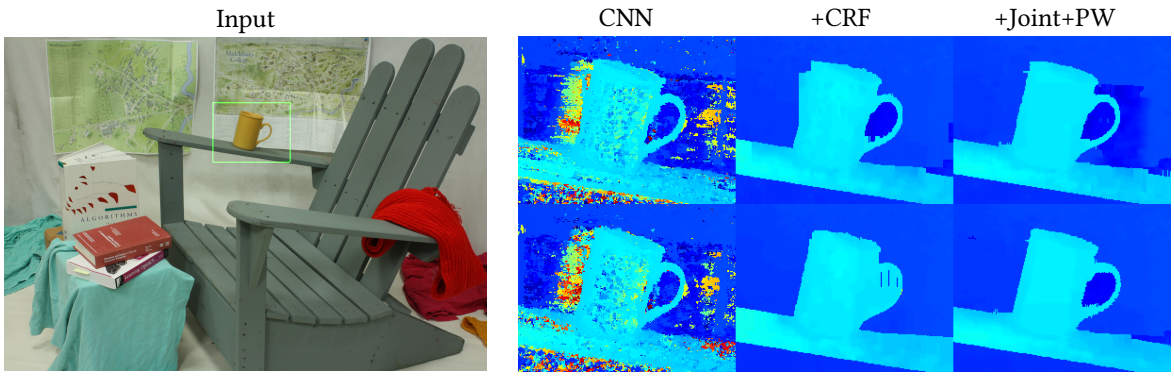


Table 3.1: Qualitative comparison of *Unary-CNN*, CNN+CRF and CNN+CRF+Joint+PW on the Middlebury benchmark. Zoom-in of disparity with 3 layers (top) and 7 layers (bottom). Note how the jointly trained models inpaint occlusions correctly.

Benchmark	**Method**	**CNN**	**+CRF**	**+Joint**	**+PW**
Middlebury	CNN3	23.89	11.18	9.48	9.45
	CNN7	18.58	9.35	8.05	7.88
Kitti 2015	CNN3	28.38	6.33	6.11	4.75
	CNN7	13.08	4.79	4.60	4.04
	[119]	5.99	4.31	-	-
	[220]	13.56	4.45	-	-

Table 3.2: Influence of the individual components of our method (Section 3.5.2) and comparison with [119, 220] without post-processing (Section 3.5.3). Standard error metrics (*bad4* on official training data for Middlebury and *bad3* on the **design** set for Kitti) are reported.

Method	**Non-occ**	**All**	**Time**
[126]	4.32	4.34	**0.06**s
[119]	4.00	4.54	1s
[220] acc.	3.33	3.89	67s
[174]	**2.58**	**3.61**	68s
Ours	4.84	5.50	1.3s

(a) Performance on the Kitti 2015 test set. We report the standard error *bad3* on both non-occluded and all pixels.

Train err.	**bad2**	**bad3**	**bad4**
[119]^3	7.39	4.31	3.14
[220]^3	11.4	4.45	**2.93**
Ours	**6.01**	**4.04**	3.15

(b) Comparison of the training error with different error metrics *badx* on the Kitti dataset.

of our CNN is inferior to the compared methods, the post-processing with a CRF significantly decreases the difference in performance. Joint training of our CNN+CRF model further improves the performance, despite using a relatively shallow network with fewer parameters. Specifically, our full joint model with 7 layers has 281k parameters, while the networks [119, 220] have about 700k and 830k parameters, respectively.

Method	Average performance	Time [sec]	Australia	AustraliaP	Bicycle2	Classroom2	Classroom2E	Computer	Crusade	CrusadeP	Djembe	DjembeL	Hoops	Livingroom	Newkuba	Plants	Staircase	Metric
Middlebury																		
[220] fst	22.4	**1.69**	22.0	20.3	12.7	28.8	42.6	9.82	28.7	25.1	5.07	32.0	23.3	16.5	30.6	25.5	34.1	
[220] acc.	21.3	150	20.8	19.6	9.6	28.6	67.4	7.67	23.2	15.7	8.49	31.8	**16.7**	13.9	38.8	18.7	28.6	
[4]	15.0	188	18.4	18.1	**8.72**	**9.06**	19.9	**6.52**	24.2	25.7	**3.91**	**12.7**	24.7	**9.58**	**17.9**	**17.5**	**17.9**	RMS
Ours	**14.4**	4.46	**15.9**	**16.2**	10.7	10.3	**11.2**	14.0	**13.7**	**13.1**	4.11	14.3	19.2	11.9	22.5	20.6	25.5	
[220] fst	9.47	**1.69**	7.35	5.07	7.18	4.71	16.8	8.47	**7.37**	**6.97**	2.82	20.7	17.4	15.4	15.1	7.9	12.6	
[220] acc.	**8.29**	150	5.59	4.55	**5.96**	**2.83**	11.4	**8.44**	8.32	8.89	**2.71**	16.3	**14.1**	13.2	**13.0**	**6.40**	11.1	
[4]	8.62	188	6.05	5.16	6.24	3.27	**11.1**	8.91	8.87	9.83	3.21	**15.1**	15.9	**12.8**	13.5	7.04	**9.99**	bad2
Ours	12.5	4.46	**4.09**	**3.97**	8.44	6.93	**11.1**	13.8	19.5	19.0	3.66	17.0	18.2	18.0	21.0	7.29	17.8	

Table 3.3: Performance on the Middlebury **test** set as of time of submission. We compare our results against work that is based on CNNs for matching costs and accepted for publication. We report the respective standard error metric *bad2* and the root-mean-squared error.

3.5.4 Benchmark Test Performance

The complete evaluation of our submission on test images is available in the online suites of Middlebury [169] and Kitti 2015 [132]. The summary of this evaluation is presented in Table 3.3. We want to stress that these results have been achieved without using any post-processing like occlusion detection and -inpainting or sub-pixel refinement.

We fine-tuned our best performing model (Table 3.2, CNN7+PW) for *half* sized images and used it for the Middlebury evaluation. Table 3.3 shows the root mean squared (RMS) error metric and the *bad2* error metric for all test images. We achieve the lowest overall RMS error. Our *bad2* error is slightly worse compared to the other methods. These two results suggest our wrong counted disparities are just slightly beside. This behavior is shown in the error plot at the bottom in Fig. 3.6, where many small discretization artefacts are visible on slanted surfaces. Note that a sub-pixel refinement would remove most of this error. Additionally, we present an example where our algorithm achieves a very low error as in the majority of images.

For Kitti we use our best performing model (Table 3.2, CNN7+PW), including the x- and y-coordinates of the pixels as features. This is justified because the sky is always at the top of the image while the roads are always at the bottom for example. The error plots for Kitti in Fig. 3.7 reveal that most of the incorrect predictions are in occluded areas. In Fig. 3.8 we show a qualitative comparison of magnified depth predictions of CNN-based methods on a Kitti test image. The depth overlays at the left side of the figure show how accurately the algorithms recover object boundaries and the images on the right side show the corresponding error plots provided by the evaluation system. Note, that very accurate predictions are partially treated as incorrect and how

[3] With our CRF as postprocessing

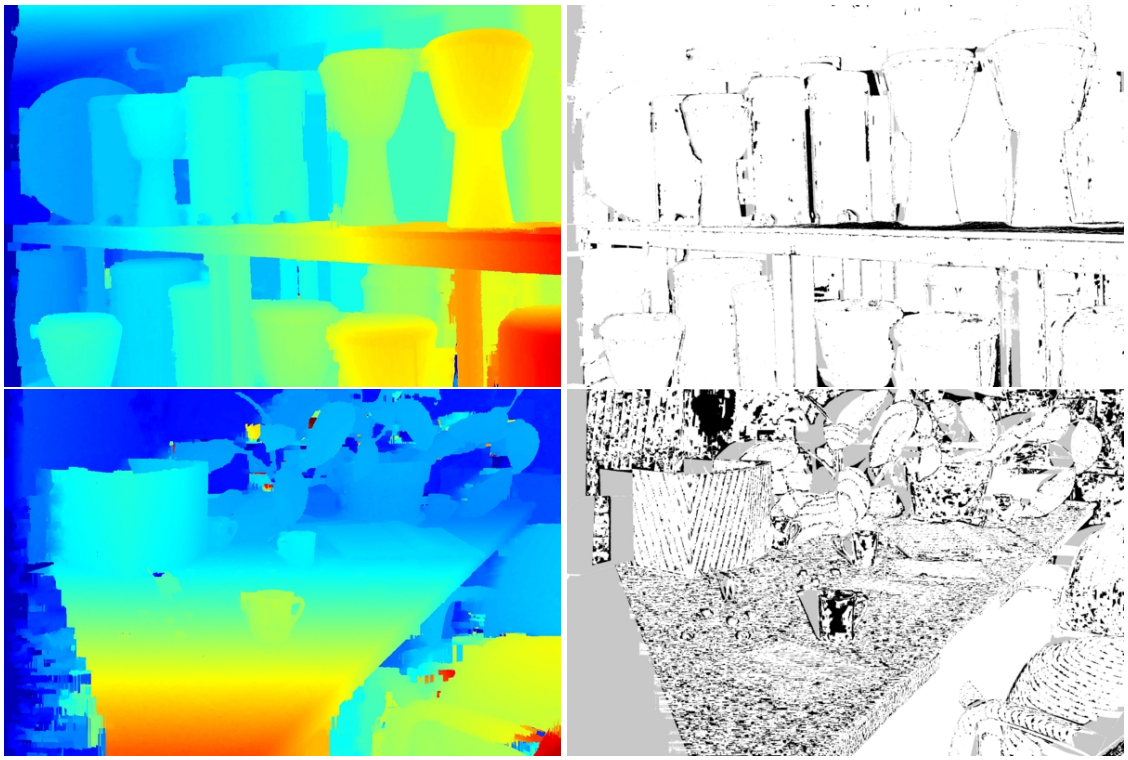


Figure 3.6: Qualitative comparison on selected *test* images (from top to bottom: *Djembe* and *Crusade*) of the Middlebury Stereo Benchmark. The left column shows the generated disparity images in false color, the right column the bad2 error image, where white = error smaller than 2 disparities, grey = occlusion and black = error greater than 2 disparities.

the competing methods tend to overfit to the fattened ground truth. Our approach works also very well in the upper third of the images, whereas the competing methods bleed out.

3.5.5 Additional Experiments

3.5.5.1 Sublabel Enhancement

A drawback of our CRF method based on dynamic programming is the discrete nature of the solution. For some benchmarks like Middlebury the discretization artifacts negatively influence the quantitative performance. Therefore, most related stereo methods perform some kind of sub-label refinement (*e.g.* [119, 220]). For the submission to online benchmarks we deliberately chose to *discard* any form of non-trainable post-processing. However, we performed additional experiments with fitting a quadratic function to the output cost volume of the CRF method around the discrete solution. The refined disparity is then given by

$$d_{se} = d + \frac{C(d-h) - C(d+h)}{2(C(d+h) - 2C(d) + C(d-h))} \tag{3.23}$$

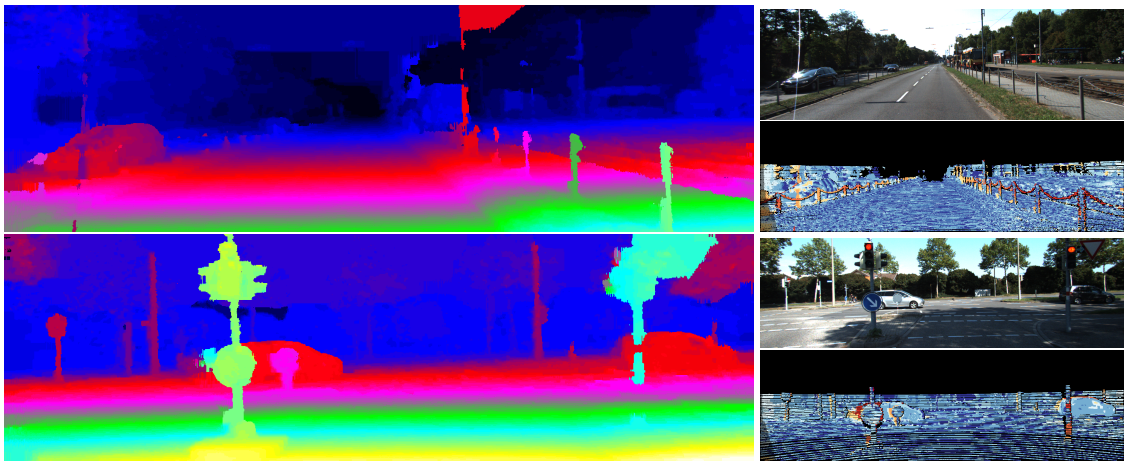
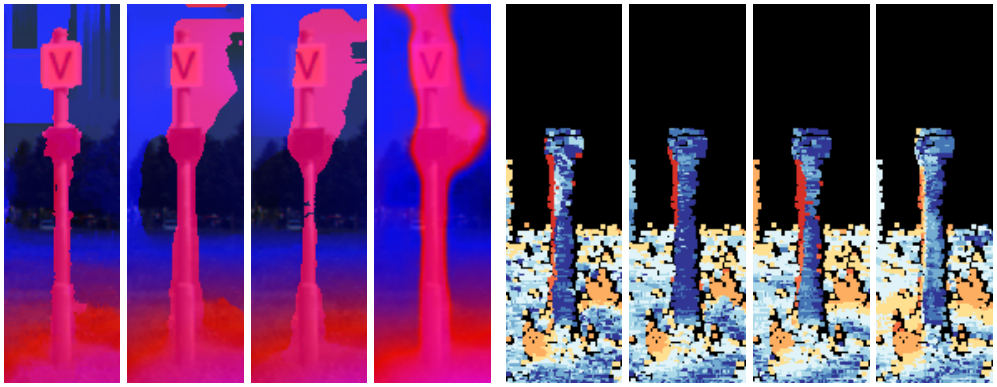


Figure 3.7: Qualitative comparison on the test set of Kitti 2015. Cold colors = error smaller than 3 disparities, warm colors = error larger than 3 disparities.



Ours [220] [126] [119] Ours [220] [126] [119]

Figure 3.8: Zoom-in comparison with state-of-the-art methods on a selected test image. Left images show an overlay of depth prediction and input image and right images show the corresponding error plots.

where $C(d)$ is the cost of disparity d. A qualitative experiment on the *Motorcycle* image of Middlebury stereo can be seen in Fig. 3.9. Quantitative experiments have been conducted on both Kitti 2015 and Middlebury and will be reported in the follow sections (columns **w. ref.** in Figs. 3.11 and 3.11b). Again, in the main paper and in the submitted images we always report the performance of the *discrete* solution in order to keep the method pure.

3.5.5.2 Middlebury Stereo v3

In this section we report a complete overview of all tested variants of our proposed hybrid CNN-CRF model on the stereo benchmark of Middlebury Stereo v3. We report the mean error (error metric *percent of non-occluded pixels with an error bigger 4 pixels*). All results are calculated on

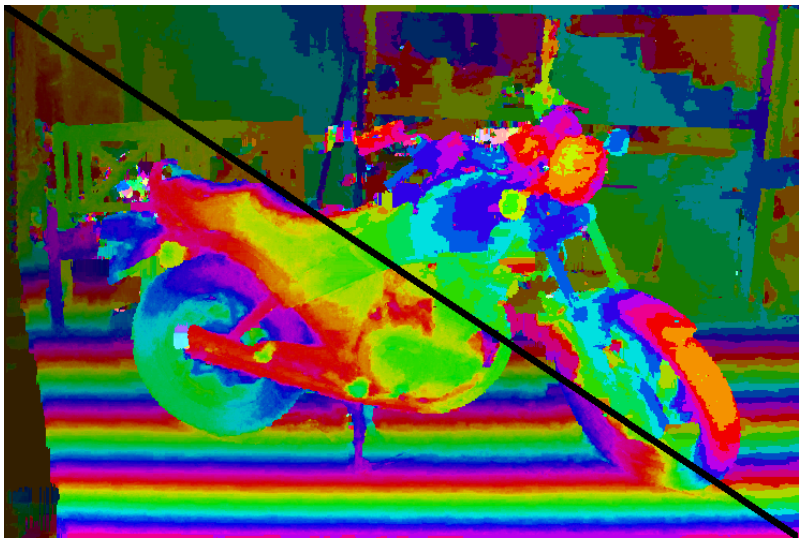


Figure 3.9: Qualitative comparison on *Motorcycle* of discrete (upper-right) and sublabel enhanced (bottom-left) solution. Note how smooth the transitions are in the sublabel enhanced region (*e.g.* at the floor or the rear wheel).

quarter resolution and upsampled to the original image size. We present the results in Figs. 3.10 and 3.11. Note, how the quality increases when we add more parameters and therefore allow a more general model (visualized from left to right in Fig. 3.10. The last row shows the *Vintage* image, where our model produces a rather high error. The reason for that lies in the (almost) completely untextured region in the top-left corner. Our full model is able to recover some disparities in this region, but not all. A very interesting byproduct visible in Fig. 3.10 concerns our small 3-layer model. Visually, one can hardly see any difference to the deeper 7-layer model, when our models are full jointly trained. Hence, this small model is suited very well for a real-time application.

Additionally, we compared to the performance of the model learned on Kitti, denoted Kitti-CNN in Fig. 3.11. The performance is inferior, which means that the model trained on Kitti does not generalize well to Middlebury. Generalizing from Middlebury to Kitti, on the other hand is much better, as discussed in the next section.

3.5.5.3 Kitti 2015

In this section we report a complete overview of all tested variants of our proposed hybrid CNN-CRF model on the stereo benchmark of KITTI 2015. We report the mean error (official error metric *percent of pixel with an error bigger 3 pixels*) on the complete design set. Fig. 3.11b shows a performance overview of our models. In the last row of Fig. 3.11b we apply our best performing model on Middlebury to the Kitti design set. Interestingly, the performance decreases only by $\approx 1.5\%$ on all pixels. This experiment indicates, that our models generalize well to the scenes of the Kitti benchmark.

Due to lack of space in the main paper, we could only show a few qualitative results of the

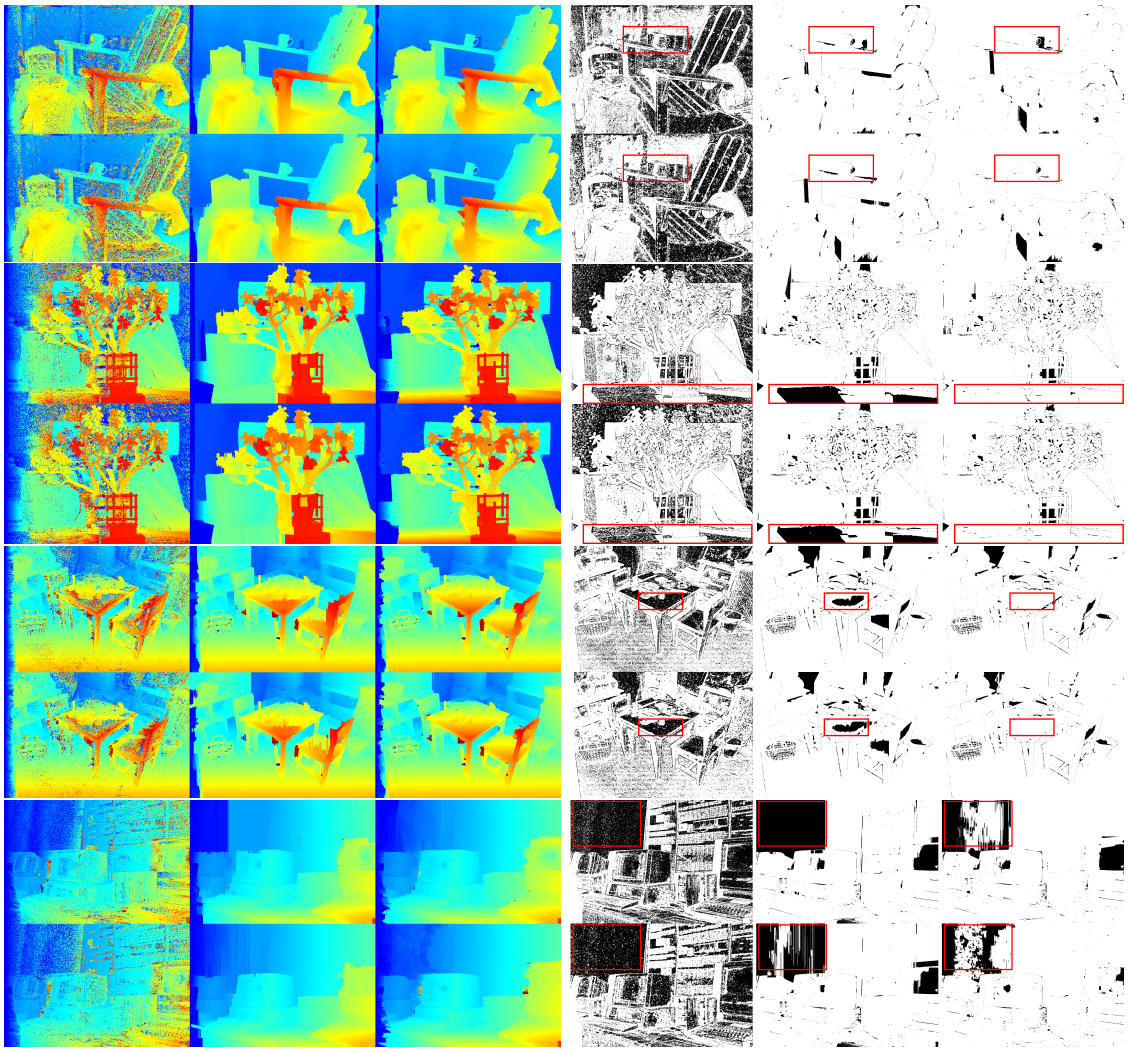


Figure 3.10: Qualitative comparison of our models on Middlebury. For each image, the first row shows our 3-layer model and the second row shows the result of our 7-layer model. The first column shows out *Unary-CNN* with argmax desicion rule, the second column *CNNx+CRF* and the third column shows the result of *CNNx+CRF+Joint+PW*. The remaining columns show the respective error-plots for the different models, where white indicates correct and black indicates wrong disparities. The red boxes highlight differences between our models. Disparity maps are color-coded from blue (small disparities) to red (large disparities).

submitted method. In Fig. 3.12 we show additional results, more of which can be viewed online.

Looking at Kitti results in more detail, we observe that most of the errors happen in either occluded regions or due to a fattened ground-truth. Since we train edge-weights to courage label-jumps at strong object boundaries, our model yields very sharp results. It is these sharp edges in our solution which introduce some errors on the benchmark, even when our prediction is correct. Fig. 3.13 shows some examples on the *test* set (provided by the online submission system).

Method	w/o. ref.	w. ref.
CNN3	23.89	-
CNN3+CRF	11.18	10.50
CNN3 Joint	9.48	8.75
CNN3 PW+Joint	9.45	8.70
CNN7	18.58	-
CNN7+CRF	9.35	8.68
CNN7 Joint	8.05	7.32
CNN7 PW+Joint	7.88	7.09
Kitti-CNN	15.22	14.43

(a) Middlebury 2014

Method	w/o. ref.		w. ref.	
	all	non occ.	all	non occ.
CNN3	29.58	28.38	-	-
CNN3+CRF	7.88	6.33	7.77	6.22
CNN3 Joint	7.66	6.11	7.57	6.02
CNN3 PW+Joint	6.25	4.75	6.14	4.65
CNN7	14.55	13.08	-	-
CNN7+CRF	5.85	4.79	5.76	4.70
CNN7 Joint	5.98	4.60	5.89	4.50
CNN7 PW+Joint	5.25	4.04	5.18	3.96
[220]+CRF	6.10	4.45	5.74	4.08
[119]+CRF	5.89	4.31	5.81	4.21
[220]	15.02	13.56	-	-
[119]	7.54	5.99	-	-
MB-CNN	6.82	5.35	6.69	5.21

(b) KITTI 2015

Figure 3.11: Comparison of differently trained models and their performance on the design set of Kitti and on the official training images of the Middlebury V3 stereo benchmark. The results are given in % of pixels farther away than 4 disparities from the ground-truth on all pixels.

3.5.6 Timing

In Table 3.4 we report the runtime of individual components of our method for different image sizes and number of labels (=disparties). All experiments are carried out on a Linux PC with a Intel Core i7-5820K CPU with 3.30GHz and a NVidia GTX TitanX using CUDA 8.0. For Kitti 2015, the image size is 1242×375. For Middlebury V3 we selected the *Jadeplant* data set with *half* resolution, leading to an image size of 1318×994. We observe that with a constant number of layers in the Unary CNN and disparity range, the runtime depends linearly on the number of pixels in the input images. Correlation and CRF layer also depend on the number of estimated disparities, where we report numbers using 128 and 256 disparities.

3.6 Conclusion

We have proposed a fully trainable hybrid CNN+CRF model for stereo and its joint training procedure. Instead of relying on various post-processing procedures we designed a clean model without post-processing, where each part has its own responsibility. Therefore we gain interpretability of what is learned in each component of the model. This gives the insight that using a well defined model decreases the number of parameters significantly while still achieving a competitive performance. We have shown that the joint training allows to learn unary costs as well as pairwise costs, while having the evidence that the increased generality always improves the

Component	**# Disp.**	**Kitti 2015** 0.4 MP	**Middlebury** 1.3 MP	**Real-Time** 0.3 MP
Input processing		7.58	6.40	6.02
Pairwise CNN		21.12	59.46	13.75
Unary CNN		262.48	664.19	62.54
Correlation	128	154.86	437.02	46.70
Correlation	256	286.87	802.86	–
CRF	128	309.48	883.57	155.85
CRF	256	605.35	1739.34	–
Total	128	755.52	2050.64	284.86
Total	256	1183.40	3272.25	–

Table 3.4: Timing experiments for 7 layer CNN and 5 CRF iterations (3 layer and 4 iterations for **Real-Time**). Runtimes in ms.

performance. Our newly proposed trainable pairwise terms allow to delineate object boundaries more accurately. For the SSVM training we detailed the approximation of a subgradient and have shown that our training procedure works experimentally. For future work we plan to introduce an additional occlusion label to our model to further improve the performance in occluded areas. In addition, it will be interesting to investigate a continuous label space [134] to improve the performance of the model on slanted surfaces.

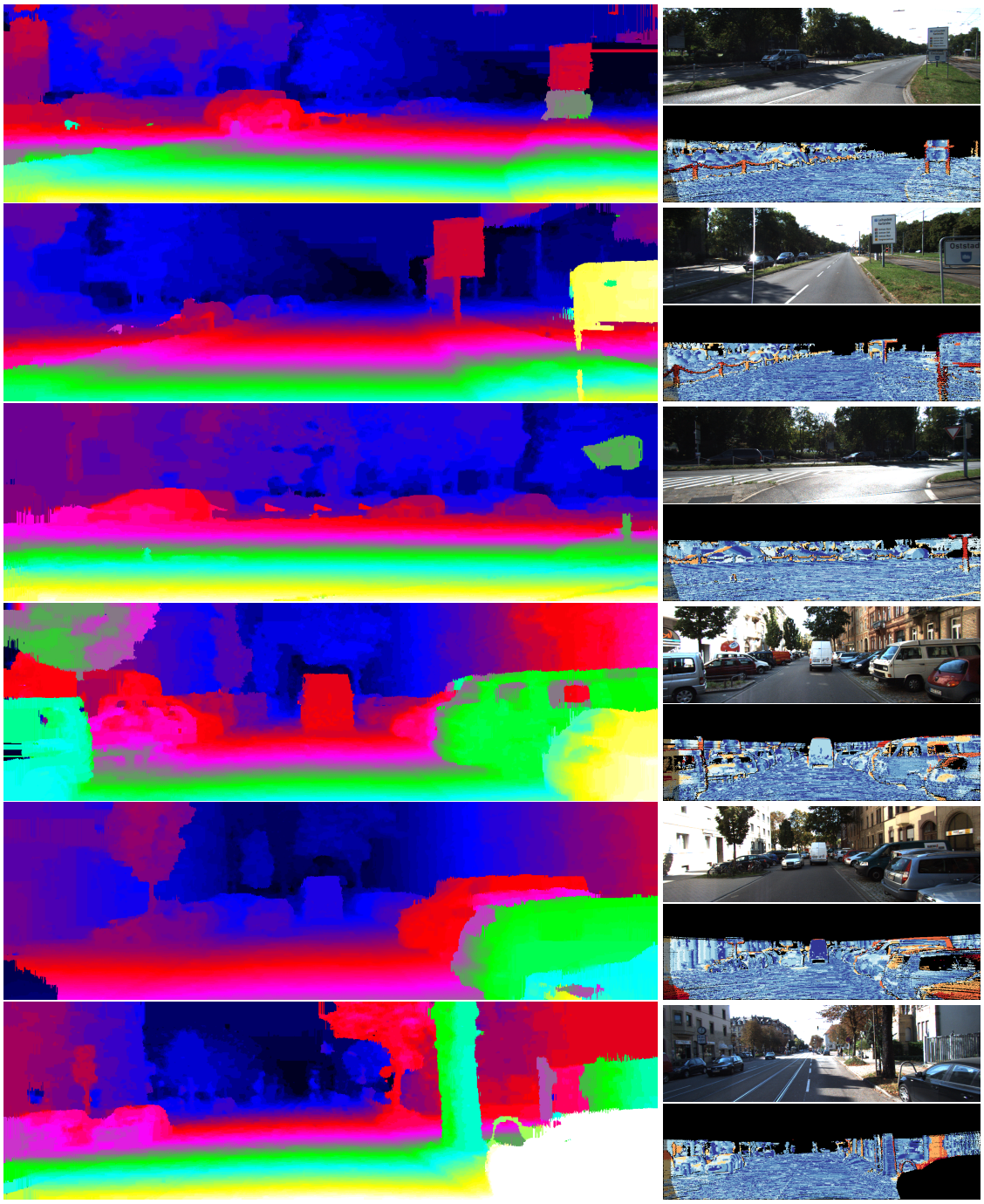


Figure 3.12: Qualitative comparison on the test set of KITTI 2015.

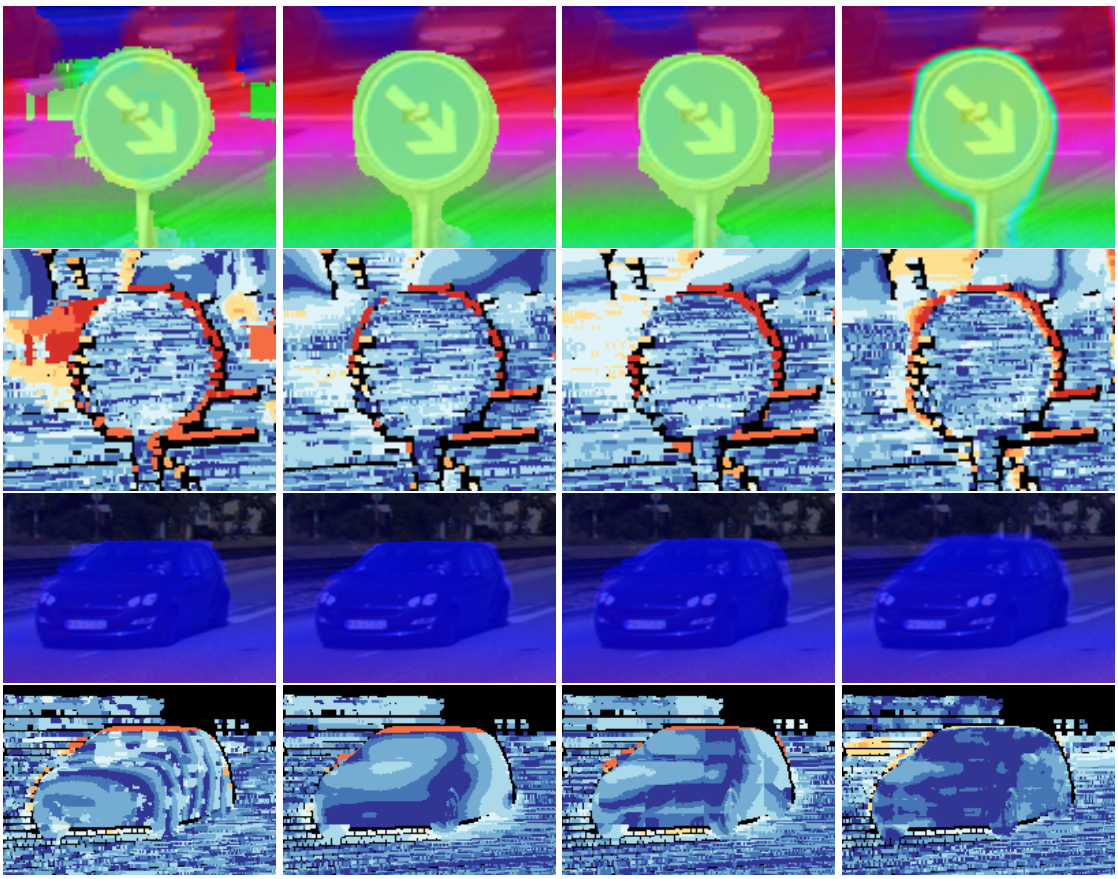


Figure 3.13: Error comparison on magnified parts of Kitti 2015 test images: The first and third row show the color-coded disparity map of *Ours*, *MC-CNN*, *ContentCNN* and *DispNetC*. The second and last row show the corresponding error-plots, where shades of blue mean correct and shades of orange mean wrong. Note, how our model accurately follows object boundaries, whereas all other approaches fatten the object. Nevertheless, in terms of correct or wrong we make more wrong predictions, because the ground-truth seems to be fattened as well.

4

Belief Propagation Reloaded

It has been shown by several works that the combination of Convolutional Neural Network (CNN) and Conditional Random Field (CRF) is often beneficial in Deep Learning (DL). The hybrid setting yields much smaller models which are able to generalize better to previously unseen data while maintaining the performance of CNN-only counterparts. However, efficient training of these hybrid models has been an open problem in 2019 when we have been working on the paper presented in this chapter. The method presented in the previous section [87] is still based on an approximate sub-gradient, which makes training more difficult. Mean-field methods [226] are an interesting alternative, but they usually assume complete independence of the nodes in the grid graph during inference, which makes this approach a rough approximation. In this work, we propose a generic framework to compute the exact gradient of numerous CRF inference algorithms efficiently. We are able to maintain the largest possible sub-trees for every node that we can solve exactly with dynamic programming. Our method can be seamlessly integrated into DL frameworks and can be used for a variety of computer vision problems.

This work was presented at CVPR 2020 in Seattle, USA[1].

Contents

4.1 Introduction **106**
4.2 Related Work **107**
4.3 Belief Propagation **108**
4.4 Sweep BP-Layer **110**
4.5 Models **116**
4.6 Learning **118**
4.7 Implementation Details **119**
4.8 Experiments **121**
4.9 Conclusion **133**

[1]The conference was a virtual conference due to the COVID-19 virus.

4.1 Introduction

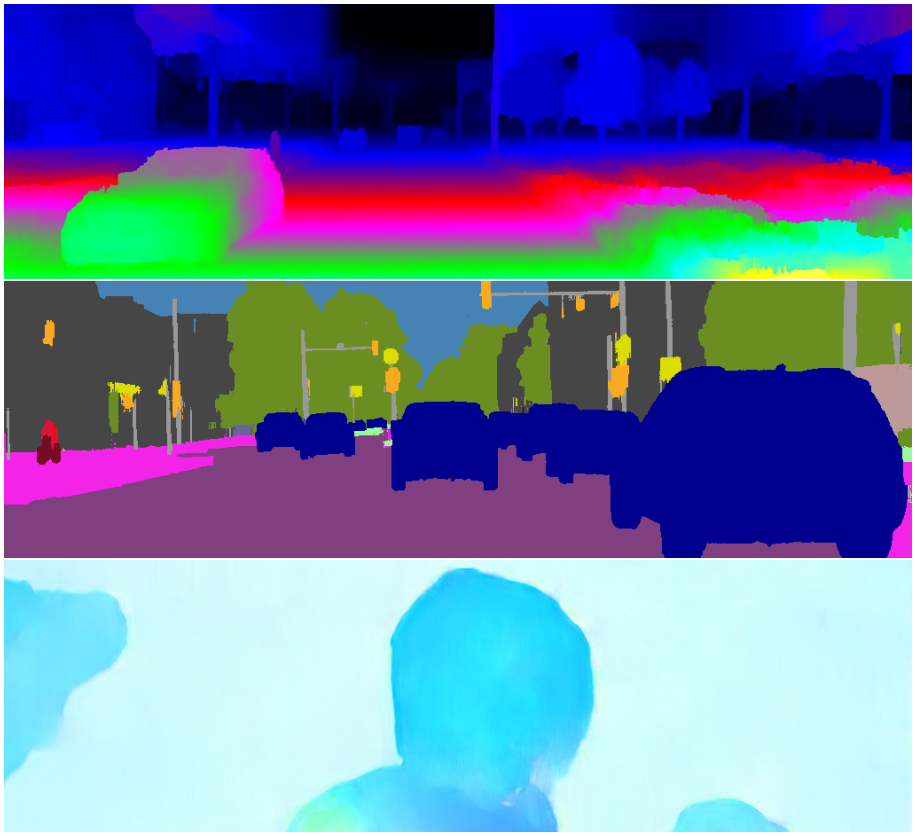


Figure 4.1: BP-Layer in action. The BP-Layer can be used for dense prediction problems such as stereo (top) semantic segmentation (middle) or optical flow (bottom). Note the sharp and precise edges for all three tasks. Input images are from Kitti, Cityscapes and Sintel benchmarks.

We consider dense prediction tasks in computer vision that can be formulated as assigning a categorical or real value to every pixel. Of particular interest are the problems of semantic segmentation, stereo depth reconstruction and optical flow. The importance of these applications is indicated by the active development of new methods and intense competition on common benchmarks.

Convolutional Neural Networks (CNNs) have significantly pushed the limits in dense prediction tasks. However, composing only CNN blocks, though a general solution, becomes inefficient if we want to increase robustness and accuracy: with the increase of the number of blocks and respectively parameters the computational complexity and the training data required grow significantly. The limitations are in particular in handling long-range spatial interactions and structural

constraints, for which Conditional Random Fields (CRFs) are much more suitable. Previous work has shown that a combination of CNN+CRF models can offer an increased performance, but incorporating inference in the stochastic gradient training poses some difficulties.

In this work we consider several simple inference methods for CRFs: A variant of Belief Propagation (BP) [186], tree-structured dynamic programming [15] and semi-global matching [71]. We introduce a general framework, where we view all these methods as specific schedules of max-product BP updates and propose how to use such BP inference as a layer in neural networks fully compatible with deep learning. The layer takes categorical probabilities on the input and produces refined categorical probabilities on the output, associated with marginals of the CRF. This allows for direct training of the truncated inference method by propagating gradients through the layer. The proposed BP-Layer can have an associated loss function on its output probabilities, which we argue to be more practical than other variants of CRF training. Importantly, it can be also used as an inner layer of the network. We propose a multi-resolution model in which BP-Layers are combined in a hierarchical fashion and feature both, associated loss functions as well as dependent further processing blocks.

We demonstrate the effectiveness of our BP-Layer on three dense prediction tasks. The BP-Layer performs a global spatial integration of the information on the pixel-level and is able to accurately preserve object boundaries as highlighted in Fig. 4.1. Deep models with this layer have the following beneficial properties: (i) they contain much fewer parameters, (ii) have a smaller computation cost than the SoTA fully CNN alternatives, (iii) they are better interpretable (for example we can visualize and interpret CRF pairwise interaction costs) and (iv) lead to robust accuracy rates. In particular, in the high-resolution stereo Middlebury benchmark, amongst the models that run in less than 10 seconds, our model achieves the second best accuracy. The CRF for stereo is particularly efficient in handling occlusions, explicitly favoring slanted surfaces and in modelling a variable disparity range. In contrast, many CNN techniques have the disparity range hard-coded in the architecture.

4.2 Related Work

We discuss the related work from the points of view of the learning formulation, gradient computation and application in dense prediction tasks.

CRF Learning CRFs can be learned by the maximum margin approach (*e.g.*, [79, 85]) or the maximum likelihood approach and its variants (*e.g.*, [1, 82, 113, 146]). In the former, the loss depends on the optimal (discrete) solution and is hard to optimize. In the latter, the gradient of the likelihood is expressed via marginals and approximate marginals can be used. However, it must be ensured that during learning enough iterations are performed, close to convergence of the approximation scheme [44], which is prohibitive in large-scale learning settings. Instead, several works advocate truncated inference and a loss function directly formulated on the approximate marginals [44, 45, 76]. This gives a tighter connection between learning and inference, is better corresponding to the empirical loss minimization with the Hamming loss and is easy to apply with

incomplete ground truth labelings. Experimental comparison of multiple learning approaches for CRFs [45] suggest that marginalization-based learning performs better than likelihood-based approximations on difficult problems where the model being fit is approximate in nature. Our framework follows this approach.

Differentiable CRF Inference For learning with losses on marginals Domke [45] introduced Back-Mean Field and Back-TRW algorithms allowing back-propagation in the respective inference methods. Back-Belief Propagation [49] is an efficient method applicable at a fixed point of BP, originally applied in order to improve the quality of inference, and not suitable for truncated inference. While the methods [44, 45, 49] consider the sum-product algorithms and back-propagate their elementary message passing updates, our method back-propagates the sequence of max-product BP updates on a chain at once. Max-product BP is closely related with the Viterbi algorithm and Dynamic Programming (DP). However, DP is primarily concerned with finding the optimal configuration. The smoothing technique [131] addresses differentiating the optimal solution itself and its cost. In difference, we show the back propagation of max-marginals.

The *mean field inference* in fully connected CRFs for semantic segmentation [31, 226] like our method maps label probabilities to label probabilities, is well-trainable and gives improvements in semantic segmentation. However, the model does not capture accurate boundaries [123] and cannot express constraints needed for stereo/flow such as non-symmetric and anisotropic context dependent potentials.

Gaussian CRFs (GCRFs) use quadratic costs, which is restrictive and not robust if the solution is represented by one variable per pixel. If K variables are used per pixel [196], a solution of a linear system of size $K \times K$ is needed per each pairwise update and the propagation range is only proportional to the number of iterations.

Semi-Global Matching (SGM) [71] is a very popular technique adopted by many works on stereo due to its simplicity and effectiveness. However, its training has been limited either to learning only a few global parameters [131] or to indirect training via auxiliary loss functions [175] avoiding backpropagating SGM. Although we focus on a different inference method, our framework allows for a simple implementation of SGM and its end-to-end learning.

Non-CRF Propagation Many methods train continuous optimization algorithms used inside neural networks by unrolling their iterations [84, 162, 201]. Spatial propagation networks [114], their convolutional variant [34] and guided propagation [221] apply linear spatial propagation models in particular in stereo reconstruction. In difference, we train an inference algorithm that applies non-linear spatial propagation. From this point of view it becomes related to recurrent non-linear processing methods such PixelCNN [195].

4.3 Belief Propagation

In this section we give an overview of sum-product and max-product belief propagation (BP) algorithms and argue that max-marginals can be viewed as approximation to marginals. This

allows to connect learning with losses on marginals [45] and the max-product inference in a non-standard way, where the output is not simply the approximate MAP solution, but the whole volume of max-marginals.

Let $\mathcal{G} = (\mathcal{V}, E)$ be an undirected graph and $\mathcal{L}$ a discrete set of labels. A pairwise Markov Random Field (MRF) [104] over $\mathcal{G}$ with state space $\mathcal{V}^{\mathcal{L}}$ is a probabilistic graphical model $p : \mathcal{V}^{\mathcal{L}} \to \mathbb{R}_+$ that can be written in the form

$$p(x) = \frac{1}{Z} \exp\Big(\sum_{i \in \mathcal{V}} g_i(x_i) + \sum_{(i,j) \in \mathcal{E}} f_{ij}(x_i, x_j) \Big), \tag{4.1}$$

where Z is the normalization constant, functions $g_i : \mathcal{L} \to \mathbb{R}$ are the *unary scores*[2], typically containing data evidence; and functions $f_{ij} : \mathcal{L}^2 \to \mathbb{R}$ are *pairwise scores* measuring the compatibility of labels at nodes i and j. A CRF $p(x|y)$ is a MRF model (4.1) with scores depending on the inputs y.

Belief Propagation [151] was proposed to compute marginal probabilities of a MRF (4.1) when the graph $\mathcal{G}$ is a tree. BP iteratively sends *messages* $M_{ij} \in \mathbb{R}_+^L$ from node i to node j with the update:

$$M_{ij}^{k+1}(t) \propto \sum_s e^{g_i(s)} e^{f_{ij}(s,t)} \prod_{n \in \mathcal{N}(i) \setminus j} M_{ni}^k(s), \tag{4.2}$$

where $\mathcal{N}((, i))$ is the set of neighboring nodes of a node i and k is the iteration number. In a tree graph a message M_{ij} is proportional to the marginal probability that a configuration of a tree branch ending with (i, j) selects label t at j. Updates of all messages are iterated until the messages have converged. Then the marginals, or in a general graph *beliefs*, are defined as

$$B_i(x_i) \propto e^{g_i(x_i)} \prod_{n \in \mathcal{N}(i)} M_{ni}(x_i), \tag{4.3}$$

where the proportionality constant ensures $\sum_s B_i(s) = 1$.

The above *sum-product* variant of BP can be restated in the log domain, where the connection to max-product BP becomes apparent. We denote $\widetilde{\max}$ the operation $\mathbb{R}^n \to \mathbb{R}$ that maps $(a_1, \dots a_n)$ to $\log \sum_i e^{a_i}$, known as log-sum-exp or *smooth maximum*. The update of the sum-product BP (4.2) can be expressed as

$$m_{ij}^{k+1}(t) := \widetilde{\max_s} \Big(g_i(s) + f_{ij}(s,t) + \sum_{n \in \mathcal{N}(i) \setminus j} m_{ni}^k(s) \Big), \tag{4.4}$$

where m are the log domain messages, defined up to an additive constant. The *log-beliefs* are

[2] The negative scores are called *costs* in the context of minimization.

respectively

$$b_i(x_i) = g_i(x_i) + \sum_{n \in \mathcal{N}(i)} m_{ni}(x_i). \tag{4.5}$$

The *max-product BP* in the log domain takes the same form as (4.4) but with the hard max operation. Max-product solves the problem of finding the configuration x of the maximum probability (MAP solution) and computes *max-marginals* via (4.5). It can be viewed as an approximation to the marginals problem since there holds

$$\max_i a_i \leq \widetilde{\max_i} a_i \leq \max_i a_i + \log n \tag{4.6}$$

for any tuple $(a_1 \dots a_n)$. Preceding work has noticed that max-marginals can in practice be used to assess uncertainty [90], *i.e.*, they can be viewed as approximation to marginals. The perturb and MAP technique [148] makes the relation even more precise. In this work we apply max-marginal approximation to marginals as a practical and fast inference method for both, prediction time and learning. We rely on deep learning to make up for the approximation. In particular the learning can tighten (4.6) by scaling up all the inputs.

To summarize, the approximation to marginals that we construct is obtained by running the updates (4.4) with hard max and then computing beliefs from log-beliefs (4.5) as

$$B_i(x_i{=}s) = \operatorname*{softmax}_s b_i(s), \tag{4.7}$$

where $\operatorname{softmax}_s b_i(s) = e^{b_i(s)} / \sum_s e^{b_i(s)}$. Beliefs constructed in this way may be used in the loss functions on the marginal or as an input to subsequent layers, similarly to how simple logistic regression models are composed to form a sigmoid neural network. This approach is akin to previous work that used the regularized cost volume in a subsequent refinement step [80], but is better interpretable and learnable with our methods.

4.4 Sweep BP-Layer

When BP is applied in general graphs, the schedule of updates becomes important. We find that the parallel synchronous update schedule [152] requires too many iterations to propagate information over the image and rarely converges. For application in deep learning, we found that the schedule which makes sequential sweeps in different directions as proposed by [186] is more suitable. For a given sweep direction, we can compute the result of all sequential updates and backpropagate the gradient in a very efficient and parallel way. This allows to propagate information arbitrarily far in the sweep direction, while working on a pixel level, which makes this schedule very powerful.

Before detailing the sweep variant of BP [186], let us make clear what is needed in order to make an operation a part of an end-to-end learning framework. Let us denote the gradient of a loss function L in variables y as $đy := \frac{\mathrm{d}L}{\mathrm{d}y}$. If a layer computed $y = f(x)$ in the forward pass, the

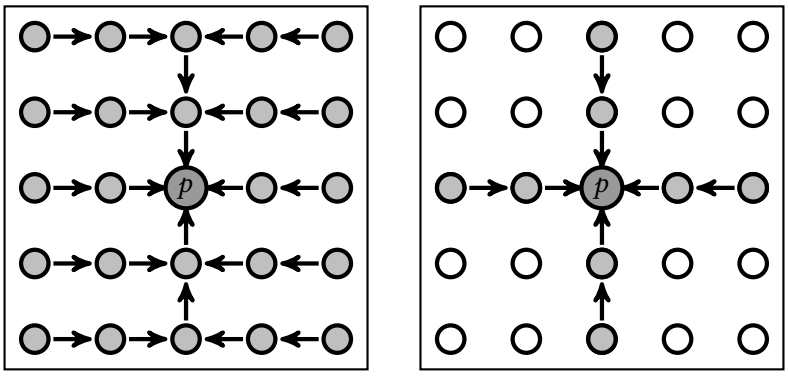


Figure 4.2: Max-marginal computation for node p on the highlighted trees. *Left*: Left-right-up-down BP [186] or equivalent tree DP [15]. *Right*: SGM [71] on a 4-connected graph. Note that SGM prediction for node p uses much smaller trees, ignoring the evidence from out of tree nodes.

gradient in x is obtained as

$$đx_j = \sum_i \frac{\partial f_i}{\partial x_j} đy_i, \tag{4.8}$$

called the *backprop* of layer f. For the BP-Layer the input probabilities x and output beliefs y are big arrays containing all pixels and all labels. It is therefore crucial to be able to compute the backprop in linear time.

4.4.1 Sweep BP as Dynamic Programming

The BP variant of [186] (called left-right-up-down BP there and BP-M in [185]) performs sweeps in directions left→right, right→left, up→down, down→up. For each direction only messages in that direction are updated sequentially, and the rest is kept unmodified. We observe the following properties of this sweep BP: (i) Left and right messages do not depend on each other and neither on the up and down messages. Therefore, their calculation can run independently in all horizontal chains. (ii) When left-right messages are fixed, they can be combined into unary scores, which makes it possible to compute the up and down messages independently in all vertical chains in a similar manner. These properties allow us to express left-right-up-down BP as shown in algorithm 16 and illustrated in Fig. 4.2 (left). In algorithm 16, the notation $a_{\mathcal{V}'}$ means the restriction of a to the nodes in $\mathcal{V}'$, *i.e.* to a chain.

It is composed of dynamic programming subroutines computing max-marginals. Since individual chains in each of the loops do not interact, they can be processed in parallel (denoted as par. for). The max-marginals a of a horizontal chain are computed as

$$a_i(s) = g_i(s) + m_i^L(s) + m_i^R(s), \tag{4.9}$$

where $m_i^L(s)$ denotes the message to i from its left neighbour and $m_i^R(s)$ from its right. The max-marginals (4.9) are indeed the beliefs after the left-right pass. The max-marginals b for vertical

Algorithm 16: Sweep Belief Propagation

Input: CRF scores $g \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$, $f \in \mathbb{R}^{E\times\mathcal{L}^2}$;
Output: Beliefs $B \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$;

par. for *each horizontal chain subgraph* $(\mathcal{V}', E')$ **do**
  $a_{\mathcal{V}'} := \texttt{max\_marginals}(g_{\mathcal{V}'}, f_{E'})$;
par. for *each vertical chain subgraph* $(\mathcal{V}', E')$ **do**
  $b_{\mathcal{V}'} := \texttt{max\_marginals}(a_{\mathcal{V}'}, f_{E'})$;
return beliefs $B_i(s) := \text{softmax}_s(b_i(s))$;

Algorithm 17: Dynamic Programming (DP)

Input: Directed chain $(\mathcal{V}, E)$, nodes $\mathcal{V}$ enumerated in chain direction from 0 to $n{=}|\mathcal{V}|{-}1$, scores $g \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$, $f \in \mathbb{R}^{E\times\mathcal{L}^2}$;
Output: Messages $m \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$ in chain direction;

Init: Set: $m_0(s) := 0$; `/* first node */`
for $i = 0 \ldots n-2$ **do**
  `/* Compute message: */`
  $m_{i+1}(t) := \max_s \big(g_i(s) + m_i(s) + f_{i,i+1}(s,t)\big)$;
  `/* Save argmax for backward: */`
  $o_{i+1}(t) := \operatorname{argmax}_s \big(g_i(s) + m_i(s) + f_{i,i+1}(s,t)\big)$;
return m;

Algorithm 18: Backprop DP

Input: $đm \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$, gradient of the loss in the messages m returned by DP on chain $(\mathcal{V}, E)$;
Output: $đg \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$, $đf \in \mathbb{R}^{E\times\mathcal{L}^2}$, gradients of the loss in the DP inputs g, f;

Init: $đg := 0$; $đf := 0$;
for $i = n-2 \ldots 0$ **do**
  for $t \in \mathcal{L}$ **do**
    $s := o_{i+1}(t)$;
    $z := đm_{i+1}(t) + đg_{i+1}(t)$;
    $đg_i(s)$ += z;
    $đf_{i,i+1}(s,t)$ += z;
return $đg$, $đf$;

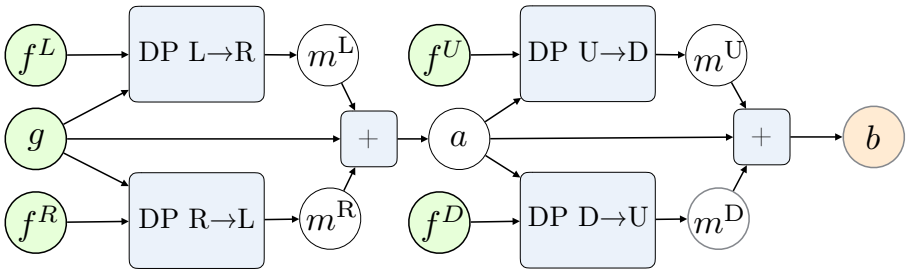


Figure 4.3: Computation graph of BP-Layer with Sweep BP in algorithm 16 down to log-beliefs b. Dynamic Programming computational nodes (DP) are made differentiable with the backprop in algorithm 18. The pairwise terms f^{L}, f^{R}, f^{U}, f^{D} illustrate the case when pairwise scores f_{ij} are different for all four directions.

chains are, respectively,

$$b_i(s) = a_i(s) + m_i^U(s) + m_i^D(s). \tag{4.10}$$

It remains to define how the messages m are computed and back-propagated. Given a chain and the processing direction (*i.e.*, L-R for left messages m^{L}), we order the nodes ascending in this direction and apply dynamic programming in algorithm 17. The Jacobian of algorithm 17 is well defined if the maximizer in each step is unique[3]. In this case we have a linear recurrent dependence in the vicinity of the input:

$$m_{i+1}(t) = g_i(s) + m_i(s) + f_{i,i+1}(s,t), \tag{4.11}$$

where $s = o_{i+1}(t)$, *i.e.* the label maximizing the message, as defined in algorithm 17. Back-propagating this linear dependence is similar to multiplying by the transposed matrix, *e.g.*, for the gradient in $g_i(s)$ we need to accumulate over all elements to which $g_i(s)$ is contributing. This can be efficiently done as proposed in algorithm 18.

Thus we have completely defined sweep BP, further on referred to as *BP-Layer*, as a composition of differential operations. The computation graph of the BP-Layer shown in Fig. 4.3 can be back-propagated using standard rules and our Backprop DP in order to compute the gradients in all inputs very efficiently.

4.4.2 Other Inference Methods

We show the generality of the proposed framework by mapping several other inference techniques to the same simple DP operations. This allows to make them automatically differentiable and suitable for learning with marginal losses.

[3]Otherwise we take any maximizer resulting in a conditional derivative like with ReLU at 0.

Algorithm 19: Semi-Global Matching

Input: CRF scores $g \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$, $f \in \mathbb{R}^{E\times\mathcal{L}^2}$;
Output: Beliefs $b \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$;

par. for *each direction* k *in* $\{\mathrm{L}, \mathrm{R}, \mathrm{U}, \mathrm{D}\}$ **do**
  par. for *each chain* $(\mathcal{V}', E')$ *in direction to* k **do**
    $m^k_{\mathcal{V}'} := DP(g_{\mathcal{V}'}, f_{E'})$;
return $b = g + \sum_k m^k$;

4.4.2.1 SGM

We can implement SGM using the same DP function we needed for BP (algorithm 19), where for brevity we considered a 4-connected grid graph. As discussed in the related work, the possibility to backpropagate SGM was previously missing and may be useful.

4.4.2.2 Tree-structured DP

Bleyer and Gelautz [15] proposed an improvement to SGM by extending the local tree as shown in Fig. 4.2 (left), later used *e.g.* in a very accurate stereo matching method [211]. It seems it has not been noticed before that sweep BP [186] is exactly equivalent to the tree-structured DP of [15], as clearly seen from our presentation.

4.4.2.3 TRW and TBCA

With minor modifications of the already defined DP subroutines, it is possible to implement and back-propagate several inference algorithms addressing the dual of the LP relaxation of the CRF: the Tree-Reweighted (TRW) algorithm by Wainwright et al. [203] and Tree Block Coordinate Ascent (TBCA) by Sontag and Jaakkola [181], which we show in the upcoming paragraphs. These algorithms are parallel, incorporate long-range interactions and avoid the evidence over-counting problems associated with loopy BP [203]. In addition, the TBCA algorithm is monotone and has convergence guarantees. These methods are therefore good candidates for end-to-end learning, however they may require more iterations due to cautious monotone updates, which is undesirable in the applications we consider. Nevertheless, as they improve on the issues of BP in loopy graphs, this makes them potential candidates for drop-in replacement of our sweep BP-layer.

Tree Re-weighted BP Wainwright et al. [203] proposed a correction to BP, which turns it into a variational inference algorithm optimizing the dual of the LP relaxation. Suppose that we are given an edge-disjoint decomposition of the graph into trees. For our models it is convenient to take horizontal and vertical chain subproblems. The TRW-T algorithm [203] can be implemented as proposed in algorithm 20. In this representation we keep the decomposition into subproblems explicitly and messages are encapsulated in the computation of max-marginals. This is in order to reuse the same subroutines we already have for BP-Layer. An explicit form of updates in terms of

Algorithm 20: Tree Reweighted BP (TRW-T)

Input: CRF scores $g \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$, $f \in \mathbb{R}^{E\times\mathcal{L}^2}$;
Output: Beliefs $B \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$;

$g^{\mathrm{h}} := g^{\mathrm{v}} := \frac{1}{2}g$;
for *iteration* $t = 1 \dots T$ **do**
  `/* Compute max-marginals: */`
  par. for *horizontal chain subgraphs* $(\mathcal{V}', E')$ **do**
    $b^{\mathrm{h}}_{\mathcal{V}'} := \texttt{max\_marginals}(g^{\mathrm{h}}_{\mathcal{V}'}, f_{E'})$;
  par. for *vertical chain subgraphs* $(\mathcal{V}', E')$ **do**
    $b^{\mathrm{v}}_{\mathcal{V}'} := \texttt{max\_marginals}(g^{\mathrm{v}}_{\mathcal{V}'}, f_{E'})$;
  `/* Enforce consistency: */`
  $b := (b^{\mathrm{h}} + b^{\mathrm{v}})$;
  $g^{\mathrm{h}} \mathrel{+}= (\frac{1}{2}b - b^{h})$;
  $g^{\mathrm{v}} \mathrel{+}= (\frac{1}{2}b - b^{v})$;
return Log-beliefs b;

messages only which reveals the similarity to loopy belief propagation with weighting coefficients can be also given [203]. This algorithm is not guaranteed to be monotonous because it does block-coordinate ascent steps in multiple blocks in parallel. However thanks to parallelization it is fast to compute (in particular on a GPU), incorporates long-range interactions and avoids the over-counting problems associated with loopy BP [203].

Tree Block Coordinate Ascent The TBCA method [181] is an inference algorithm optimizing the dual of the LP relaxation. It does so by a block-coordinate ascent in the variables associated with tree-structured subproblems. The variables are the same as the messages in BP. At each iteration a sub-tree $(\mathcal{V}', E')$ from the graph is selected. For simplicity and ease of parallelization we will assume $(\mathcal{V}', E')$ is a horizontal chain and consider it to be ordered from left to right. The following updates are performed on this chain:

- Compute the current reparametrized costs, excluding the messages from inside the chain:

$$a_i(s) = g_i(s) + \sum_{(i,j)\in E\setminus E'} m_{ji}(s) \forall i \in \mathcal{V}'. \tag{4.12}$$

- Compute the right messages m^{R} by DP in the direction R→L.
- Compute the left messages m^{L} by a *redistribution DP* (rDP) in the direction R→L.

We can write the rDP update equation [181] in the form

$$m^{\mathrm{L}}_{i+1}(t) := \max_s \Big(\tilde{g}_i(s) + r_i m^{\mathrm{L}}_i(s) + f_{i,i+1}(s,t)\Big), \tag{4.13}$$

where $\tilde{g}_i(s) = g_i(s) + (1 - r_i)m_i^{\mathrm{R}}(s)$ and $r_i \in [0, 1]$ are the redistribution coefficients. For $r = 1$, this recovers the regular dynamic programming. Similarly to DP, the update is linear and depend on the current maximizers that we record as $o_{i+1}(t)$. It differs from DP in two ways: i) it depends on the right messages, which we have taken into account by incorporating them to the unary costs in $\tilde{g}_i(s)$ and ii) there are coefficients r_i in the recursion. To handle the latter, we only need to modify algorithm 18 of algorithm 18 to

$$z := đm_{i+1}(t) + r_{i+1}đg_{i+1}(t). \tag{4.14}$$

It follows that we have defined the TBCA subproblem update with standard operations on tensors and the two new operations DP and rDP, for which we have shown efficient backprop methods. The TBCA method [181], when specialized to horizontal and vertical chains, would then alternate the above updates in parallel for all horizontal chains and then for all vertical chains. This method also achieves high parallelization efficiency and long-range propagation. Thanks to the redistribution mechanism it is also guaranteed to be monotonous. However, this monotonicity may slow down the information propagation, which can make it less suitable as a truncated inference technique in deep learning.

4.5 Models

We demonstrate the effectiveness of the BP-Layer on the three labeling problems: Stereo, Optical Flow and Semantic Segmentation. We have two CNNs (Table 4.1) which are used to compute i) score-volumes and ii) pairwise jump-scores, at three resolution levels used hierarchically. Fig. 4.4 shows processing of one resolution level with the BP-Layer. The label probabilities from these predictions are considered as weak classifiers and the inference block combines them to output a stronger finer-resolution classification. Accordingly, the unary scores $g_i(s)$, called the *score volume*, are set from the CNN prediction probabilities $q_i(s)$ as

$$g_i(s) = Tq_i(s), \tag{4.15}$$

where T is a learnable parameter. Note that g_i is itself a linear parameter of the exponential model (4.1). The preceding work more commonly used the model $g_i(s) = \log q_i(x)$, which, in the absence of interactions, recovers back the input probabilities. In contrast, the model (4.15) has the following interpretation and properties: i) it can be viewed as just another non-linearity in the network, increasing flexibility; ii) in case of stereo and flow it corresponds to a robust metric in the feature space (see below), in particular it is robust to CNN predictive probabilities being poorly calibrated.

To combine the up-sampled beliefs B^{up} from the coarser-resolution BP-Layer with a finer-resolution evidence q, we trilinearly upsample the beliefs from the lower level and add it to the

score-volume of the current level, *i.e.*

$$g_i(s) = T\big(q_i(s) + B_i^{\text{up}}(s)\big). \tag{4.16}$$

On the output we have an optional refinement block, which is useful for predicting continuous values for stereo and flow. The simplest refinement takes the average in a window around the maximum:

$$y = \sum_{d:|d-\hat{d}_i|\leq\tau} d\, B_i(d) \Big(\sum_{d:|d-\hat{d}_i|\leq\tau} B_i(d) \Big)^{-1}, \tag{4.17}$$

where $\hat{d}_i = \operatorname{argmax} B_i(d)$ and we use the threshold $\tau = 3$. Such averaging is not affected by a multi-modal distribution, unlike the full average used in [78]. As a more advanced refinement block we use a variant of the refinement [80] with one up-sampling step using also the confidence of our prediction as an additional input.

4.5.1 Stereo

For the rectified stereo problem we use two instances of a variant of the UNet detailed in Section 4.7. This network is relatively shallow and contains significantly fewer parameters than SoTA. It is applied to the two input images I^0, I^1 and produces two dense feature maps f^0, f^1. The initial prediction of disparity k at pixel i is formed by the distribution

$$q_i(k) = \underset{k\in\{0,1,\ldots,D\}}{\operatorname{softmax}} \big(-\|f^0(i) - f^1(i-k)\|_1 \big), \tag{4.18}$$

where $i-k$ denotes the pixel location in image I^1 corresponding to location i in the reference image I^0 and disparity k and D is the maximum disparity. This model is related to robust costs [95]. The pairwise terms f_{ij} are parametric like in the SGM model [71] but with context-dependent parameters. Specifically, f_{ij} scores difference of disparity labels in the neighbouring pixels. Disparity differences of up to 3 pixels have individual scores, all larger disparity jumps have the same score. All these scores are made context dependent by regressing them with our second UNet from the reference image I^0.

4.5.2 Optical Flow

The optical flow problem is very similar to stereo. Instead of two rectified images, we consider now two consecutive frames in a video, I^0 and I^1. We use the same UNets to compute the per-pixel features and the jump scores as in the stereo setting. The difference lies in the computation of the initial prediction of flow $u = (u_1, u_2)$. The flow for a pixel i is formed by the two distributions

$$q_i^1(u_1) = \underset{u_1}{\operatorname{softmax}} \max_{u_2} \big(-\|f^0(i) - f^1(i+u)\|_1 \big), \tag{4.19}$$

$$q_i^2(u_2) = \underset{u_2}{\operatorname{softmax}} \max_{u_1} \big(-\|f^0(i) - f^1(i+u)\|_1 \big), \tag{4.20}$$

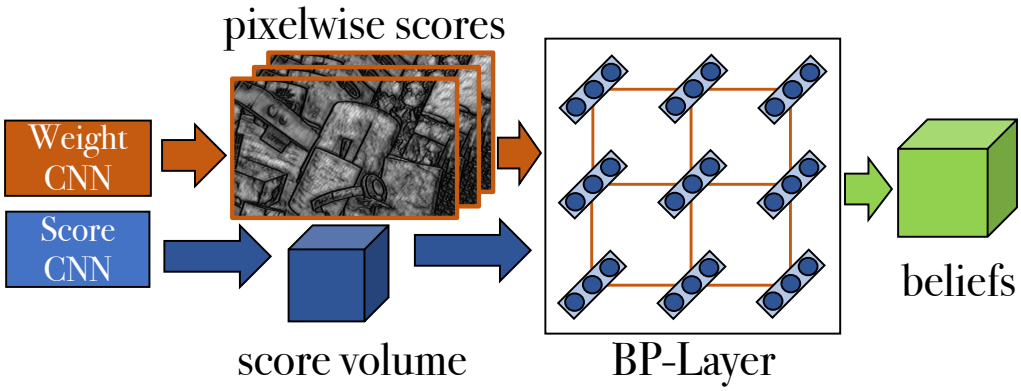


Figure 4.4: BP-Layer overview. The weight and score CNNs compute pixelwise weights and a score volume from the input image. This is used as an input for the BP-Layer which returns beliefs as an output.

which follows the scalable model of Munda et al. [135], avoiding the storage of all matching scores that for an $M \times N$ image have the size $M \times N \times D^2$. The inner maximization steps correspond to the first iteration of an approximate MAP inference [135]. They form an "optimistic" estimate of the score volume for each component of the optical flow, which we process then independently. This scheme may be sub-optimal in that u^1 and u^2 components are inferred independently until the refinement layer, but it scales well to high resolutions (the search window size D needs to grow with the resolution as well) and allows us to readily apply the same BP-Layer model as for the stereo to q^1 and q^2 input probabilities.

4.5.3 Semantic Segmentation

The task in semantic segmentation is to assign a semantic class label from a number of classes to each pixel. In our model, the initial prediction probabilities are obtained with the ESPNet [128], a lightweight solution for pixel-wise semantic segmentation. This initial prediction is followed up directly with the BP-Layer, which can work with two different types of pairwise scores f_{ij}. The inhomogeneous anisotropic pairwise terms depend on each pixel and on the edge direction, while the homogeneous anisotropic scores depend only on the edge direction. We implement the homogeneous pairwise terms as parameters within the model and constrain them to be non-negative. The pixel-wise pairwise-terms are computed from the input image using the same UNet as in stereo. We follow the training scheme of [128].

4.6 Learning

We use the same training procedure for all three tasks. Only the loss function is adapted for the respective task. The loss function is applied to the output of each BP-Layer in the coarse-to-fine scheme and also to the final output after the refinement layer. Such a training scheme is known as deep supervision [109]. For BP output beliefs B^l at level l of the coarse-to-fine scheme, we apply at

each pixel i the negative log-likelihood loss $\ell_{\text{NLL}}(B_i^l, d_i^{*l}) = -\log B_i^l(d_i^{*l})$, where d_i^{*l} is the ground truth disparity at scale l.

For the stereo and flow models that have a refinement block targeting real-valued predictions, we add a loss penalizing at each pixel the distance from the target value according to the Huber function:

$$\ell_H(y_i, y_i^*) = \begin{cases} \frac{r^2}{2\delta} & \text{if } |r| \leq \delta, \\ |r| - \frac{\delta}{2} & \text{otherwise,} \end{cases} \tag{4.21}$$

where y_i is the continuous prediction of the model, y_i^* is the ground-truth and $r = y_i - y_i^*$.

Losses at all levels and the losses on the continuous-valued outputs are combined with equal weights[4].

4.7 Implementation Details

We implemented our model in PyTorch[5] and the core of the BP-layer as a highly efficient CUDA kernel. For geometrical problems such as stereo and optical flow, we use a truncated compatibility function (see Fig. 4.5a). This is allows us to decrease the asymptotic runtime for K labels to $O(K)$ and makes very efficient inference and training possible. For semantic segmentation we want to learn the full compatibility matrix. Nevertheless, since we learn the cost from any source label to any target label, the runtime is $O(K^2)$ and thus quadratic in the number of labels. The practical impact on the runtime can be seen in Tables 4.3 and 4.6.

In our optimized CUDA implementation we utilize the following parallelization: All chains of the same direction as well as the chains in the opposing directions can be processed in parallel. Furthermore, the message-passing also parallelizes over the labels. For an image of size $N \times N$, assuming that the number of disparities also grows as $K = O(N)$, our implementation achieves parallelism of $O(N^2)$ while requiring sequential processing $O(N)$, which is an acceptable scaling with the image size. The backprop operation of the DP, has the same level of parallelism, which is important for large-scale learning. These implementations are connected as extensions to PyTorch, which allows them to be used in any computation graphs. In order to increase numerical accuracy, we also normalize the messages by subtracting the maximum over all labels on each step of DP. This does not affect the output beliefs, as the normalization cancels in the softmax operation.

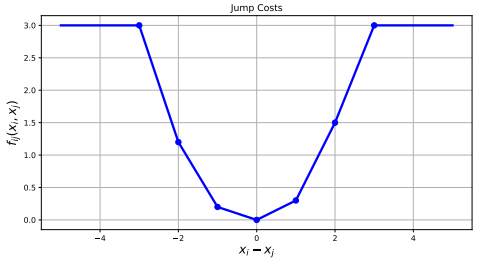
We trained the model with the Adam optimizer [81] with a learning rate of $3 \cdot 10^{-3}$. We always start with a pre-training for 300k iterations on large scale synthetic data for stereo and optical flow to get a good initialization for our model. Finally, we fine-tune the pre-trained models on the target dataset for 1000 epochs using a learn-rate of 10^{-5}.

4.7.1 Runtime Analysis

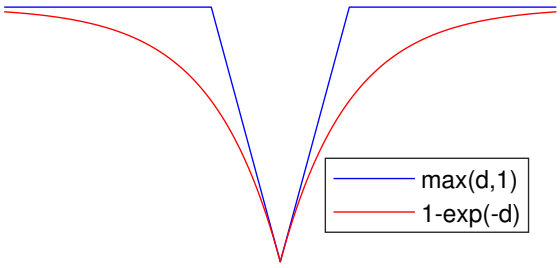
We give a brief comparison of the runtime of the proposed BP-Layer and 3D convolutions here. Compared to other networks such as [28, 78, 221] we completely avoid the usage of the very

[4] the relative weights could be considered as hyper-parameters, but we did not tune them.

[5] `https://pytorch.org`



(a) Robust penalty function. Similar as the $P1, P2$ model in SGM, but with one additional learnable step. We allow to learn this function asymmetrically, because positive occlusions appear only on left-sided object boundaries.



(b) The cost $-g_i(k)$ as a function of $d = \|f^0(i) - f^1(i-k)\|_1$ in our model is similar to robust costs $\max(d, \tau)$ previously used to better handle occlusions [95].

costly 3D convolution layers. 3D convolution layers have a runtime of $O(MNKCP^3)$ while our proposed BP-Layer has a runtime of $O(MNK)$, where M and N are the width and the height of the image, K is the number of disparities, C is the number of feature channels and P is the size of the 3D kernel. Although Zhang et al. [221] have a similar runtime of their SGA Layer, they still use 15 3D conv layers with 48 feature volumes in every layer in their full model which is very expensive. Note that their LGA Layer also operates on a 4D input, *i.e.* on multiple 3D feature volumes, where in difference we use only one 3D volume in all stereo experiments. Chang and Chen [28], Kendall et al. [78] use 19 and 25 3D conv layers, respectively. In difference, as our ablation study in the main paper shows, we are on-par with these methods on several metrics. Furthermore, our method is the only method which is also able to achieve high quality results on the difficult Middlebury 2014 benchmark.

4.7.2 Model Architecture

Table 4.1 shows our very lightweight architecture which we use for feature extraction. "convXX" with XX being to scalars denotes a block with consisting of a Convolution, GroupNormalization and the LeakyReLU activation function We actually maintain two copies of this networks with non-shared parameters. The first one is used as the feature network for matching and the second one is the feature network for predicting the pairwise jump-scores. Figs. 4.5a and 4.5b show the functions used for unary costs and pairwise costs respectively. Note, that both functions are robust due to the truncation.

On every hierarchical level we add one convolution layer to map the features to pixel-wise descriptors used for matching and to pixel-wise jump-scores respectively. We denote the convolutions as "convD{0,1,2}" and "convS{0,1,2}", where D stands for disparity and S for scores. We do not apply any activation function after these two convolutions in order to not remove any information which could be utilized in the feature matching. The highest resolution is here level 0 and the lowest resolution is level 2 in our setting. In the last group in Table 4.2 we show the hierarchical inference block. We apply our BP-Layer on the score-volume with the coarsest scale,

Layer	KS	Resolution	Channels	Input
conv00	3	$W \times H$ / $W \times H$	3 / 16	Image
conv01	3	$W \times H$ / $W \times H$	16 / 16	conv00
pool0	2	$W \times H$ / $\frac{W}{2} \times \frac{H}{2}$	16 / 16	conv01
conv10	3	$\frac{W}{2} \times \frac{H}{2}$ / $\frac{W}{2} \times \frac{H}{2}$	16 / 32	pool0
conv11	3	$\frac{W}{2} \times \frac{H}{2}$ / $\frac{W}{2} \times \frac{H}{2}$	32 / 32	conv10
pool1	2	$\frac{W}{2} \times \frac{H}{2}$ / $\frac{W}{4} \times \frac{H}{4}$	32 / 32	conv10
conv20	3	$\frac{W}{4} \times \frac{H}{4}$ / $\frac{W}{4} \times \frac{H}{4}$	32 / 64	pool1
conv21	3	$\frac{W}{4} \times \frac{H}{4}$ / $\frac{W}{4} \times \frac{H}{4}$	64 / 64	conv20
bilin1	-	$\frac{W}{4} \times \frac{H}{4}$ / $\frac{W}{2} \times \frac{H}{2}$	64 / 64	conv21
conv12	3	$\frac{W}{2} \times \frac{H}{2}$ / $\frac{W}{2} \times \frac{H}{2}$	96 / 32	{bilin1, conv11}
conv13	3	$\frac{W}{2} \times \frac{H}{2}$ / $\frac{W}{2} \times \frac{H}{2}$	32 / 32	conv12
bilin0	-	$\frac{W}{2} \times \frac{H}{2}$ / $W \times H$	32 / 32	conv12
conv02	3	$W \times H$ / $W \times H$	48 / 32	{bilin0, conv01}
conv03	3	$W \times H$ / $W \times H$	32 / 32	conv02

Table 4.1: Detailed Architecture of our UNet for feature extraction.

i.e. level 2, upsample the result trilinearly and combine it with SAD matching from the next level. We apply this procedure until we get a regularized score-volume on the finest level, *i.e.* level 0.

Note that the resolutions given in Tables 4.1 and 4.2 are relative to the input image size. We use with a factor 2 bilinearly downsampled images as the input to our feature networks in all experiments but Kitti. In Kitti we do all computations on the full-size images directly.

4.8 Experiments

We implemented the BP-Layer and hierarchical model in PyTorch and used CUDA extensions for time and memory-critical functions (forward and backward for DP, score volume min-projections).[6]

4.8.1 Improvements brought by the BP-Layer

We investigate the importance of different architectural choices in our general model on the stereo task with the synthetic stereo data from the Scene Flow dataset [125]. The standard error metric in stereo is the bad$X$ error measuring the percentage of disparities having a distance larger than X to the ground-truth. This metric is used to assess the robustness of a stereo algorithm. The second

[6] `https://github.com/VLOGroup/bp-layers`

Layer	KS	Resolution	Channels	Input
convD2	3	$\frac{W}{4} \times \frac{W}{4}$ / $\frac{H}{4} \times \frac{H}{4}$	64 / 32	conv21
convD1	3	$\frac{W}{2} \times \frac{W}{2}$ / $\frac{H}{2} \times \frac{H}{2}$	32 / 32	conv13
convD0	3	$W \times W$ / $H \times H$	32 / 32	conv03
convS2	3	$\frac{W}{4} \times \frac{W}{4}$ / $\frac{H}{4} \times \frac{H}{4}$	64 / 32	conv21
convS1	3	$\frac{W}{2} \times \frac{W}{2}$ / $\frac{H}{2} \times \frac{H}{2}$	32 / 32	conv13
convS0	3	$W \times W$ / $H \times H$	32 / 32	conv03
sad2	-	$\frac{W}{4} \times \frac{W}{4}$ / $\frac{H}{4} \times \frac{H}{4}$	32 / $\frac{D}{4}$	convD2_0, convD2_1
sad1	-	$\frac{W}{2} \times \frac{W}{2}$ / $\frac{H}{2} \times \frac{H}{2}$	32 / $\frac{D}{2}$	convD1_0, convD1_1
sad0	-	$W \times W$ / $H \times H$	32 / D	convD0_0, convD0_1
BP2	-	$\frac{W}{4} \times \frac{W}{4}$ / $\frac{H}{4} \times \frac{H}{4}$	$\frac{D}{4}$ / $\frac{D}{4}$	sad2, convS2
BP2_up	-	$\frac{W}{4} \times \frac{W}{4}$ / $\frac{H}{2} \times \frac{H}{2}$	$\frac{D}{4}$ / $\frac{D}{2}$	BP2
BP1	-	$\frac{W}{2} \times \frac{W}{2}$ / $\frac{H}{2} \times \frac{H}{2}$	$\frac{D}{2}$ / $\frac{D}{2}$	sad1 + BP2_up, convS1
BP1_up	-	$\frac{W}{2} \times \frac{W}{2}$ / $W \times H$	$\frac{D}{2}$ / D	BP1
BP0	-	$W \times W$ / $H \times H$	D / D	sad0 + BP1_up, convS0

Table 4.2: Hierarchical BP inference block. We add convolutions to map the features from the feature net to appropriate input to our BP-Layer. The plus operation '+' indicates a point-wise addition.

Model	#P	time	bad1		bad3		MAE	
WTA (NLL)	0.13	0.07	10.3	(18.0)	5.27	(13.2)	3.82	(15.1)
BP (NLL)	0.27	0.10	12.6	(17.9)	4.97	(8.12)	1.23	(3.36)
BP+MS (NLL)	0.33	0.11	10.0	(16.5)	3.66	(7.86)	1.13	(2.84)
BP+MS (H)	0.33	0.11	8.15	(15.1)	3.07	(8.00)	0.96	(3.42)
BP+MS+Ref (H)	0.56	0.15	**7.73**	(**13.8**)	**2.67**	(6.46)	**0.74**	(1.67)
GC-Net [78]	3.5	0.95	-	(16.9)	-	(9.34)	-	(2.51)
GA-Net-1 [221]	0.5	0.17	-	(16.5)	-	(-)	-	(1.82)
PDS-Net [194]	2.2	-	-	(-)	-	(**3.38**)	-	(**1.12**)

Table 4.3: Ablation Study on the Scene flow validation set. We report for all metrics the result on non-occluded and (all pixels). #P in millions. bold = best, underline = second best.

metric is the mean-absolute-error (MAE) which is more sensitive to the (sub-pixel) precision of a stereo algorithm.

Table 4.3 shows an overview of all variants of our model. We start from the winner-takes-all **(WTA)** model, add the proposed BP-Layer or the multi-scale model **(MS)**, then add the basic

Method	#P[M]	Middlebury 2014 bad2	Middlebury 2014 time[s]	Kitti 2015 bad3	Kitti 2015 time[s]
PSMNet [28]	5.2	42.1 (47.2)	2.62	2.14 (2.32)	0.41
PDS [194]	**2.2**	14.2 (21.0)	12.5	2.36 (2.58)	0.50
HSM [210]	3.2	**10.2** (**16.5**)	**0.51**	**1.92** (**2.14**)	**0.14**
MC-CNN [220]	**0.2**	**9.47** (20.6)	1.26	3.33 (3.89)	67.0
CNN-CRF [85]	0.3	12.5 (21.9)	3.53	4.84 (5.50)	1.30
ContentCNN [120]	0.7	-	-	4.00 (4.54)	1.00
LBPS **(ours)**	0.3	9.68 (**17.5**)	**1.05**	**3.13** (**3.44**)	**0.39**

Table 4.4: Evaluation on the Test set of the Middlebury and Kitti Stereo Benchmark using the default metrics of the respective benchmarks. *Top group*: Large models with > 1M parameters. *Bottom group*: Light-weight models. Bold indicates the best result in the group.

refinement (4.17) trained with Huber loss **(H)**, then add the refinement [80] (**Ref (H)**). The column #P in Table 4.3 shows the number of parameters of our model, which is significantly smaller than SoTA methods applicable to this dataset. Each of the parts of our model increase the final performance. Our algorithm performs outstandingly well in the robustness metric bad$X$. The ablation study shows also the impact of the used loss function. It turns out that Huber loss function is beneficial to all the metrics but the MAE in occluded pixels. The optional refinement yielded an additional improvement, especially in occluded pixels on this data, but we could not obtain a similar improvement when training and validating on Middlebury or Kitti datasets. We therefore selected BP+MS (H) model, as the more robust variant, for evaluation in these real-data benchmarks.

4.8.2 Stereo Benchmark Performance

We use the model BP+MS (H) to participate on the public benchmarks of Middlebury 2014 and Kitti 2015. Both benchmarks have real-world scences, Middlebury focusing on high-resolution indoor scenes and Kitti focusing on low-resolution autonomous driving outdoor scenes. Qualitative test-set results are shown in Fig. 4.6.

The Middlebury benchmark is very challenging due to huge images, large maximum disparities, large untextured regions and difficult illumination. These properties make it hard or even impossible for most of the best-performing methods from Kitti to be used on the Middlebury benchmark. Due to our light-weight architecture we can easily apply our model on the challenging Middlebury images. The test-set evaluation (Table 4.4) shows that we are among the best performing methods with a runtime of up to 10 seconds, and thus convincingly shows the effectiveness of our light-weight model. The challenges on the Kitti dataset are regions with over- and under-saturation, reflections and complex geometry. We significantly outperform competitors with a similar number of parameters such as MC-CNN, CNN-CRF and Content CNN, which demonstrates the effectiveness of the learnable BP-Layer. Methods achieving a better performance

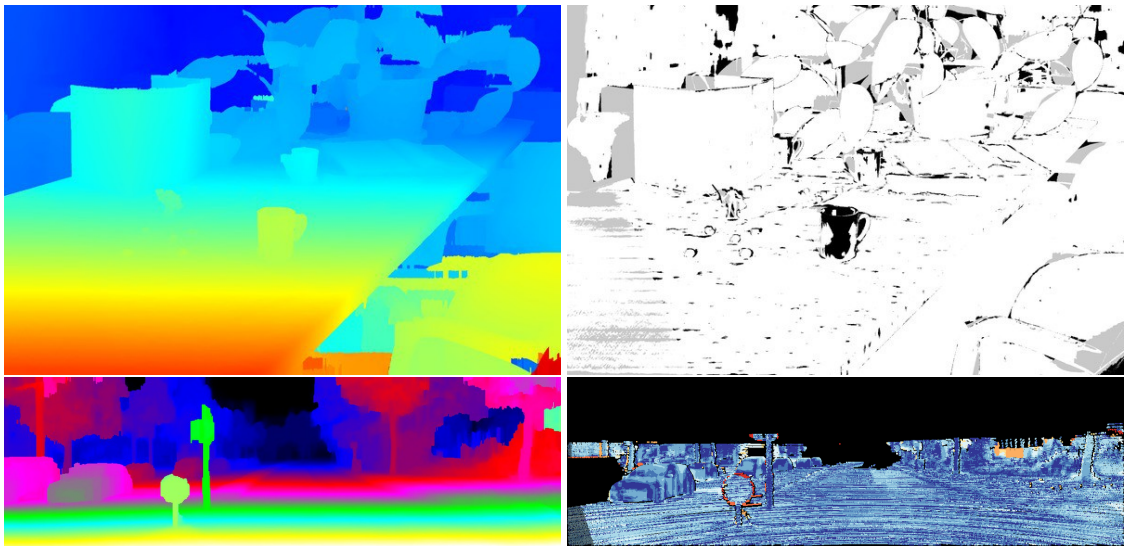


Figure 4.6: Qualitative results on the test sets of Middlebury 2014 (top) and Kitti 2015 (bottom) datasets. Left: Color coded disparity map, right error map, where white/blue = correct, gray = occluded, black/orange = incorrect. Note how our method produces sharp edges in all results.

on Kitti come with the high price of having many more parameters.

4.8.3 More stereo experiments

Fig. 4.7 shows a qualitative ablation study comparing our model variants on selected images. Note that we show here exactly the same model variants as in Table 4.3. The visual ablation study shows interesting insights about our models: First, the WTA result (2nd row in Fig. 4.7) is already a very good initialization on all matchable pixels although we use a very efficient network (Table 4.1) which uses only 130k parameters. The BP-Layer regularizes the WTA solution by removing most of the artifacts, especially in occluded regions as can be seen in the 3rd row. However, due to the NLL loss function the discretization artifacts are visible in *e.g.* the 3rd example from left. The multi-scale variant adds robustness in large, untextured regions as can be seen in *e.g.* example 1 on the gray box. Training with the Huber loss (row 5) enables sub-pixel accurate solutions. Note how this model captures fine details such as the bar better than the previous models. Our final model can then be used to recover very fine details such as the spokes of the motorcycle in the first example.

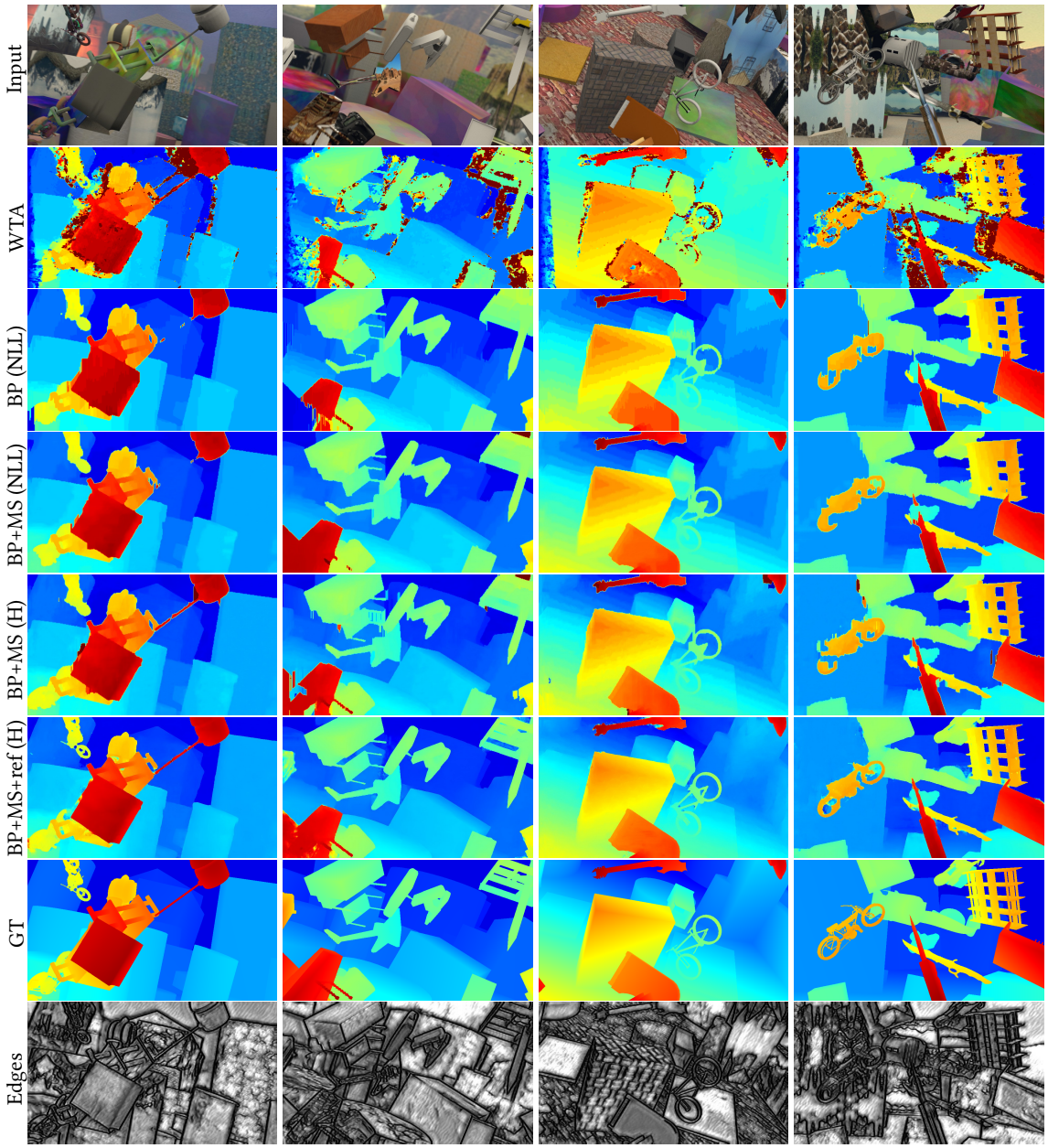


Figure 4.7: Visual ablation study. The methods are the same as used in the quantitative ablation study (Table 4.3) and compared from top to bottom. The last row shows the learned jump-costs of BP+MS+Ref (H) used in our BP-Layer, where black=low cost and white=high cost. The edge images are easily interpretable. We can see that the object edges and depth discontinuities are precisely captured.

Figs. 4.8 and 4.10 show additional qualitative results on the Middlebury 2014 test set and the Kitti 2015 test set. We include the input image and the error images which are provided by the

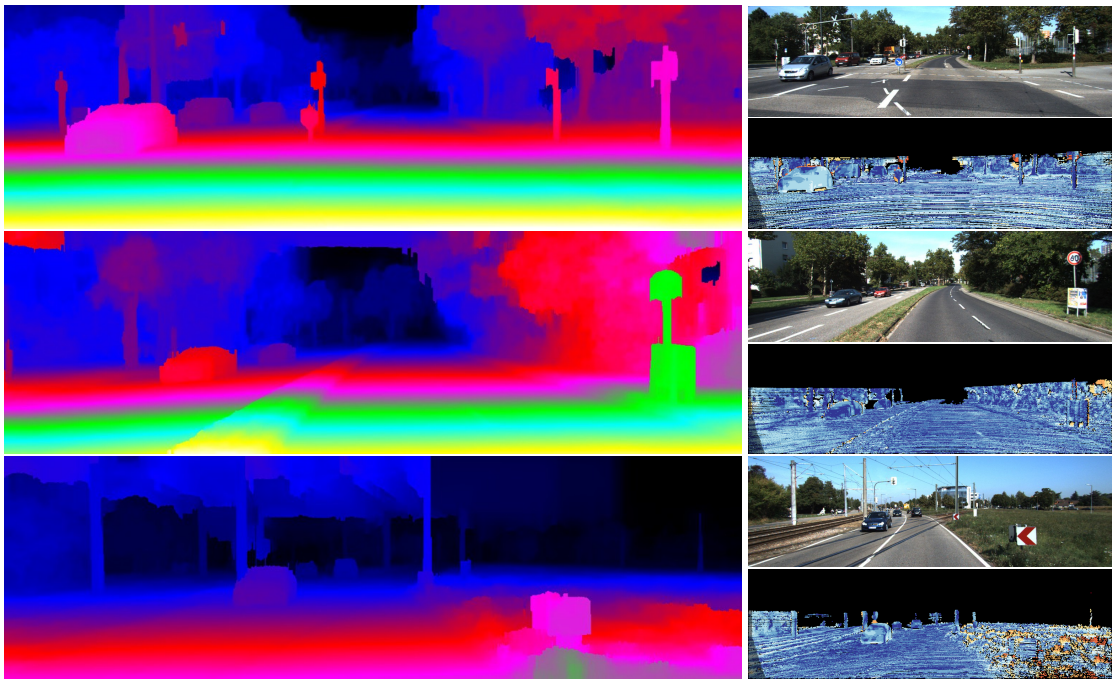


Figure 4.8: Kitti test set examples. The left column shows the color-coded disparity map, the right column shows on top the input image and on the bottom the official error map on the Kitti benchmark. The blue color in the error map indicates correct predictions, orange indicate wrong predictions and black is unknown. Note how our method produces high quality results also for regions where no ground-truth is available, *i.e.* in the upper third of the images.

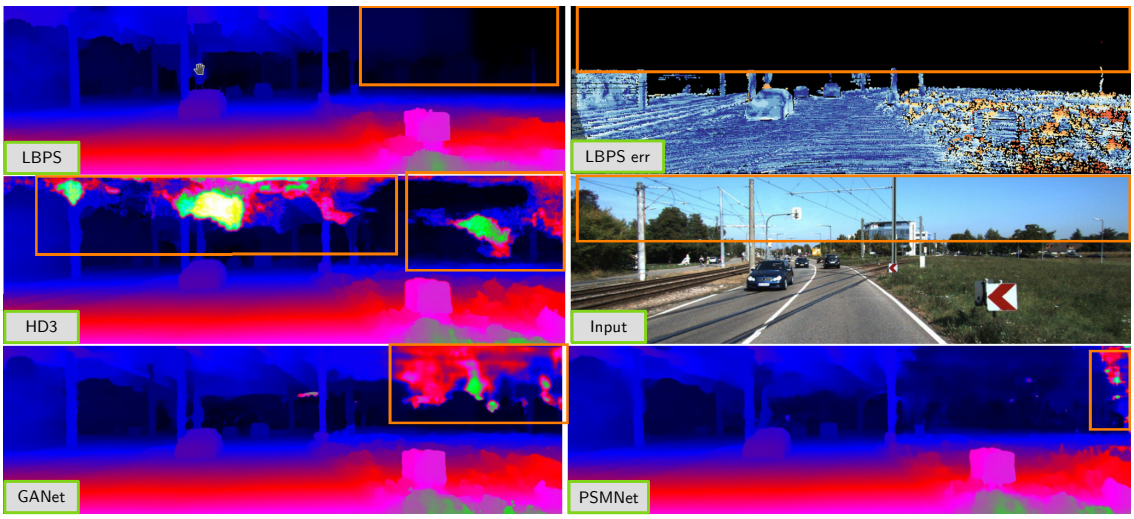


Figure 4.9: Comparison with other methods on the Kitti benchmark. Top row: LBPS (ours), LBPS error visualization. Middle row: HD3 Stereo [213], input image. Bottom row: GANet [221], PSMNet [28]. One can observe that LBPS shows no artifacting in regions where no ground truth is present.

respective benchmarks.

In Fig. 4.9 we compare our prediction with the prediction of current state-of-the-art models. While GA-Net [221], HD3-Stereo [213] and PSM-Net [28] predict precise disparity maps for pixels with available ground-truth, they often hallucinate incorrect disparities on the other pixels. In contrast, our method does not seem to be affected at all by this problem and thus this indicates that our model generalizes very well also to previously unseen structures. For a better comparison we highlighted these regions in Fig. 4.9.

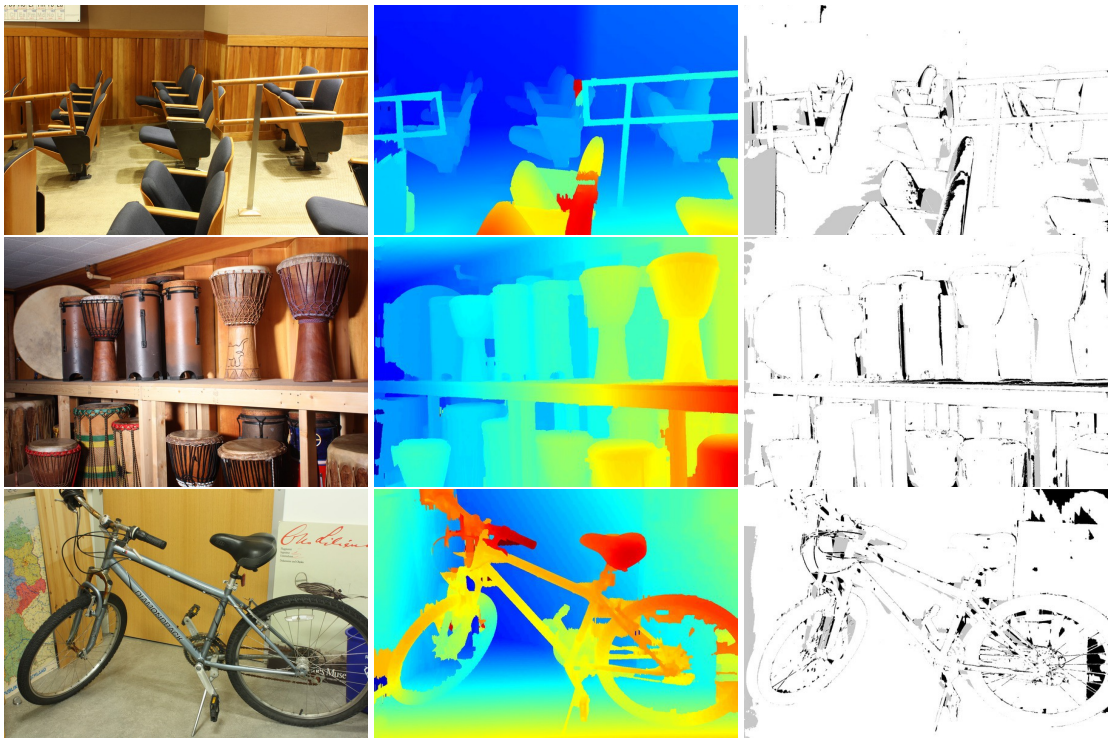


Figure 4.10: Qualitative results on the Middlebury 2014 test set. Left: color coded disparity map, right error map, where white = correct disparity, black = wrong disparity and gray = occluded area.

4.8.4 Optical Flow

Here we show the applicability of our BP-Layer to the optical flow problem. We use the FlyingChairs2 dataset [46, 74] for pre-training our model and fine-tune then with the Sintel dataset [25]. We use the same network architectures for optical flow as for stereo. Thus, we have two feature nets Table 4.1 and then apply hierarchically our BP-Layer on the cost-volumes. In the optical flow setting we set the search-window-size to 109×109 in the finest resolution.

Model	#P[M]	time	bad2	EPE
WTA	**0.13**	**0.27**	4.46 (5.67)	1.25 (1.65)
BP+MS (CE)	0.34	0.44	2.56 (3.46)	0.83 (0.94)
BP+MS (H)	0.34	0.44	2.24 (3.19)	0.66 (0.79)
BP+MS+Ref (H)	0.56	0.49	**2.06 (2.64)**	**0.63 (0.72)**

Table 4.5: Ablation Study on the Sintel Validation set.

Figure 4.11: Left: Qualitative optical flow results on the Sintel validation set. Right: Visualization of the endpoint error, where white=correct and darker pixels are erroneous.

Figure 4.12: Qualitative ablation study for optical flow. The WTA result clearly shows occluded regions (the noisy regions), while our model is able to successfully inpaint these regions. Note that the details increase from top to bottom.

We compute the 109^2 similarities per pixel without storing them and compute the two cost-volumes q^1 and q^2 using Eq. (4.20) on the fly. Fig. 4.11 shows qualitative results and Table 4.5 shows the ablation study on the validation set of the Sintel dataset.

We use only scenes where the flow is not larger than our search-window in this study. We compare the endpoint-error (EPE) and the bad2 error on the EPE. The results show that our BP-Layer can be directly used for optical flow computation and that the BP-Layer is an important building block to boost performance.

We show in Fig. 4.12 more examples on our validation set and highlight differences until we get our final model BP+MS+Ref (H). If we compare the models we see that the quality of the results increase from top to bottom. Thus, the components we add are also beneficial for optical flow. If we add our BP-Layer and use it to regularize the WTA result we can clearly see that most of the noise, mainly coming from occlusions, is gone. The Huber loss function and the refinement successfully predict then contiguous solutions. Although our approach is very simplistic in comparison with current state-of-the-art models we are still able to compute high quality optical flow.

4.8.5 Semantic Segmentation

We apply the BP-Layer also to semantic segmentation to demonstrate its general applicability. In Table 4.6 we show results with our model variants described in Section 4.5.3 using the same CNN block as ESPNet [128], evaluated on the Cityscapes [40] dataset. All model variants using the BP-Layer improve on ESPNet [128] in both the class mean intersection over union (mIOU) and the category mIOU. The best model is, as expected, the jointly trained pixel-wise model referred to as *LBPSS joint*. We have submitted this model to the Cityscapes benchmark. Table 4.7 shows the results on the test set and we can see that we outperform the baseline. Figure 4.13 shows that the BP-Layer refines the prediction by aligning the semantic boundaries to actual object boundaries in the image. Due to the long range interaction, the BP-Layer is also able to correct large incorrect regions such as on *e.g.* the road. One of the advantages of our model is that the learned parameters can be interpreted. Fig. 4.13 shows the learned non-symmetric score matrix, which allows to learn different scores for *e.g.* person → car and car → person. The upper and lower triangular matrix represent pairwise scores when jumping upwards and downwards in the image, respectively. We can read from the matrix that, *e.g.*, an upward jump from sky to road is not allowed. This confirms the intuition, since the road never occurs above the sky. Our model has thus automatically learned appropriate semantic relations which have been hand-crafted in prior work such as *e.g.* [55].

Additional Experiments We show here additional evaluation metrics provided by the Cityscapes benchmark. In Table 4.8, we show the category mIOU score for each invidual category. It can be observed, that the BP-Layer improves this metric for every category and thus the average score for all categories is also improved. The BP-layer also improves the average class mIOU, as seen in Table 4.8. For this metric, the BP-layer improves the results for most classes. However, the mIOU is slightly decreased for the classes truck, train and motorcycle. This is due to

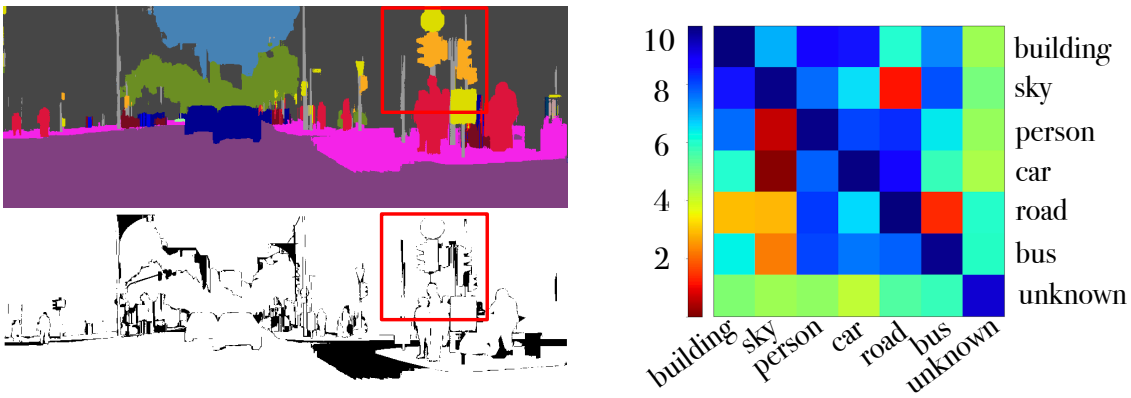


Figure 4.13: *Top Left*: Semantic segmentation result with the BP-Layer. *Bottom Left*: Corresponding error where black = incorrect, white = correct. The red square highlights the region where fine details were accurately reconstructed. *Right*: Visualization of learned vertical pairwise scores.

Method	pw	mIOU	CatmIOU	#P	time
ESPNet [128]	-	61.4	82.2	**0.36**	**0.01**
LBPSS	-	62.8	83.0	0.37	0.11
LBPSS	✓	63.6	83.7	0.73	0.90
LBPSS joint	✓	**65.2**	**84.7**	0.73	0.90

Table 4.6: Ablation study on the Cityscapes validation set. *"pw"* = pixel-wise, inhomogeneous scores.

Method	pw	mIOU	CatmIOU	#P	time
ESPNet [128]	-	60.34	82.18	**0.36**	**0.01**
LBPSS joint	✓	**61.00**	**84.31**	0.73	0.90

Table 4.7: Benchmark results on the Cityscapes [40] test set.

Method	avg	flat	nature	object	sky	construction	human	vehicle
ESPNet [128]	82.18	95.49	89.46	52.94	92.47	86.67	69.76	88.45
LBPSS pixel-wise joint	**84.31**	**97.90**	**90.01**	**58.89**	**93.10**	**88.08**	**72.79**	**89.43**

Method	avg	road	side.	build.	wall	fen.	pole	tr. light	tr. sign	veg.	terr.	sky	person	rider	car	truck	bus	train	motorc.	bic.
ESPNet [128]	60.34	95.68	73.29	86.60	**32.79**	36.43	47.06	46.92	55.41	89.83	**65.96**	92.47	68.48	**45.84**	89.90	**40.00**	47.73	**40.70**	**36.40**	54.89
LBPSS pw joint	**61.00**	**97.00**	**76.88**	**87.38**	31.29	**37.99**	**53.60**	**53.84**	**60.85**	**90.41**	65.85	**93.10**	**70.34**	43.27	**90.93**	31.59	**50.32**	33.93	31.77	**58.67**

Table 4.8: Benchmark results for categories on the Cityscapes [40] test set

the fact that a confusion between these classes in the result from ESPNet [128] can be propagated by the BP-Layer leading to larger patches of incorrect semantic labels. Figure 4.14 shows a visual ablation study of the different models for semantic segmentation. It can be seen that all of the models utilizing the BP-Layer are able to regularize over inconsistencies in the original result

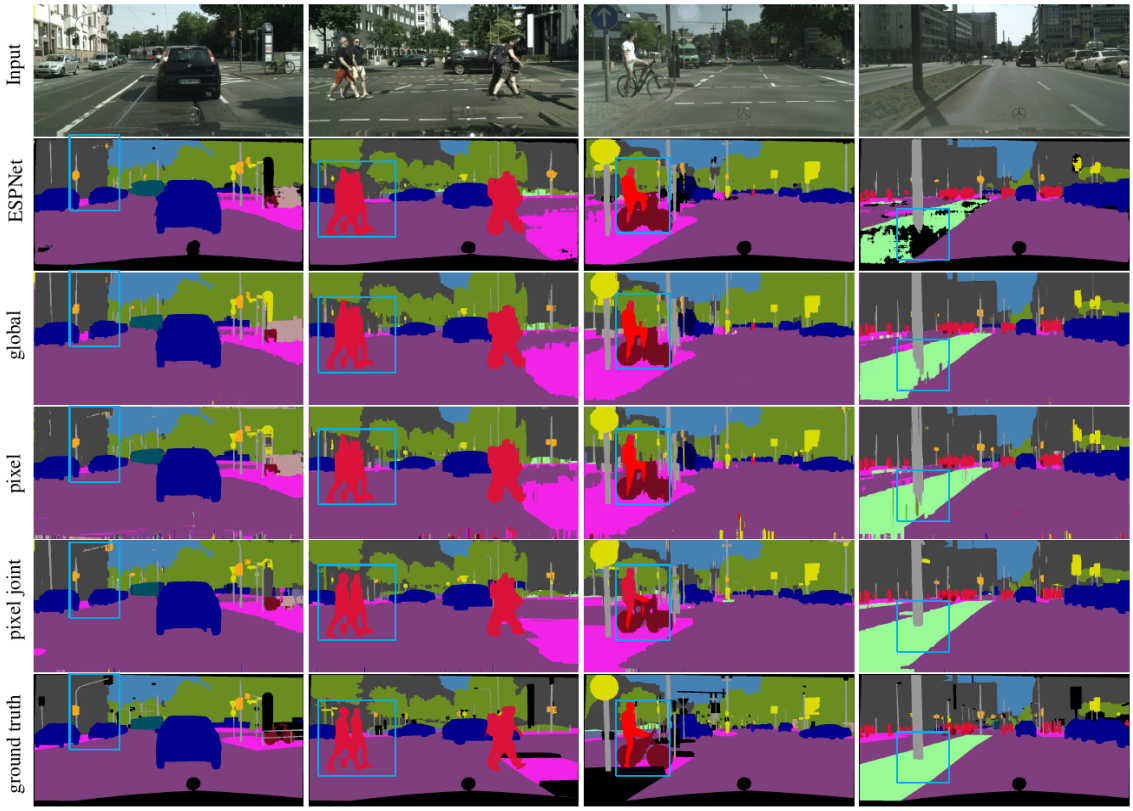


Figure 4.14: Visual ablation study for semantic segmentation on the Cityscapes [40] validation set. The results in the first column show that the BP-Layer can recover fine details such as the thin structures of the traffic light. In the second column one can observe that the legs and heads of the pedestrians are recovered and do not appear as a single blob-like structure. This can also be seen when looking at the bike in the third column. The fourt column shows that the BP-Layer can regularize over inconsistencies in the initial estimation from ESPNet [128] as seen on the sidewalk.

from ESPNet [128]. Furthermore, the pixel wise models are able to better preserve fine structures like traffic lights. If we use the BP-Layer without jointly training the ESPNet, we get some line artifacts in the global and pixel results. These artefacts are easily removed by jointly training both networks as seen in the pixel joint result.

In Figure 4.15, we show qualitative results from the LBPSS pixel joint model on the test set of Cityscapes [40]. It can be seen that the detail on the boundaries of the segmentation masks for scene elements such as cars and pedestrians is preserved, as transition scores are predicted from the input image. We can also show the full vertical transition score matrix for all classes, which we do in Figure 4.16. As described in the paper, the matrix is not symmetric which allows for different scores when transitioning upwards and downwards. If we investigate this matrix in more detail, we are actually able to interpret the learned results. An interesting observation can *e.g.* be seen when looking at the column for the sky class. It encodes that downward label

Figure 4.15: Qualitative results for semantic segmentation on the Cityscapes [40] test set. Our model is able to precisely capture object boundaries around *e.g.* pedestrians and cars.

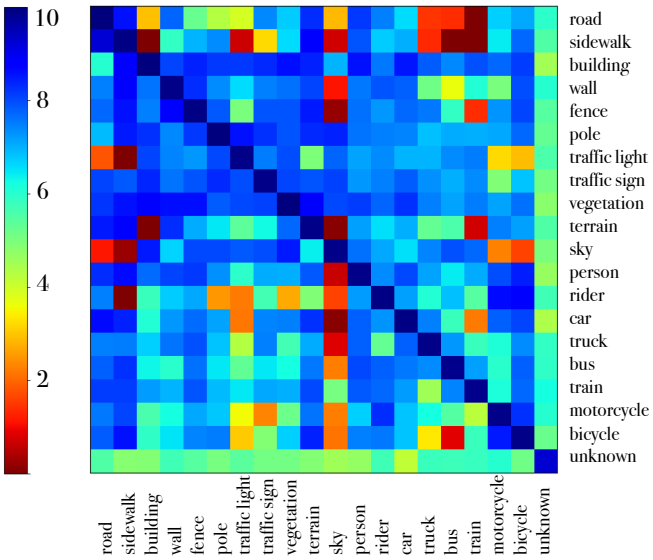


Figure 4.16: Vertical transition score matrix for all classes, where the upper triangular matrix encodes upwards transitions and the lower triangular matrix encodes downwards transitions.

transitions from car, truck or train to sky are very expensive and upwards transitions from *e.g.* car to sky are comparably cheap. This is very intuitive and encodes that the sky is always above the car and not below. Another example is that traffic lights and vegetation are often surrounded by sky and thus these scores are higher. Also the scores for the unknown class very intuitive. The very similar scores to all other classes can be interpreted as a uniform distribution. This makes

totally sense, because the class “unknown” has interactions with all other classes.

4.9 Conclusion

We have proposed a novel combination of CNN and CRF techniques, aiming to resolve practical challenges. We took one of the simplest inference schemes, showed how to compute its backprop and connected it with the marginal losses. The following design choices were important for achieving a high practical utility: using max-product for fast computation and backprop of approximate marginals, propagating the information over a long range with sequential subproblems; training end-to-end without approximations; coarse-to-fine processing at several resolution levels; context-dependent learnable unary and pairwise costs. We demonstrated the model can be applied to three dense prediction problems and gives robust solutions with more efficient parameter complexity and time budget than comparable CNNs. In particular in stereo and flow, the model performs strong regularization in occluded regions and this regularization mechanism is interpretable in terms of robust fitting with jump scores.

5

Learned Continuous Disparity Refinement

Back in 2018, we had the powerful hybrid Convolutional Neural Network (CNN) + Conditional Random Field (CRF) model [87]. As shown in the previous sections, we have used a discrete optimization algorithm to perform inference in our CRF. While the CRF is very powerful and efficient in integrating spatial constraints, the output is however always discrete. This is a limitation in the geometric stereo task where a continuous value is more natural and desirable. We will address exactly this problem in the work presented in this chapter. We therefore propose a continuous, hierarchical variational network to refine *disparity maps. The proposed method is specifically designed as a refinement module and can therefore be applied on top of any stereo method.*

This work was presented at GCPR 2019 in Dortmund and was awarded with the Best Paper Award.

Contents

5.1 Introduction **136**
5.2 Related Work **137**
5.3 Method **139**
5.4 Computing Inputs **143**
5.5 Learning **145**
5.6 Experiments **146**
5.7 Analyzing the VN **153**
5.8 Conclusion & Future Work **160**

5.1 Introduction

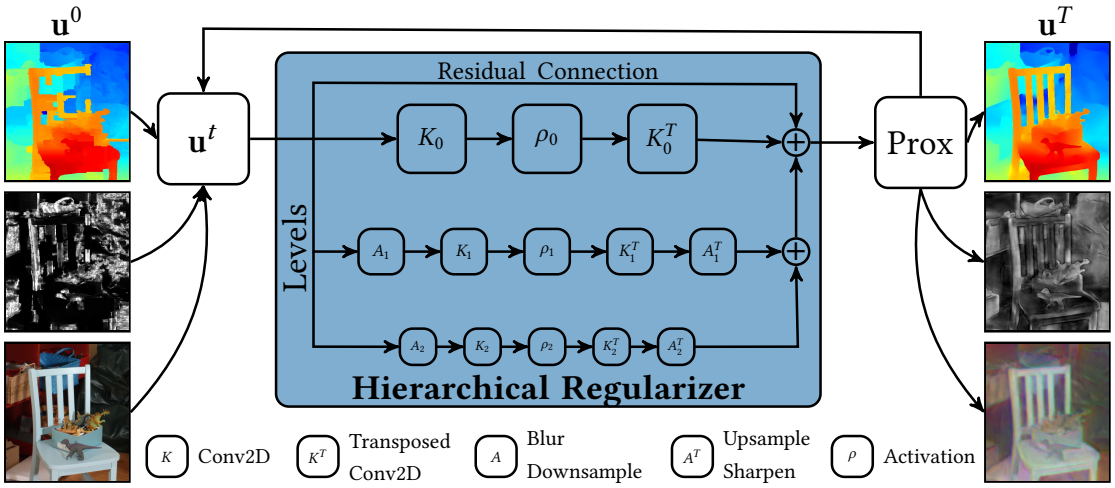


Figure 5.1: Model Overview. Our model takes three inputs, an initial disparity map, confidence map and the color image. The collaborative hierarchical regularizer iteratively computes a refined disparity map and yields refined confidences and a color image holding the main depth discontinuities. The subscripts indicate the hierarchical level of the image pyramid.

Computing 3D information from a stereo image pair is one of the most important problems in computer vision. One reason for this is that depth information is a very strong cue to understanding visual scenes, and depth information is therefore an integral part of many vision based systems. For example, in autonomous driving, it is not sufficient to identify the objects visible in the scene semantically, but the distance to the objects is also very important. A lidar scanner can be used for distance estimates, but is often too expensive and provides only sparse depth estimates. Therefore, the primary approach is to compute depth information only from stereo images. However, due to reflections, occlusions, difficult illuminations *etc.*, the calculation of depth information from images is still a very challenging task. To tackle these difficulties the computation of dense depth maps is usually split up into the four steps (i) matching cost computation, (ii) cost aggregation, (iii) disparity computation and (iv) disparity refinement [171]. In deep learning based approaches (i) and (ii) are usually implemented in a matching convolutional neural network (CNN), (iii) is done using graphical models or 3D regularization CNNs and (iv) is done with a refinement module [194].

There are many approaches to tackle (i)-(iii). However, there are only a few learning-based works for disparity refinement (iv) (see Section 5.2). Existing work to refine the disparity maps is often based on black-box CNNs to learn a residuum from an initial disparity map to a refined disparity map. In this work we want to overcome these black-box refinement networks with a simple, effective and most important easily interpretable refinement approach for disparity maps.

We tackle the refinement problem with a learnable hierarchical variational network. This allows us to exploit both the power of deep learning and the interpretability of variational methods. In order to show the effectiveness of the proposed refinement module, we conduct experiments on directly refining/denoising winner-takes-all (WTA) solutions of feature matching and as a pure post-processing module on top of an existing stereo method.

Fig. 5.1 shows an overview of our method. The inputs to our method are an initial disparity map, a pixel-wise confidence map and the corresponding RGB color image. These three inputs span a collaborative space in which our hierarchical regularizer iteratively refines the initial inputs. Finally, the output of the hierarchical regularizer is the refined disparity map, a refined confidence map and a refined color image. Note that the refined (= output) color and confidence image are a byproduct of the refinement process.

Contributions We propose a learnable variational refinement network which takes advantage of the joint information of the color image, the disparity map and a confidence map to compute a refined disparity map. Our proposed method can be derived from the iterates of a proximal gradient method specifically designed for stereo refinement. Additionally, we evaluate a broad range of possible architectural choices in an ablation study. We demonstrate the interpretability of our model by visualizing the intermediate iterates and showing the learned filters as well as the learned activation functions. We show the effectiveness of our method by participating on the two complementary publicly available benchmarks Middlebury 2014 and Kitti 2015.

This paper extends the conference paper [84], where we additionally study i) a model with shared parameters over the iterations, ii) a comparison with the recent lightweight StereoNet refinement module [80] and iii) a new section, where we analyze the VN. To this end, we show how to compute eigen disparity maps that reveal structural properties of the learned regularizer and analyze the refined confidences in order to show the increased reliability of the confidences predicted by our model.

5.2 Related Work

We propose a learnable model using the modeling power of variational calculus to explicitly guide the refinement process for stereo. This combination of learning and classical optimization for stereo refinement allows us to group the related work into the three categories (i) variational methods, (ii) disparity refinement and (iii) learnable optimization schemes. We review the most related works of these categories in the following paragraphs.

Variational Methods Variational methods formulate the dense correspondence problem as minimization of an energy functional comprising a data fidelity term and a smoothness term. We use here the term *correspondence problem* to indicate that the following methods can in general be used for both optical flow and stereo, because stereo can be considered as optical flow in horizontal direction only. The data-term usually measures the raw intensity difference [24, 27, 217] between the reference view and the warped other view. The regularizer imposes prior knowledge on the

resulting disparity map. This is, the disparity map is assumed to be piecewise smooth. Prominent regularizers are the robust Total Variation (TV) [217] and the higher order generalization of TV as e.g. used by Ranftl *et al.* [158, 160] or by Kuschk and Cremers [101]. Variational approaches have two important advantages in the context of stereo. They naturally produce sub-pixel accurate disparities and they are easily interpretable. In order to capture large displacements as well, a coarse-to-fine warping scheme [24] is necessary. To overcome the warping scheme without losing fine details, variational methods can also be used to refine an initial disparity map. This has *e.g.* be done by Shekhovtsov *et al.* [178] who refined the initial disparity estimates coming from a Conditional Random Field (CRF). Similarly, [161] and [124] used a variational method for refining optical flow.

Disparity Refinement Here we want to focus on the refinement of an initial disparity map. The initial disparity map can be *e.g.* the WTA solution of a matching volume or any other output of a stereo algorithm. One important approach of refinement algorithms is the fast bilateral solver (FBS) [4]. This algorithm refines the initial disparity estimate by solving an optimization problem containing an ℓ_2 smoothness- and an ℓ_2 data-fidelity term. The fast bilateral solver is the most related work to ours. However, in this work we replace the ℓ_2 norm with the robust ℓ_1 norm. More importantly, we additionally replace the hand-crafted smoothness term by a learnable multi-scale regularizer. Another refinement method was proposed by Gidaris and Komodakis [60]. They also start with an initial disparity map, detect erroneous regions and then replace and refine these regions to get a high-quality output. Pang *et al.* [147] proposed to apply one and the same network twice. They compute the initial disparity map in a first pass, warp the second view with the initial disparity map and then compute only the residual to obtain a high quality disparity map. Liang *et al.* [112] also improved the results by adding a refinement sub-network on top of the regularization network. We want to stress that the CNN based refinement networks [112, 147] do not have a specialized architecture for refinement as opposed to the proposed model. Khamis *et al.* [80] also focused on the refinement of coarse initial disparity maps in a hierarchical setting. They explicitly construct a light-weight network which is used to compute a residuum between the initial disparity map and the refined map. [80] therefore uses only standard CNN building blocks with explicitly modeled residual connections. In difference, our method naturally provides the residual connections and we gain control and interpretability of the refinement process through our specialized, optimization based architecture. We show a direct comparison between both methods in the experiments and it will turn out that our approach is actually beneficial in interpretability and performance.

Learnable Optimization Schemes Learnable optimization schemes are based on unrolling the iterates of optimization algorithms. We divide the approaches into two categories. In the first category the optimization iterates are mainly used to utilize the structure during learning. For example in [162] 10 iterations of a TGV regularized variational method are unrolled and used for depth super-resolution. However, they learn only the step-sizes for the algorithm and keep the algorithm fixed. Similarly, in [201] unrolling 10k iterations of the FISTA [6] algorithm is proposed.

The second category includes methods where the optimization scheme is not only used to provide the structure, but it is also generalized by adding additional learnable parameters directly to the optimization iterates. For example [199] proposed a primal-dual-network for low-level vision problems, where the authors learned the inference part of a Markov Random Field (MRF) model generalizing a primal-dual algorithm. Chen *et al.* [32] generalized a reaction-diffusion model and successfully learned a model for image denoising. Based on [32] a generalized incremental proximal gradient method was proposed in [89], where the authors showed connections to residual units [66]. Wang [204] proposed proximal deep structured models where the authors perform inference with their recurrent network. Meinhardt *et al.* [129] learned proximal operators using denoising networks for regularization. We built on the work of Chen *et al.*, but specially designed the energy terms for the stereo task. Additionally, we allow to regularize on multiple spatial resolutions jointly and make use of the robust ℓ_1 function in our data-terms.

5.3 Method

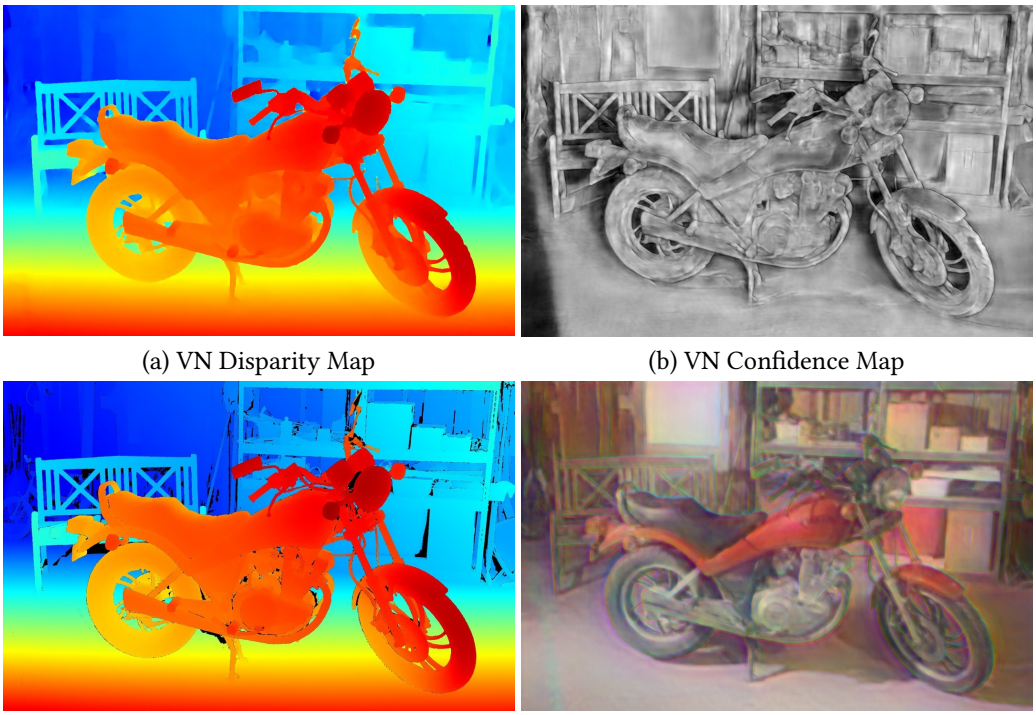
We consider images to be functions $f : \Omega \to \mathbb{R}^C$, with $\Omega \subset \mathbb{N}_+^2$ and C is the number of channels which is 3 for RGB color images. Given two images f^0 and f^1 from a rectified stereo pair, we want to compute dense disparities d such that $f^0(x) = f^1(x - \tilde{d})$, *i.e.* we want to compute the horizontal shift $\tilde{d} = (d, 0)$ for each pixel $x = (x_1, x_2)$ between the reference image f^0 and the second image f^1. Here, we propose a novel variational refinement network for stereo which operates solely in 2D image space and is thus very efficient. The input to our method is an initial disparity map $\check{u} : \Omega \to [0, D]$, where D is the maximal disparity, a reference image f^0 and a pixel-wise confidence map $c : \Omega \to [0, 1]$. We explain the computation of the initial disparity- and confidence map in detail in Section 5.4. Right now, we just assume we have given the inputs.

The proposed variational network is a method to regularize, denoise and refine a noisy disparity map with learnable filters and learnable potential functions. Hence, the task we want to solve is the following: Given a noisy disparity map $\check{u}$, we want to recover the clean disparity with T learnable variational network steps. We do not make any assumptions on the quality of the initial disparity map, i.e. the initial disparity map may contain many strong outliers.

Collaborative Disparity Denoising

As the main contribution of this paper, we propose a method that performs a collaborative denoising in the joint color image, disparity and confidence space (see Fig. 5.2). Our model is based on the following three observations: (i) Depth discontinuities coincide with object boundaries, because we use the left image as the reference image (ii) discontinuities in the confidence image are expected to be close to left-sided object boundaries and (iii) the confidence image can be used as a pixel-wise weighting factor in the data fidelity term. Based on these three observations, we propose the following collaborative variational denoising model

$$\min_{\mathbf{u}} \mathcal{R}(\mathbf{u}) + \mathcal{D}(\mathbf{u}), \tag{5.1}$$



(a) VN Disparity Map (b) VN Confidence Map

(c) Ground-truth (d) VN Color Image

Figure 5.2: Collaborative Disparity Denoising. Our method produces three outputs: (a) the refined disparity map, (b) the refined confidence map and (d) the refined color image. (c) shows the ground-truth image for comparison (black pixels = invalid). Note how our method is able to preserve fine details such as the spokes of the motorcycle.

where $\mathbf{u} = (\mathbf{u}^{rgb}, u^d, u^c) : \Omega \rightarrow \mathbb{R}^5$, *i.e.* $\mathbf{u}$ contains for every pixel an RGB color value, a disparity value and a confidence value. $\mathcal{R}(\mathbf{u})$ denotes the collaborative regularizer and it is given by a multi-scale and multi-channel version of the Fields of Experts (FoE) model [165] with L scales and K channels.

$$\mathcal{R}(\mathbf{u};\theta) = \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{x \in \Omega} \phi_k^l \left(\left(K_k^l A^l \mathbf{u} \right)(x) \right), \tag{5.2}$$

where $A^l : \mathbb{R}^5 \mapsto \mathbb{R}^5$ are combined blur and downsampling operators, $K_k^l : \mathbb{R}^5 \mapsto \mathbb{R}$ are linear convolution operators and $\phi_k^l : \mathbb{R} \mapsto \mathbb{R}$ are non-linear potential functions. The vector θ holds the parameters of the regularizer which will be detailed later. Note that multiple levels allow the model to operate on different spatial resolutions and therefore enables the denoising of large corrupted areas. Intuitively, the collaborative regularizer captures the statistics of the joint color, confidence and disparity space. Hence, it will be necessary to learn the linear operators and the non-linear potential functions from data. It will turn out that the combination of filtering in the joint color-disparity-confidence space at multiple hierarchical pyramid levels and specifically learned channel-wise potential functions make our model powerful.

$\mathcal{D}(\mathbf{u})$ denotes the collaborative data fidelity term and it is defined by

$$\mathcal{D}(\mathbf{u};\theta) = \frac{\lambda}{2}\|\mathbf{u}^{rgb} - \mathbf{f}^0\|^2 + \mu\|u^c - c\|_1 + \nu\|u^d - \check{d}\|_{u^c,1}, \tag{5.3}$$

where θ is again a placeholder for the learnable parameters. The first term ensures that the smoothed color image $\mathbf{u}^{rgb}$ does not deviate too much from the original color image $\mathbf{f}^0$. We use here a quadratic ℓ_2 term, because we do not assume any strong outliers in the color image. The second term ensures that the smoothed confidence map stays close to the original confidence map. Here we use an ℓ_1 norm in order to deal with outliers in the initial confidence map. The last term is the data fidelity term of the disparity map. It is given by an ℓ_1 norm which is pixel-wise weighted by the confidence measure u^c, *i.e.*

$$\|r\|_{w,1} = \sum_{i=1}^{N} w_i |r_i|, \tag{5.4}$$

where $r, w \in \mathbb{R}^N$. Hence, data fidelity is enforced in high-confidence regions and suppressed in low-confidence regions. Note that the weighted ℓ_1 norm additionally ties the disparity map with the confidence map during the steps of the variational network.

Proximal Gradient Method (PGM) We consider a PGM [149] whose iterates are given by

$$\mathbf{u}_{t+1} = \operatorname{prox}_{\alpha_t \mathcal{D}}(\mathbf{u}_t - \alpha_t \nabla \mathcal{R}(\mathbf{u}_t)), \tag{5.5}$$

where α_t is the step-size, $\nabla\mathcal{R}(\mathbf{u}_t)$ is the gradient of the regularizer which is given by

$$\nabla\mathcal{R}(\mathbf{u}) = \sum_{l=1}^{L}\sum_{k=1}^{K} (K_k^l A^l)^T \rho_k^l \left(K_k^l A^l \mathbf{u}\right), \tag{5.6}$$

where $\rho_k^l = \operatorname{diag}((\phi_k^l)')$. Hence, ρ_k^l is the derivative of the potential function and can be interpreted as the activation function in our regularizer. A visual comparison between potential and activation-functions is shown in Fig. 5.10. $\operatorname{prox}_{\alpha_t \mathcal{D}}$ denotes the proximal operator with respect to the data fidelity term, which is defined by

$$\operatorname{prox}_{\alpha_t \mathcal{D}}(\tilde{\mathbf{u}}) = \arg\min_{\mathbf{u}} \mathcal{D}(\mathbf{u}) + \frac{1}{2\alpha_t}\|\mathbf{u} - \tilde{\mathbf{u}}\|_2^2. \tag{5.7}$$

Note that the proximal map allows to handle the non-smooth data fidelity terms such as the ℓ_1 norm. Additionally, there is a strong link between proximal gradient methods and residual units which allows to incrementally reconstruct a solution (see Fig. 5.1).

Proximal Operators for the Data Terms The proximal operator in Eq. (5.7) is an optimization problem itself. We need to compute the proximal operator for the ℓ_1 and the ℓ_2 function. Both can

be computed in closed form. Therefore, let us consider the proximal operator of a function f:

$$\text{prox}_{\tau f}(\tilde{u}) = \arg\min_u f(u) + \frac{1}{2\tau}\|u - \tilde{u}\|^2. \tag{5.8}$$

First, we present the result of the proximal operator for the ℓ_2 function

$$f(u) = \frac{\lambda}{2}\|u - u_0\|^2. \tag{5.9}$$

Inserting Eq. (5.9) into Eq. (5.8) and setting the derivative w.r.t. u to zero, we can compute the optimal solution u^* with

$$u^* = \frac{\tilde{u} + \tau\lambda u_0}{1 + \tau\lambda}, \tag{5.10}$$

where for the color image data term, $u_0 = I_0$ and $\tilde{u} = u^{rgb}$.

Similarly, we compute the proximal operator of the weighted ℓ_1 function

$$f(u) = \gamma\|u - u_0\|_{w,1} = \gamma \sum_{x\in\Omega} w(x)|u(x) - u_0(x)|. \tag{5.11}$$

The absolute function is not differentiable at 0 and therefore the optimality condition requires the sub-differential to contain 0. The closed form solution of the proximal operator Eq. (5.8) with f being the ℓ_1 function as defined in Eq. (5.11) is given by

$$u^* = u_0 + \max(0, |\tilde{u} - u_0| - \tau\gamma w) \cdot \text{sign}(\tilde{u} - u_0). \tag{5.12}$$

Thus, for the disparity data term we set $w = c$ and $u_0 = \check{d}$. Since the confidence u^c is present in the confidence data term, and linearly dependent in the disparity data term, we make use of the identity

$$\text{prox}_{\tau f}(\tilde{u}) = \text{prox}_{\tau g}(\tilde{u} - a) \tag{5.13}$$

for functions $f(u) = g(u) + a^T u + b$. In our setting $g(u) = \mu\|u^c - c\|_1$ is the confidence data-term and $a = |u^d - \check{d}|$.

Variational Network Our collaborative denoising algorithm consists of performing a fixed number of T iterations of the proximal gradient method Eq. (5.5). In order to increase the flexibility we allow the model parameters to change in each iteration.

$$\mathbf{u}_{t+1} = \text{prox}_{\alpha_t \mathcal{D}(\cdot,\theta_t)}(\mathbf{u}_t - \alpha_t \nabla\mathcal{R}(\mathbf{u}_t, \theta_t)),\ 0 \le t \le T-1 \tag{5.14}$$

Following [32, 89] we parametrize the derivatives of the potential functions ρ_k^l in (5.6) using Gaussian radial basis functions (RBF)

$$\rho_k^{l,t}(s) = \beta_k^{l,t} \sum_{b=1}^{B} w_{k,b}^{l,t} \exp\left(-\frac{(s - \gamma_b)^2}{2\sigma^2}\right) \tag{5.15}$$

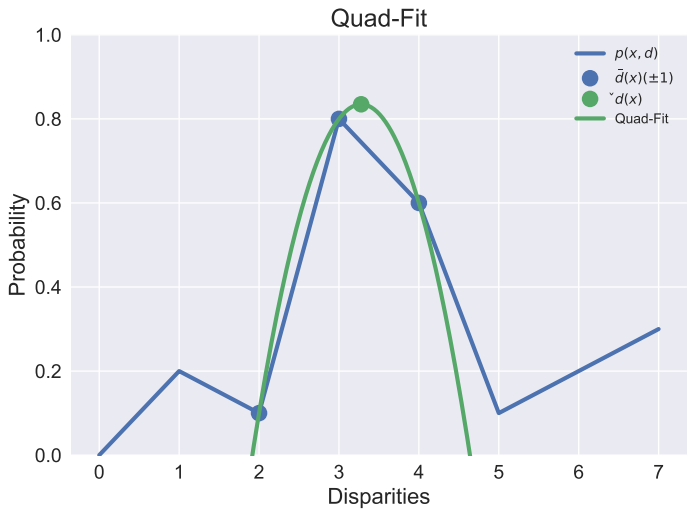


Figure 5.3: Visualization of the quadratic fitting. We select the points next to the maximum value and fit a quadratic function. Computing the extremum of the quadratic functions yields the refined disparity and the refined probability.

to allow learning of appropriate activation functions from the data. We sample the means γ_b regularly on the interval $[-3, 3]$, σ is the standard deviation of the Gaussian kernel and $\beta_k^{l,t}$ is a scaling factor. The linear operators $K_k^{l,t}$ are implemented as multi-channel 2D convolutions with convolution kernels $\kappa_k^{l,t}$. In summary, the parameters in each step are given by $\theta_t = \{\kappa_k^{l,t}, \beta_k^{l,t}, w_{k,b}^{l,t}, \mu^t, \nu^t, \lambda^t, \alpha_t, \}$.

5.4 Computing Inputs

Our proposed refinement method can be applied to any stereo method coming along with a cost-volume, which is the case for the majority of existing stereo methods.

Probability Volume Assume we have given a cost-volume $v : \Omega \times \{0, \dots, D-1\} \to \mathbb{R}$, where smaller costs mean a higher likelihood of the respective disparity values. In order to map the values onto probabilities $p : \Omega \times \{0, \dots, D-1\}$, we make use of the "softmax" function, that is

$$p(x, d) = \frac{\exp(\frac{-v(x,d)}{\eta})}{\sum_{d'=0}^{D-1} \exp(\frac{-v(x,d')}{\eta})}, \tag{5.16}$$

where η influences the smoothness of the probability distribution.

Initial Disparity Map From Eq. (5.16) we can compute the WTA solution by a pixel-wise arg max over the disparity dimension, i.e.,

$$\bar{d}(x) \in \arg\max_{d} \; p(x,d). \tag{5.17}$$

Moreover, we compute a sub-pixel accurate disparity map $\check{d}(x)$ by fitting a quadratic function to the probability volume. This is equivalent to performing one step of Newton's algorithm:

$$\check{d}(x) = \bar{d}(x) - \frac{\delta^+(p(x,\cdot))(\bar{d}(x))}{\delta^-(\delta^+(p(x,\cdot)))(\bar{d}(x))}, \tag{5.18}$$

where $\delta^{\{+,-\}}$ denote standard forward and backward differences in the disparity dimension. Furthermore, we compute the refined value of the probabilities, denoted as $\check{p}(x)$, via linear interpolation in the probability volume.

In the joint training of our feature network and the regularization network we need to backpropagate the gradient through the refined disparities. Therefore, we must compute the gradient of our sub-pixel accurate disparity map w.r.t. the probability volume. The gradient is non-zero only for the supporting points of the quadratic function (shown in blue in Fig. 5.3) and it is given by

$$\frac{\partial \check{d}(x)}{\partial p(x,d)} = \begin{cases} \frac{\delta^c(p(x,\cdot))(\bar{d}(x))}{(\delta^-(\delta^+(p(x,\cdot)))(\bar{d}(x)))^2} & \text{if } d = \bar{d}(x) \\ \frac{\delta^+(p(x,\cdot))(\bar{d}(x))}{(\delta^-(\delta^+(p(x,\cdot)))(\bar{d}(x)))^2} & \text{if } d = \bar{d}(x) - 1 \\ \frac{\delta^-(p(x,\cdot))(\bar{d}(x))}{(\delta^-(\delta^+(p(x,\cdot)))(\bar{d}(x)))^2} & \text{if } d = \bar{d}(x) + 1 \\ 0 & \text{else,} \end{cases} \tag{5.19}$$

where $\delta^{\{+,-,c\}}$ are standard forward-, backward- and central-differences in the disparity dimension. Note, that we overcome the problem of the non-differentiable arg min function with the fitting of the quadratic function. Fig. 5.3 shows a visualization of the quadratic fitting procedure.

Initial Confidence Measure The computation of a confidence measure of the stereo results is important for many applications and a research topic on its own [73]. Here we take advantage of the probabilistic nature of our matching costs $\check{p}(x)$. Moreover, we make use of geometric constraints by using a left-right (LR) consistency check, where the left and right images are interchanged. This allows us to identify occluded regions. We compute the probability of a pixel being not occluded as

$$p_o(x) = \frac{\max(\varepsilon - \mathrm{dist}_{lr}(x), 0)}{\varepsilon} \in [0,1], \tag{5.20}$$

where

$$\mathrm{dist}_{lr}(x) = |\check{d}_l(x) + \check{d}_r(x + \check{d}_l(x))| \tag{5.21}$$

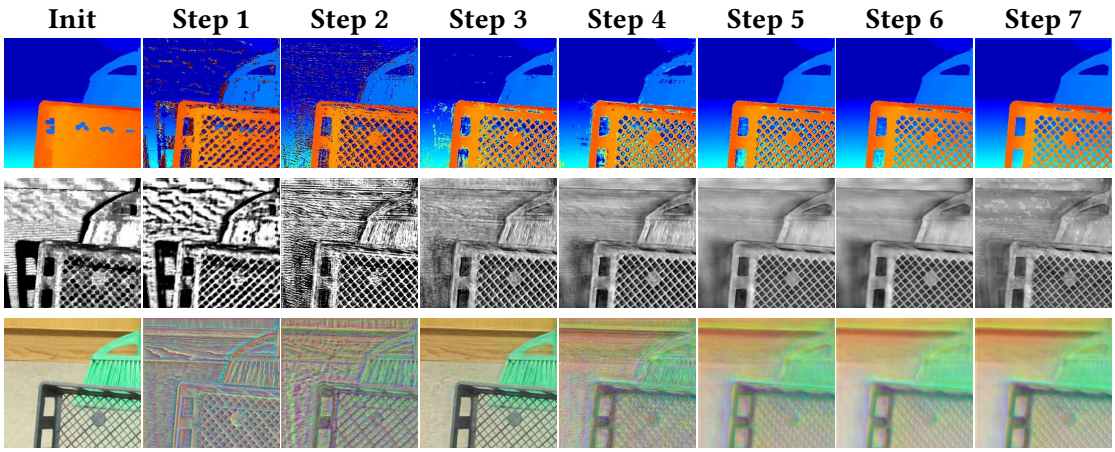


Figure 5.4: Visualization of steps in the VN. Top to bottom: disparity map, confidence map, image. Left to right: Initialization, VN Steps 1 - 7. Note how the color image and the confidence map help to restore very fine details in the disparity map.

is the disparity difference between the left prediction $\check{d}_l$ and the right prediction $\check{d}_r$ and the parameter ε acts as a threshold and is set to $\varepsilon = 3$ in all experiments. The final confidence measure is given by

$$c(x) = \check{p}(x)p_o(x) \in [0,1]. \tag{5.22}$$

Thus, we define our total confidence as the product of the matching confidence and the LR confidence. Most of the pixels not surviving the LR check are pixels in occluded regions. To get a good initialization for these pixels as well, we inpaint the disparities of these pixels from the left side. The experiments show that this significantly increases the performance of the model (see Table 5.2).

5.5 Learning

In this section we describe our learning procedure for the collaborative denoising model. To remove scaling ambiguities we require the filter kernels $\kappa_k^{l,t}$ to be zero-mean and to have an ℓ_2 norm ≤ 1. Moreover, we constrain the weights of the RBF kernels to have an ℓ_2 norm ≤ 1, too. This is defined with the following convex set:

$$\Theta = \{\theta_t : \|\kappa_k^{l,t}\| \leq 1,\ \sum_{j=1}^{J} \kappa_{k,j}^{l,t} = 0,\ \|w_k^{l,t}\| \leq 1\} \tag{5.23}$$

For learning, we define a loss function that measures the error between the last iterate of the disparity map u_T^d and the ground-truth disparity d^*. Note that we do not have a loss function for the confidence and the color image. Their aim is rather to support the disparity map to achieve

the lowest loss. We use a truncated Huber function of the form

$$\min_{\theta\in\Theta} \sum_{s=1}^{S} \sum_{x\in\Omega} \min\left(|u_{s,T}^{d}(x,\theta) - d_s^*(x)|_\delta,\ \tau\right) \tag{5.24}$$

where τ is a truncation value, s denotes the index of the training sample and

$$|r|_\delta = \begin{cases} \frac{r^2}{2\delta} & \text{if } |r| \leq \delta \\ |r| - \frac{\delta}{2} & \text{else} \end{cases} \tag{5.25}$$

is the Huber function.

Implementation Details We implemented our model in the PyTorch machine learning framework[1]. We train the refinement module for 3000 epochs with a learning rate of 10^{-3} with a modified projected Adam optimizer [81]. While in [81] the stepsize is adjusted element-wise, we use a constant stepsize within each parameter block. This is necessary to ensure an orthogonal projection of the parameter blocks onto the constraint set Θ. After 1500 epochs we reduce the truncation value τ from ∞ to 3.

5.6 Experiments

We split the experiments into two parts. In the first part we evaluate architectural choices based on the WTA result of a matching network and compare with the Fast Bilateral Solver (FBS) [4] and the StereoNet (SN) refinement method of [80]. In the second part, we use the best architecture and train a variational network for refining the disparity maps computed by the CNN-CRF method [85]. We use this method to participate in the publicly available stereo benchmarks Middlebury 2014 and Kitti 2015. To ensure a fair comparison we choose methods with similar numbers of parameters and runtimes. Fig. 5.4 shows how our method constructs the final result. The method recovers step-by-step fine details with the guidance of the confidences and the color image. Qualitative results on the official tests sets of Middlebury and Kitti are visualized in Figs. 5.5 and 5.6 and additional qualitative results are shown in Figs. 5.7 and 5.8.

Kitti 2015 The Kitti 2015 dataset [132] is an outdoor dataset specifically designed for autonomous driving. It contains 200 images with available ground-truth to train a model and 200 images with withheld ground-truth which is used for testing the models on previously unseen data. The ground-truth is captured using a laser scanner and is therefore sparse in general. The cars are densified by fitting CAD models into the laser point-cloud. We report the *badX* error metric for occluded (occ) and non-occluded (noc) pixels with $X = 3$. In the badX measure the predicted disparity $\hat{d}$ is treated incorrect, if the distance to the ground-truth disparity d^* is larger than X.

[1] https://pytorch.org

Layer	KS	Resolution	Channels	Input
conv00	3	$W \times H$ / $W \times H$	3 / 64	Image
conv01	3	$W \times H$ / $W \times H$	64 / 64	conv00
pool0	2	$W \times H$ / $\frac{W}{2} \times \frac{H}{2}$	64 / 64	conv01
conv10	3	$\frac{W}{2} \times \frac{H}{2}$ / $\frac{W}{2} \times \frac{H}{2}$	64 / 64	pool0
conv11	3	$\frac{W}{2} \times \frac{H}{2}$ / $\frac{W}{2} \times \frac{H}{2}$	64 / 64	conv10
pool1	2	$\frac{W}{2} \times \frac{H}{2}$ / $\frac{W}{4} \times \frac{H}{4}$	64 / 64	conv10
conv20	3	$\frac{W}{4} \times \frac{H}{4}$ / $\frac{W}{4} \times \frac{H}{4}$	64 / 64	pool1
conv21	3	$\frac{W}{4} \times \frac{H}{4}$ / $\frac{W}{4} \times \frac{H}{4}$	64 / 64	conv20
deconv1	3	$\frac{W}{4} \times \frac{H}{4}$ / $\frac{W}{2} \times \frac{H}{2}$	64 / 64	conv21
conv12	3	$\frac{W}{2} \times \frac{H}{2}$ / $\frac{W}{2} \times \frac{H}{2}$	128 / 64	{deconv1, conv11}
conv13	3	$\frac{W}{2} \times \frac{H}{2}$ / $\frac{W}{2} \times \frac{H}{2}$	64 / 64	conv12
deconv0	3	$\frac{W}{2} \times \frac{H}{2}$ / $W \times H$	64 / 64	conv12
conv02	3	$W \times H$ / $W \times H$	128 / 64	{deconv0, conv01}
conv03	3	$W \times H$ / $W \times H$	64 / 64	conv02

Table 5.1: Detailed architecture of our multi-level feature network. *KS* denotes the kernel size, *Resolution* contains the spatial resolution of the input and output, respectively and *Channels* contain the number of input and output feature channels, respectively. We use curly brackets to indicate a concatenation of feature maps. We use the LeakyReLU activation function after every convolution layer.

Middlebury 2014 The Middlebury 2014 stereo dataset [169] is orthogonal to the Kitti 2015 dataset. It consists of 153 high resolution indoor images with highly precise dense ground-truth. The challenges in the Middlebury dataset are large, almost untextured regions, huge occluded regions, reflections and difficult lighting conditions. The generalization capability of the method is evaluated on a 15 image test-set with withheld ground-truth data. We report all available metrics, i.e., bad$\{0.5, 1, 2, 4\}$ errors, the average error (avg) and the root-mean-squared error (rms).

5.6.1 Ablation Study

To find the most appropriate hyper parameters for the proposed method, we generate our initial disparity map with a simple feature network. The learned features are then compared using a fixed matching function for a pre-defined number of discrete disparities.

Feature Network Our feature network is a modified version of the U-Net [116, 163] which we use to extract features suitable for stereo matching. We keep the number of parameters low by

Model	Conf	Img	OccIp	Joint	Error [bad3] occ	Error [bad3] noc	#P
WTA					8.24	6.78	480k
WTA + $\text{VN}_4^{7,5}$			✓		5.42	4.68	50k
WTA + $\text{VN}_4^{7,5}$	✓	✓			5.12	3.98	140k
WTA + $\text{VN}_4^{7,5}$	✓		✓		4.43	3.90	73k
WTA + $\text{VN}_4^{7,5}$		✓	✓		3.77	3.07	118k
WTA + $\text{VN}_4^{7,5}$	✓	✓	✓		3.46	2.72	140k
WTA + $\text{VN}_4^{7,5}$	✓	✓	✓	✓	**3.37**	**2.55**	140k
WTA + $\text{VN}_3^{5,7}$	✓	✓	✓	✓	3.43	2.58	133k
WTA + $\text{VN}_2^{8,7}$	✓	✓	✓	✓	3.62	2.97	141k
WTA + $\text{VN}_4^{14,3}$	✓	✓	✓	✓	4.37	3.71	136k
WTA + $\text{VN}_5^{11,3}$	✓	✓	✓	✓	4.25	3.49	134k
WTA + $\text{VNS}_4^{30,5}$	✓	✓	✓		5.24	4.35	20k
WTA + FBS [4]	✓	✓	✓		7.48	6.08	-
WTA + SN [80]		✓	✓		4.02	3.11	114k
WTA + SN [80]	✓	✓	✓		3.78	2.88	114k

Table 5.2: Ablation study on the Kitti 2015 dataset. Conf = Confidences, Img = Image, OccIp = Occlusion inpainting, Joint = joint training, Shared = shared VN parameters, #P = number of parameters. The super-script indicates the number of steps and the filter-size while the sub-script indicates the number of levels in the variational network. $\text{VN}_4^{7,5}$ is therefore a variational network with 7 steps and 4 levels.

only using 64 channels at every layer. The output of our feature network is thus a 64-dimensional feature vector for every pixel. Table 5.1 shows the architecture in tabular format.

Feature Matching Next, we use the extracted features ψ^0 from the left image and ψ^1 from the right image to compute a matching score volume $\tilde{p} : \Omega \times \{0, \dots, D-1\} \to \mathbb{R}$ with

$$\tilde{p}(x, d) = \langle \psi^0(x),\ \psi^1(x - \tilde{d}) \rangle. \tag{5.26}$$

We follow Section 5.4 to compute the inputs for the variational network.

Ablation Study We systematically remove parts of our method in order to show how the final performance is influenced by the individual parts. Table 5.2 shows an overview of all experiments. First, we investigate the influence of our data-terms, the disparity data-term, the confidence data-term and the RGB image data-term. The study shows that each of the data-terms positively influences the final performance. Especially, adding the original input image significantly increases the performance. This can be *e.g.* seen in Fig. 5.4, where the information of how the basket needs to be reconstructed, is derived from the input image. In the second part of the study, we evaluate

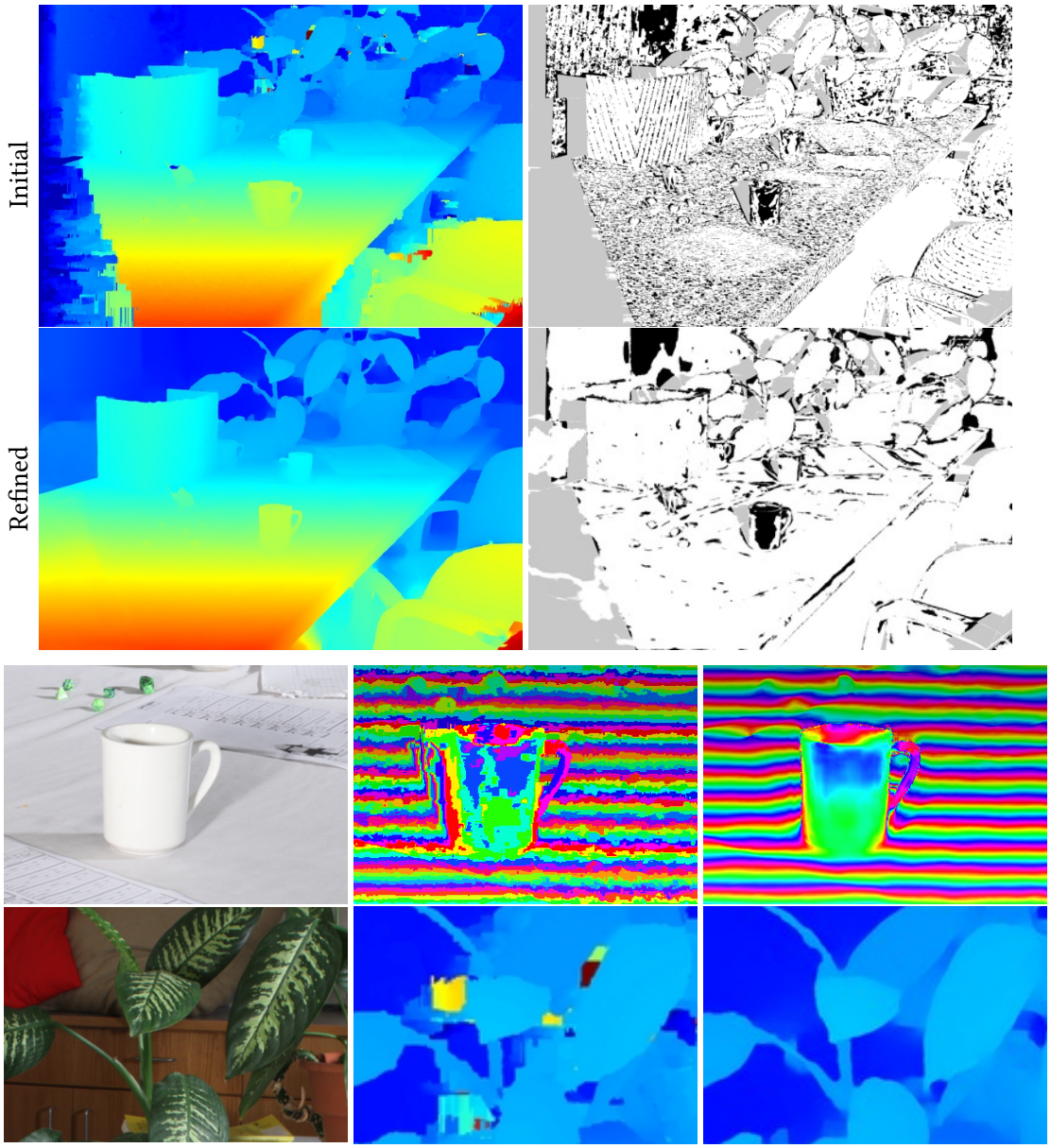


Figure 5.5: Qualitative results on the Middlebury test set. Top-group: Left: Color-coded disparity maps ranging from blue = far away to red = near. Right: Error maps, where white = correct and black = incorrect. The top row shows the initial disparity map (=input to the VN) and the bottom row shows our refined result. Bottom group: Close-up results with input-image, initial disparity map and refined disparity map from left to right. The second column shows a high-frequency visualization of the disparity map.

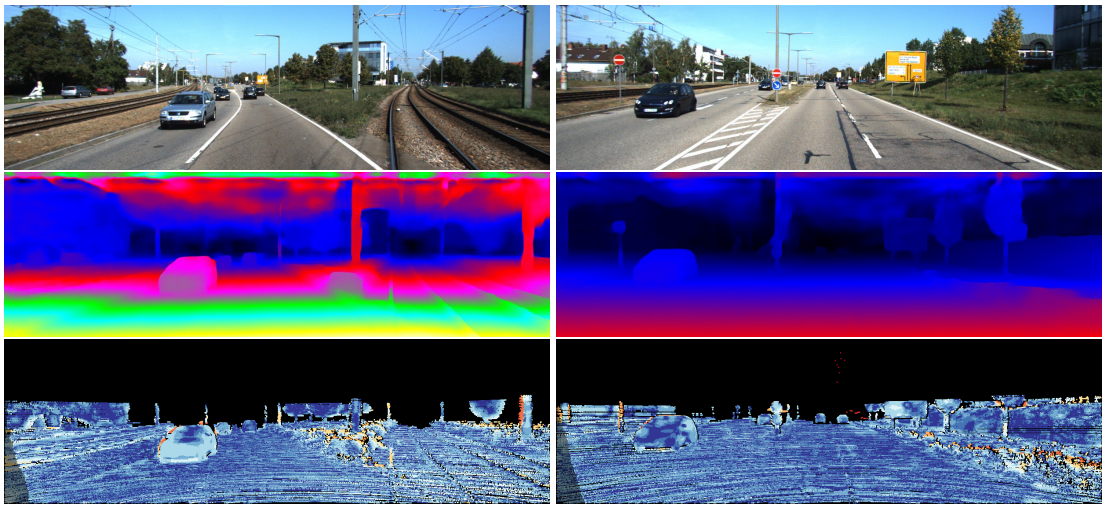


Figure 5.6: Qualitative results on the Kitti 2015 test set. Top-to-bottom: Reference image, disparity map which is color coded with blue = far away to yellow = near, error map, where blue = correct disparity, orange = incorrect disparity.

different variational network architectures. To make the comparison as fair as possible, we chose the variants such that the total number of parameters is approximately the same for all architectures. The experiments show, that a compromise between number of steps, pyramid levels and filter-size yields the best results. The best performing model is the model $\text{VN}_4^{7,5}$, where the filter-size is set to 5×5 for 4 pyramid levels and 7 steps. The average runtime of this VN is as low as 0.09s on an NVidia 2080Ti graphics card.

We use the model $\text{VNS}_4^{30,5}$ to run another experiment where we share the parameters over all iterations in the VN. This shows that we can use the same procedure also in a pure optimization setting. Here, we have significantly less parameters, *i.e.* we have only 20k parameters in the VN while the non-shared version has 140K parameters. We trained the shared model for $T = 30$ iterations and show the result in Table 5.2. The shared model needs more iterations to converge to a good result.

Additionally, we compare with the FBS, because the FBS is defined via a similar optimization problem as our VN. We therefore use exactly the same inputs as we did in our method, i.e., the refined WTA solution $\check{d}$, our confidence measure c and the RGB input image. To ensure the best performance for the FBS, we performed a grid-search over its hyper-parameters on the Kitti dataset. As shown in Table 5.2 the FBS clearly improves the performance upon the initial solution, but the FBS cannot compete with the proposed method.

The next method we want to directly compare with is the StereoNet [80]. StereoNet performs a hierarchical refinement on top of initial disparity maps and is similar lightweight as our model. The refinement in the StereoNet approach is performed with a refinement module consisting of 6 residual blocks and an input and an output mapping layer. While our model contains residual connections implicitly through the optimization structure the authors of StereoNet explicitly designed

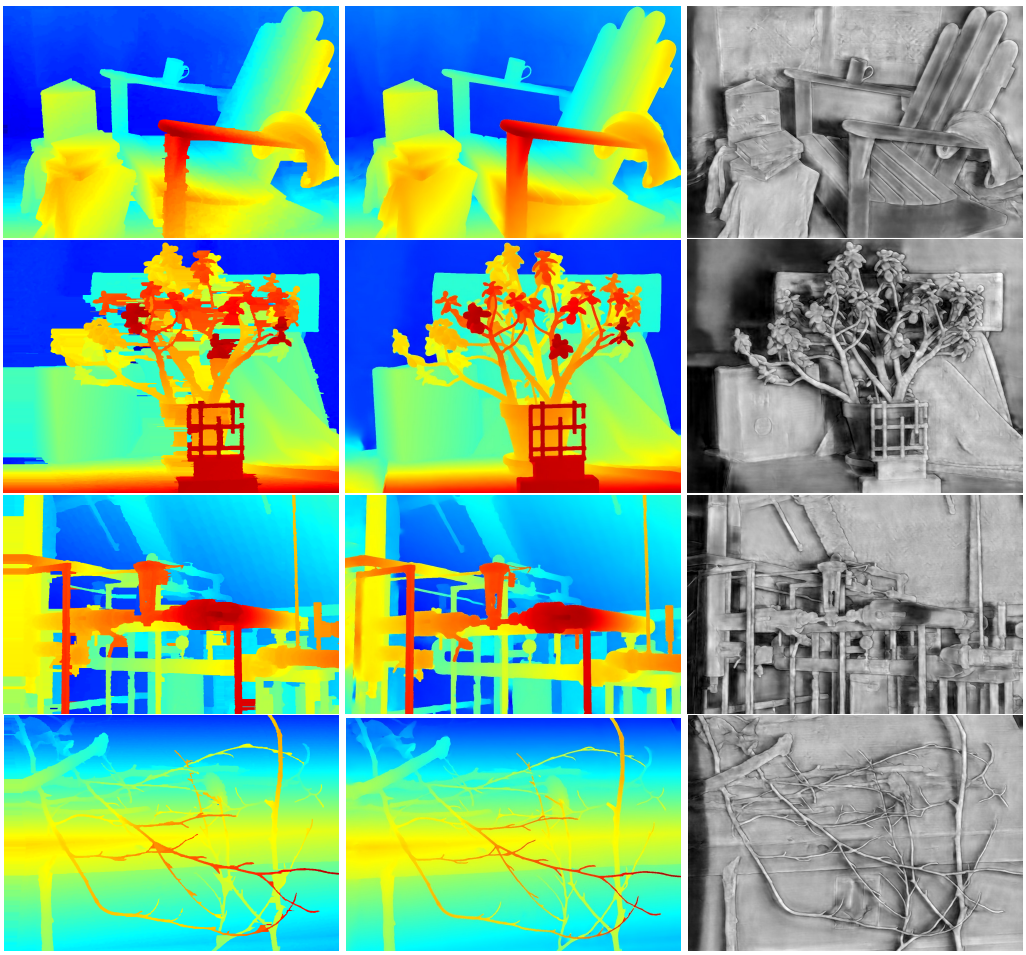


Figure 5.7: Results of $VN_4^{7,11}$ on half size (H) Middlebury images. Left to right: Initial disparity map, refined disparity map, confidences and color image. Our model learns to use object edges to guide the denoising of the disparity map. Best viewed with zoom on the PC.

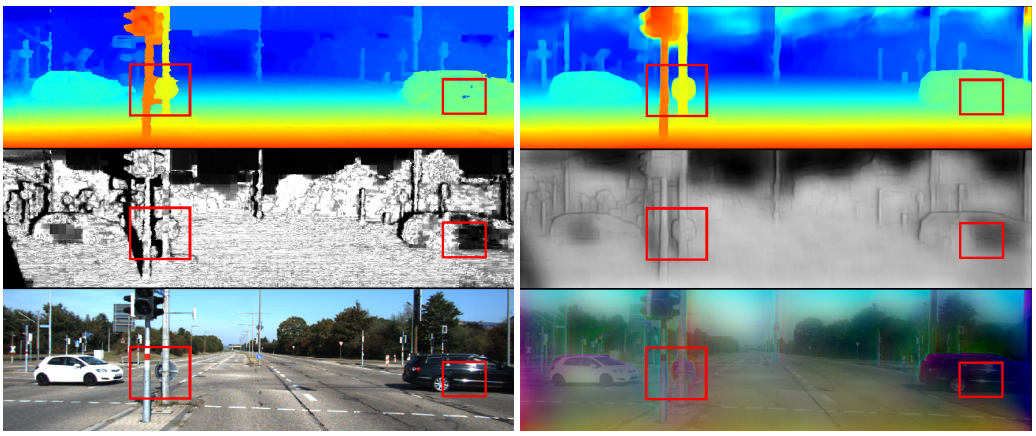


Figure 5.8: Refinement on Kitti. Top to bottom are the disparity map, the confidence map and the color image. Left: Initial results, right refined results. Note especially the highlighted boxes, where artefacts are corrected and fine details are recovered.

Method	Middlebury 2014						
	bad0.5	**bad1**	**bad2**	**bad4**	**avg**	**rms**	**time**
PSMNet [28]	90.0 (90.8)	78.1 (79.9)	58.5 (61.8)	32.2 (37.3)	9.60 (13.3)	21.7 (27.1)	2.62
PDS [194]	54.2 (58.2)	26.1 (31.9)	11.4 (16.7)	5.10 (9.09)	1.98 (3.26)	9.10 (12.7)	10
MC-CNN [220]	42.1 (49.0)	20.5 (29.8)	11.7 (21.5)	7.94 (17.7)	3.87 (12.8)	16.5 (37.5)	1.26
CNN-CRF [85]	56.1 (60.5)	25.1 (32.5)	10.8 (18.9)	6.12 (13.7)	2.30 (9.57)	9.89 (32.0)	3.53
[85] + VN (ours)	***41.8*** (***46.6***)	***17.1*** (***23.0***)	***7.05*** (***12.1***)	***2.96*** (***6.49***)	***1.21*** (***2.06***)	***5.80*** (***8.57***)	4.06
PSMNet [28]	81.1 (82.9)	63.9 (67.3)	42.1 (47.2)	23.5 (27.2)	6.68 (8.78)	19.4 (23.3)	2.62
PDS [194]	58.9 (62.8)	21.1 (38.3)	14.2 (21.0)	6.98 (**12.6**)	3.27 (6.90)	15.7 (27.5)	10.3
MC-CNN [220]	**41.3** (***48.5***)	**18.0** (**28.4**)	**9.47** (**20.6**)	6.7 (17.7)	4.37 (19.3)	22.4 (55.7)	1.26
CNN-CRF [85]	60.9 (65.1)	31.9 (39.4)	12.5 (21.9)	**6.61** (15.9)	3.02 (15.7)	14.4 (49.0)	3.53
[85] + VN (ours)	*56.2* (*61.0*)	*30.0* (*37.5*)	14.2 (22.4)	7.71 (*14.6*)	***2.49*** (***4.98***)	***10.8*** (***17.3***)	4.06

Table 5.3: Performance on the Middlebury 2014 benchmark. We report the numbers of the official online system for non-occluded (all) pixels. Top = Official training set, Bottom = Official test set. Bold font: Overall best. Italic font = improvement of base-line. Note especially the significant improvement of the continuous error metrics avg and rms on all pixels.

Method	Kitti 2015 (train)		Kitti 2015 (test)	
	noc	**all**	**noc**	**all**
PSMNet [28]	-	**1.83**	**2.14**	**2.32**
PDS [194]	-	-	2.36	2.58
MC-CNN [220]	-	-	3.33	3.89
CNN-CRF [85]	-	4.04	4.84	5.50
[85] + VN (ours)	***1.90***	*2.04*	*4.45*	*4.85*

Table 5.4: Performance on the Kitti 2015 benchmark. We provide the official bad3 error metric on non-occluded (noc) and all pixels on the training set (left) and on the test set (right). The VN improves the base-line method on both metrics.

them in their architecture. The receptive field is similar to ours, but instead of downsampling the authors used dilated convolutions. The inputs to the StereoNet are the RGB color image and the initial disparity map. We will investigate the performance of StereoNet on top of our feature net in the original setting *i.e.* without the confidences and additionally we show the benefit of using confidences in the StereoNet as well in Table 5.2. The ablation study shows that the proposed VN compares favorable to the StereoNet in both variants, with and without additional confidences as input. Thus we can conclude that the structure arising from an optimization problem is also beneficial in terms of final performance in the learning setting.

5.6.2 Benchmark Performance

We use our method on top of the CNN-CRF [85] stereo method for the official test set evaluation (see Tables 5.3 and 5.4). We set the temperature parameter $\eta = 0.075$ in all experiments.

We used the model $\text{VN}_4^{7,5}$ on the Kitti dataset, since this model performed best in the ablation study. As shown in Table 5.4 we reduce the bad3 error in both, occluded and in non-occluded regions. The relative improvement brought by the VN is 8% for occluded pixels and 12% for all pixels. Thus, the experiment shows that the VN is especially beneficial in occluded pixels. Fig. 5.6 shows qualitative results with the corresponding error maps on the Kitti test set.

On the Middlebury benchmark we use the model $\text{VN}_4^{7,11}$ for all evaluations, where we have choosen a larger filter size to account for the high-resolution images in this benchmark. We compare the errors on the training set with the errors on the test set (Table 5.3) and observe first that our method shows a significant improvement over the baseline method on the continuous error metrics avg and rms on both the test set and the training set in non-occluded and all pixels. This is understandable, because we have used the continuous Huber loss (5.24) for training the VN. The Huber loss is a combination of the ℓ_1 and ℓ_2 error and thus minimizes the continuous error metrics. However, we can also see that minimizing the continuous error metric does not necessarily yield better results for the badX error measure, which can be explained by the fact that the Huber loss does not provide a good proxy for the badX measures. While the VN can at least slightly improve the results on bad$\{0.5,1,4\}$, the error is slightly increased on the bad2 error on the test set compared to [85]. This is in contrast to the training set, where the VN can improve on all badX error measures as well. Similar to the behavior on the Kitti dataset, the benefit of the VN is significant especially in occluded regions, where we have reduced the average error from 15.7 to 4.98 which is a relative improvement of almost 70% over the baseline method. This is also noticeable visually in Fig. 5.7, where the VN is often able to perfectly fill in occluded regions. To conclude, we have seen that the VN yields state-of-the-art (SOTA) results using continuous error metrics for evaluation, but trails SOTA on the badX error metric. Fig. 5.5 shows a qualitative example of the Middlebury test set. Note that the tabletop is nice and smooth while the sharp edges of the objects are very well preserved.

5.7 Analyzing the VN

One of the main benefits of a variational network compared to other CNNs is the interpretability of the VN. Due to the optimization-like architecture, we can visualize the individual steps, interpret the learned filters and activation functions, compute eigen-disparity maps, which are non-linear eigenvectors of our learned regularizer, and investigate the quality of our confidence maps. We address all these properties of our model in the next sub-sections.

5.7.1 Learned Filters and Activation Functions

In this section we investigate the structure of our learned filters and plot the learned activation functions. Visualizing the filters can be easily done in the VN, because our filters always operate

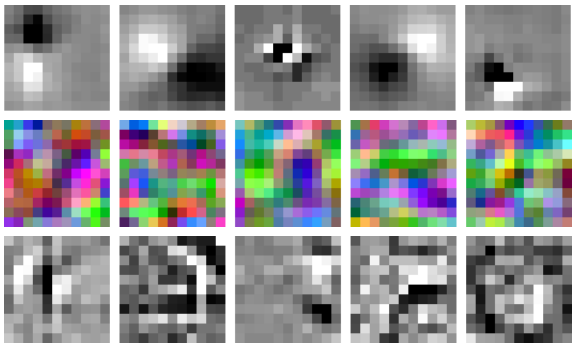


Figure 5.9: Visualization of the learned filters of our model. Top to bottom: Filters of the disparity map, filters of the RGB color image and filters of the confidence map.

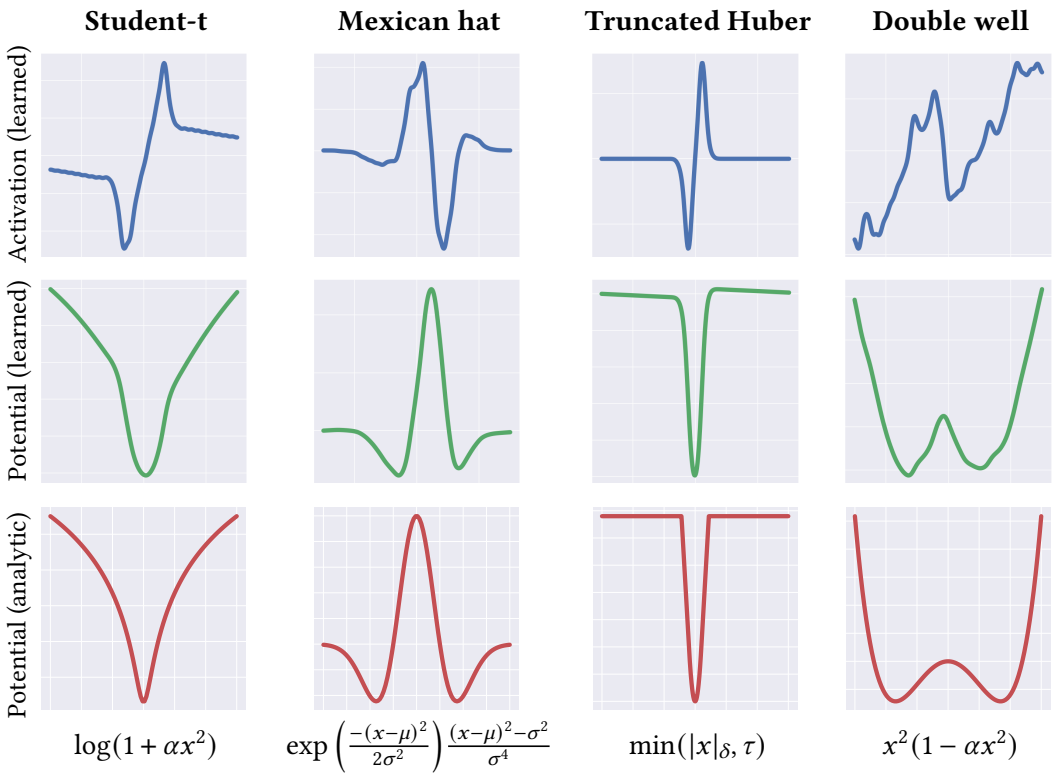


Figure 5.10: Visualization of learned activation functions of our model. The first row (blue) shows the activation functions ρ in the derivative space. The second row (green) shows the corresponding potential functions ϕ in the energy domain. The thrid row (red) shows analytic potential functions corresponding to the expressions shown on the left for comparison.

in the 2D image space directly. Note that this visualization technique is not possible in other CNNs, because the filters are usually 3D in convolution layers and thus, they can not be directly plotted. For our visualization we split the learned 5-dimensional filters into three parts which can then be interpreted as the filters for the disparity map, for the color image and for the confidence

maps. Fig. 5.9 shows selected filter kernels. The first row contains filters for the disparity map, the second row contains filters for the RGB color image and the third row contains filters for the confidence map. Note that the learned filters contain structure which makes them interpretable. The structure can be clearly seen in the disparity filters, which look like Gabor filters. The color filters contain structure as well and can be interpreted as texture filters. The middle filter could be an ellipsoidal blob detector. The confidence filters seem to capture the edge information between low confident and high confident regions around edges. The color- and confidence-filters are not as smooth as the disparity filters, which can be explained by the fact that we did not use any loss function on the color and confidence channel. The structure in the filters suggests that our model actually captures statistics of how to appropriately refine disparity maps, confidence maps and color images jointly.

Figure 5.10 shows the learned activation functions. We can integrate the learned activation functions (blue) to get the potential functions (green) used in our energy. Similar as for the learned filters, the learned activation functions can also be interpreted. We plot in Fig. 5.10 prototypical learned potential function of our model. Starting from left to right we can see instances of a Student-t potential, the Mexican hat function, a truncated Huber function and a double-well potential. For comparison, we show the analytic potential functions in the last row in red and state the corresponding analytic expressions. We can *e.g.* see that our model has learned to be robust against outliers with the first (Student-t) and the third (truncated Huber) potential function. We also observe that we have found similar functions as *e.g.* Chen *et al.* [32] for denoising and Zhu *et al.* [229].

5.7.2 Shared Parameters

In this section we restrict the parameters to be shared for all iterations of the VN. Since we are in a pure optimization setting we can perform additional experiments such as computing eigen disparity maps, eigen image and eigen confidence maps.

Using shared parameters during the iterations of the VN requires us to change Eq. (5.14) to

$$\mathbf{u}_{t+1} = \operatorname{prox}_{\alpha \mathcal{D}(\cdot,\theta)}(\mathbf{u}_t - \alpha \nabla \mathcal{R}(\mathbf{u}_t, \theta)), \;\; 0 \leq t \leq T-1 \tag{5.27}$$

where we removed the index t in all parameters. Next, we compute the eigenmodes of the learned regularizer. We therefore use the same shared model as in the ablation study in Table 5.2, *i.e.* $\text{VNS}_4^{30,5}$.

Eigenmodes of the VN We show how we can compute eigenmodes of our learned regularizer in the refinement VN by adapting the approach of [50]. This allows us to visualize the eigenmodes of our regularizer as images and we can thus interpret them. The eigenimages give insights into what the regularizers has learned, since they reveal prototypical structures yielding a low energy of the regularizer.

Recall the classical eigenvalue/eigenvector problem

$$Au = \lambda u \tag{5.28}$$

with $A \in \mathcal{S}^{N \times N}$, where $\mathcal{S}$ is the space of symmetric positive definite matrices. λ and u are the sought eigenvalue/eigenvector pairs. To motivate the way we compute our eigenmodes, we note that the left hand side of Eq. (5.28) can also be derived using the gradient of a quadratic function $Q(u) = \frac{1}{2}u^T Au$, where we get

$$\nabla Q(u) = \lambda u. \tag{5.29}$$

In order to apply the eigenmode analysis to our refinement VN, we replace the quadratic function $Q(u)$ with our non-linear regularizer and get

$$\nabla \mathcal{R}(\mathbf{u}) = \lambda \mathbf{u}, \tag{5.30}$$

which corresponds to a non-linear eigenvalue/eigenvector problem. Since we cannot compute a solution to (5.30) in closed form, we propose to compute approximate solutions by solving the nonlinear least squares optimization problem

$$\min_{\mathbf{u},\lambda} \frac{1}{2} \|\nabla \mathcal{R}(\mathbf{u}) - \lambda \mathbf{u}\|^2. \tag{5.31}$$

First, we observe that we can actually solve for λ^* in closed form by setting the derivative w.r.t. λ to zero. Thus, we get

$$\lambda^* = \frac{\langle \mathbf{u}, \nabla \mathcal{R}(\mathbf{u}) \rangle}{\|\mathbf{u}\|^2}, \tag{5.32}$$

which is known as the Rayleigh quotient. Substituting (5.32) back into (5.30) yields the new optimization problem

$$\min_{\mathbf{u}} \frac{1}{2} \left\| \nabla \mathcal{R}(\mathbf{u}) - \frac{\langle \mathbf{u}, \nabla \mathcal{R}(\mathbf{u}) \rangle \mathbf{u}}{\|\mathbf{u}\|^2} \right\|^2, \tag{5.33}$$

where we have additionally restricted the elements of the variable $\mathbf{u}$ to the interval $[0, 1]$.

Now we are ready to move on to the eigenmode computation of the VN. Therefore, we solve problem (5.33) with Nesterov's proximal accelerated gradient method [139] in order to actually compute the eigen disparity-maps. We iterate until convergence and get a pixel-wise residuum of less than $2 \cdot 10^{-6}$, which indicates that we approximate eigenmaps of high quality. The computed eigenimages are of particular interest since substituting an eigenmode back into our model (5.5) yields

$$\begin{aligned} \mathbf{u}_{t+1} &= \operatorname{prox}_{\alpha_t \mathcal{D}}(\mathbf{u}_t - \alpha_t \overbrace{\nabla \mathcal{R}(\mathbf{u}_t)}^{\lambda \mathbf{u}_t}) \\ &= \operatorname{prox}_{\alpha_t \mathcal{D}}(\mathbf{u}_t (1 - \alpha_t \lambda)) \end{aligned} \tag{5.34}$$

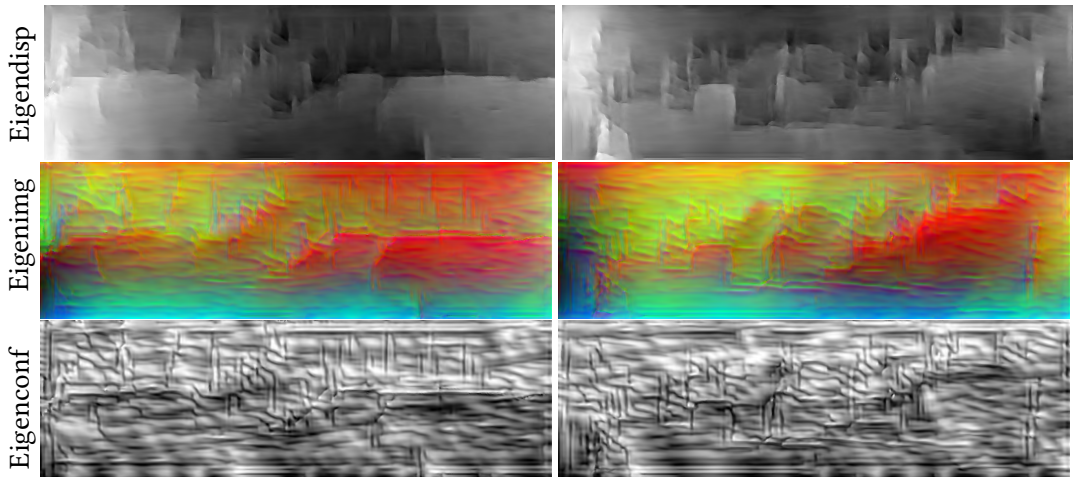


Figure 5.11: Eigenimages. Top to bottom shows the eigenimage of our learned regularizer for disparity map, color image and confidence map. Note that the regularizer learned to favour pole-like structures, car parts and slanted surfaces. This fits perfectly to the scenery of the Kitti dataset. Best viewed in color on the screen.

and reveals that the regularizer adapts only the contrast to capture the correct disparity. Thus, the structure contained in the eigenimages is kept and transferred to the outputs of our model.

Fig. 5.11 shows two examples of all three eigen maps, where we have used different initializations to compute different eigenmaps. It can be easily seen that our regularizer learned the stucture of local disparity maplets, *i.e.* local parts of natural disparity maps. We see that our regularizer prefers to accurately align the edges in all three components, the eigendisparity, eigenimage and eigenconfidence. They consist of *e.g.* pole-like and car-like structures as well as slanted surfaces.

Another way to interpret the eigen disparity maps is in terms of energy. Therefore, we observe that the Karush-Kuhn-Tucker condition of the constraint optimization problem

$$\min_{\mathbf{u}} \mathcal{R}(\mathbf{u}) \quad \text{s.t. } \frac{1}{2}\|\mathbf{u}\|_2^2 = \rho(\lambda), \tag{5.35}$$

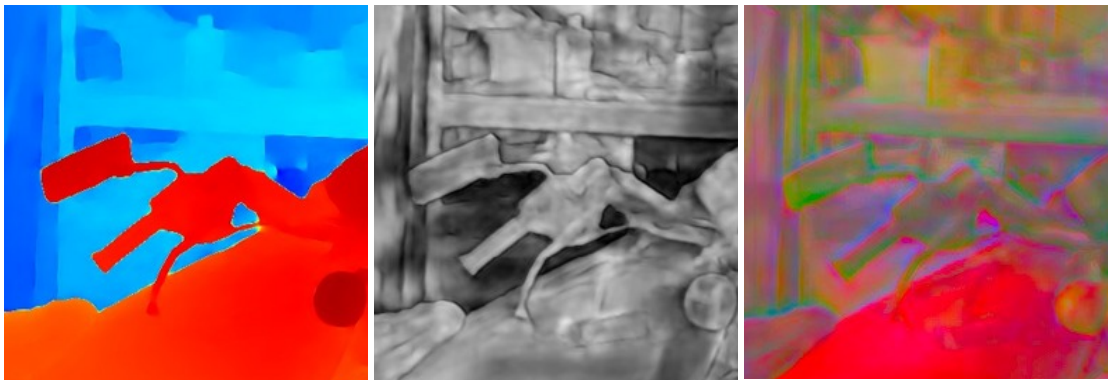
for an unknown function ρ depending on the eigenvalue λ is given by

$$\nabla\mathcal{R}(\mathbf{u}) = \lambda\mathbf{u}, \tag{5.36}$$

which resembles exactly the non-linear eigenvalue problem defined in Eq. (5.30). Thus, we are seeking images which contain structure, but have a low energy in the regularizer. The eigenmaps shown in Fig. 5.11 contain frequent structures of natural disparity maps and thus confirm that we have learned a regularizer suitable for disparity refinement.

5.7.3 VN Color Image

In this section we finally provide a possible interpretation of the processed color image. The color



Disparity Confidence Color

Figure 5.12: Detailed view of the outputs of the VN. Note that all three images contain sharp edges at depth discontinuities. The color image is normalized for better visualization. Best viewed zoomed on the PC.

image is used to support the VN during refining the disparity map. As visualized in Fig. 5.4 it provides *e.g.* edges to guide the disparity refinement process. Figure 5.12 shows a detailed view of the three outputs of the VN, the refined disparity map, confidence image and color image. We first observe that both the confidence and the color image have edges at depth discontinuities. For the color image, we see on the one hand that the green channel captures depth discontinuities on the right side of object boundaries, where no occlusions exist. On the other hand, the blue channel seems to capture the left side of the object boundaries and can be interpreted as an occlusion detector. Thus, the color image shows a tendency to capture problem specific information in the processed color images. It is therefore used as a memory channel for the VN. Using the color image as an additional input during the refinement process yields not only better quantitative results as shown in Table 5.2, but contains also abstracted, but still interpretable information about the stereo problem.

5.7.4 VN Confidences

In our collaborative refinement model we do not only refine the disparity map but we additionally implicitly refine the initial confidences as well. Note that we do not put any loss on the confidences during learning. We show in this section that the refined confidences are more reliable compared to the initial confidence values. Therefore, we compared the initial confidences from our feature network (*e.g.* Fig. 5.13, left top) with the confidences generated by the VN (*e.g.* Fig. 5.13, left bottom). For showing the reliability of the confidences we compute Receiver Operating Characteristics (ROC) for both confidence maps. Therefore, we compute the True Positive Rate[2] (TPR)

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{5.37}$$

[2] Also known as *Recall*

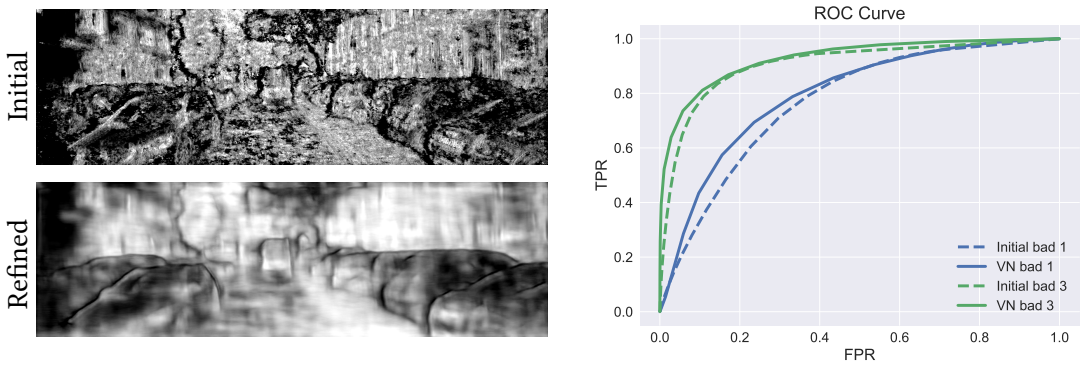


Figure 5.13: Left: Initial confidences (top) and VN confidences (bottom). Right: ROC curve for initial confidences and VN confidences. The blue curve shows the confidences provided by the $\text{VN}_4^{7,5}$ and the green curve shows the initial confidences. The larger the area under the curve, the better. Thus, the confidences provided by our VN reflect the actual performance better than the initial confidences. TPR = True positive rate, FPR = False positive rate.

where TP are the true positives and FN are the false negatives and the False Positive Rate (FPR)

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}, \tag{5.38}$$

where FP are the false positives and TN are the true negatives. To compute the all terms in Eqs. (5.37) and (5.38) we use a threshold δ to split all predicted disparities into two two sets $A \geq \delta$ for confident predictions and $B < \delta$ for vague predictions, respectively. Intuitively, the larger δ, the more reliable the predictions in set A should be. Thus, we define the TP and FP as the data points in A having a disparity error less than and larger than 1 or 3 pixels, respectively. Similarly, we use the set B to define the FN and TN as the data points in B having a disparity error less than and larger than 1 or 3 pixels, respectively. The ROC curve can then be computed by evaluating Eqs. (5.37) and (5.38) for a range of thresholds δ.

Fig. 5.13 right shows the ROC curves of the initial confidences and the VN confidences. We construct the plot using the same data as we used in the ablation study. The larger the area under curve (AUC) in this plot, the better the confidences. The solid lines show the ROC curves for the VN and the dashed curves show the ROC curves of the initial confidences. We report the curves for the bad3 (green) and the bad1 (blue) error metric. Consider for example the point $(0.1, 0.82)$ on the green curve (bad3), where we have selected the FPR of 0.1. The ROC curve reveals here that the confidences predicted by our model are reliable with a TPR of 0.82, while the FPR is as low as 0.1. If we decrease δ, we get more points into the set A and have therefore the chance to get a higher TPR, but we will also increase the FPR as can be seen in Fig. 5.13. The ROC curve of the confidences of the VN are always above the curve of the initial confidences which yields a higher AUC for the VN. Thus, we have shown that the refined confidences are more reliable than the initial confidences.

5.8 Conclusion & Future Work

We have proposed a learnable variational network for efficient refinement of disparity maps. The learned collaborative and hierarchical refinement method allows the use of information from the joint color, confidence and disparity space from multiple spatial resolutions. In an ablation study, we evaluated a broad range of architectural choices and demonstrated the impact of our design decisions. Our method can be applied on top of any other stereo method and explicitly exploits confidence information contained in a cost volume. We demonstrated this by adding the variational refinement network on top of the CNN-CRF method and have shown improved results. We have shown insights and interpretations of our model in terms of visualizing the indermediate steps, the learned filters and activation functions. The optimization like structure of our model additionally allowed us to compute eigen disparity maps, eigen color images and eigen confidence maps. Furthermore, we have proven the effectiveness of our method by participating in the publicly available stereo benchmarks of Middlebury and Kitti. In future work, we would like to include a matching score during the refinement process and perform data augmentation to increase the training set for learning.

6

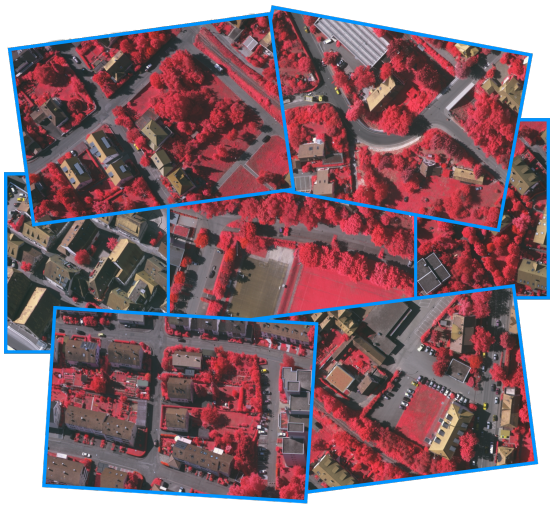
Self-learning for Stereo

Deep Convolutional Neural Networks (CNNs) have replaced almost any hand-crafted stereo methods. This is, because they achieve a significantly better performance on the available benchmarks. However, this high performance is expensively payed in terms of the required amount of accurately labeled training data. In the paper presented in this chapter, we show how we can exploit Deep Learning (DL) without *the requirement of any hand-labelled training data. We therefore use an aerial dataset and show how we can adapt self-supervised learning for stereo. The results show impressively that self-supervised learning exploits all available data and therefore results in highly accurate disparity maps.*

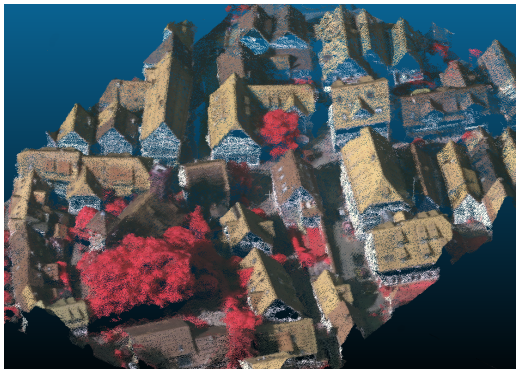
This paper was presented at IGARSS 2018 in Valencia and was awarded with the Symposium Paper Award.

Contents

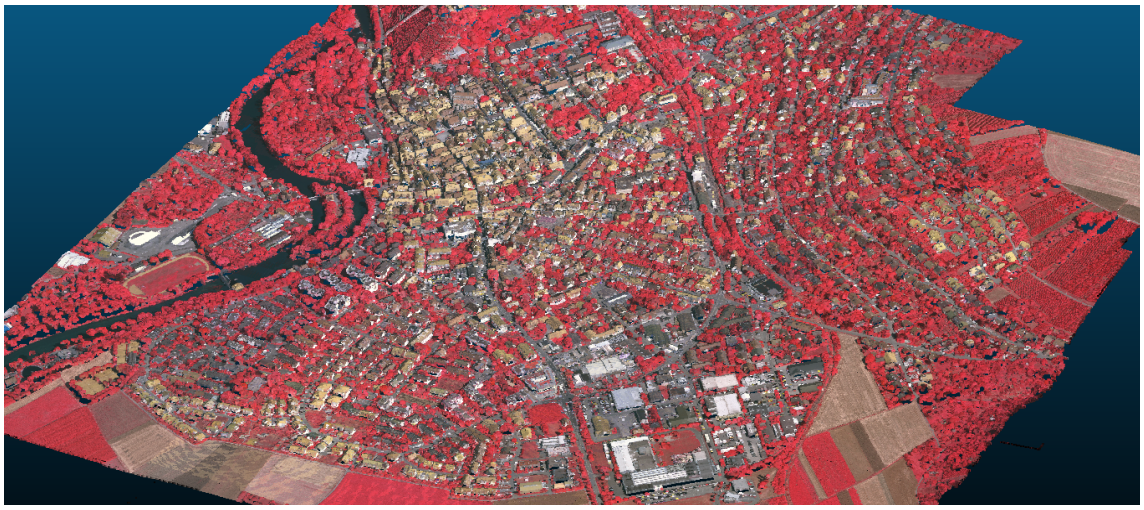
6.1 Introduction **162**
6.2 Related Work **163**
6.3 Self-Supervised Dense Matching **164**
6.4 Experiments **166**
6.5 Conclusion & Future Work **171**



(a) Input Images



(b) Textured 3D Pointcloud



(c) Complete 3D point cloud of Vaihingen

Figure 6.1: Visualization of the textured 3D point cloud of Vaihingen generated by our algorithm. Figure 6.1a shows the input images, Fig. 6.1b shows a close-up visualization of our computed textured 3D point cloud and Fig. 6.1c shows the complete reconstructed area consisting of ≈ 250 million points. Note that the channels of the image are Infrared-Red-Green and thus vegetation appears red.

6.1 Introduction

Acquired with modern high resolution cameras, aerial images can provide accurate 3D measurements of the observed scene via dense image matching. Consequently, through the years, stereo estimation has emerged as an attractive alternative to LiDAR (Light Detection and Ranging) in various tasks, like high resolution Digital Surface Model (DSM) generation or orthoimage produc-

tion [58], leading to a simplified processing pipeline and reduced (flight) costs. Furthermore, the generation of stereo data is a common first step in many different applications, *e.g.* in 3D change detection [157] or semantic 3D reconstruction of urban scenes [13].

Recently, machine learning, and in particular, deep learning has affected many low-level vision tasks including stereo estimation, leading to considerable improved performance. Here, convolutional neural networks (CNNs) can be used to replace different parts in conventional stereo pipelines, *e.g.* the feature generation for computing the data term [220]. A different path is to directly formulate stereo estimation as a regression task [126]. In this work, we follow the former approach, which naturally requires less parameters, leading to easier to train networks and in our experience also a better generalization performance. Especially the latter feature is attractive for our aerial reconstruction task. Dense ground truth data is notoriously hard to acquire, while artificial datasets [126] usually lack the photogrammetric properties of 'real world scenes' and especially of the specific dataset in consideration. The problem of missing ground truth data is further magnified by the fact that CNNs demand a lot of labeled training data to expose their performance. While LiDAR measurements could provide at least (very) sparse ground truth, such an approach would mitigate the advantages of utilizing image based matching at all, with the additional problem that these measurements appear too sparse to be of use for CNN training. Nevertheless, the aim of this work is to utilize CNNs for stereo estimation of aerial scenes. To that end, we propose a *self-supervised learning framework*. Instead of formulating the problem as an unsupervised learning task, which ultimately leads to a fully generative approach, we rather directly utilize the dataset that has to be reconstructed as training data. In that sense, we are able to learn the specific imaging characteristics at hand. Our self-learning approach is summarized in Fig. 6.2 Starting from a pre-trained version of our network, we generate the training data simply by applying our reconstruction method on the whole dataset. To secure the integrity of our training data we employ strict and conservative outlier filtering and apply our training procedure on the unmasked, but still dense data. Our experiments indicate that this concept can lead to highly accurate reconstructions, improving the completeness (and accuracy) from 5 (4) percent up to 22 (24) percent, if compared to our pre-trained model and other competing stereo methods.

6.2 Related Work

Commonly, dense stereo estimation from aerial images is formulated as a label-based Markov-Random Field (MRF) energy optimization problem, where methods operate on rectified image pairs. A popular representative is Semi-Global Matching (SGM) [68, 70] that approximately solves the MRF energy via dynamic programming (DP), with four scanlines per pixel. Later, the work of Zbontar et al. [220] paved the way for deep learning for stereo. They propose to replace the usual, handcrafted features that are used to define the data term in the energy, with a learned representation. Later, Luo et al. [119] exchange the patch-wise training of [220] with a method that learns the features on whole images instead, introducing a differentiable cost volume formulation in the CNN. Both methods rely on SGM to find a solution of their energy formulation and employ various post-processing steps to refine the solution. Mayer et al. [126] instead directly formulate

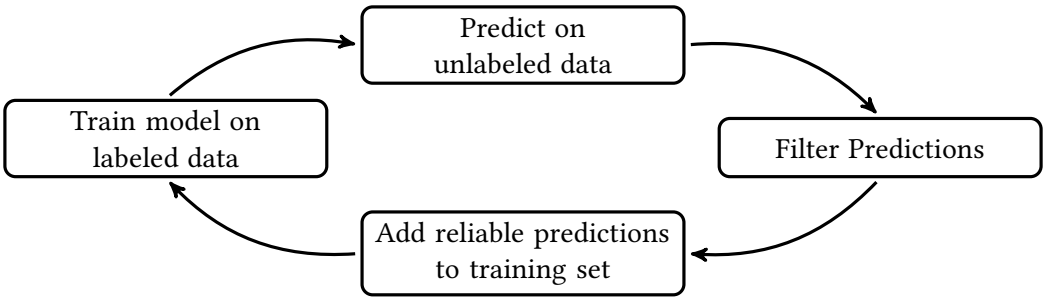


Figure 6.2: Self-Learning Cycle. We first prediction on unlabeled data, filter the predictions and add the reliable predictions to the training set in order to train the model with the labeled data. Note that the cycle can also be repeated.

the problem as an end-to-end regression task. Their CNN possesses several millions of parameters and, hence, requires a large amount of synthetic data for training.

To overcome the requirement for a sufficient amount of training data, the recent trend is to use only weak supervision. Tonioni et al. [191] generate their training data from a traditional formulation [216], but estimate a confidence score for the established matches with another CNN [154]. For training their regression network, the loss function combines the confidence weight to penalize deviations to their generated training data with an additional smoothness constraint on the solution. In contrast, we generate our training data using a state-of-the-art learned model [87] and employ a geometrically motivated consistency check with a hard, conservatively chosen threshold. [193] explicitly utilize a pre-defined list of matching constraints to guide the learning. To that end, they are restricted to train the network per scanline to encode the constraints in the learning procedure. Another regression based approach is proposed by Zhou et al. [227]. They start from a randomly initialized network and construct their training data using their own reliable predictions. Matches are considered as reliable if they survive a left-right (LR) consistency check. The network is then trained using only the reliable matches. The method is similar in spirit to our approach. In contrast, we advocate to start from a much better initialization using a pre-trained model [87]. In our experience this procedure is both, beneficial in training time and final accuracy. Apart from that, our model is much closer to the traditional MRF problem.

6.3 Self-Supervised Dense Matching

In our setting we assume to have access to a larger set of already rectified image pairs on which we want to perform stereo matching. What we do not assume is to have access to ground truth data for any of these image pairs, which could be used for training. Our objective is to still apply a state-of-the-art stereo CNN and boost its performance on this specific dataset. In a nutshell, we exploit a pre-trained and – during training – continuously improving versions of the CNN to generate our own training data.

CNN-CRF Model. In this work we utilize the hybrid CNN-CRF model proposed in [87] that

incorporates deep learning into classical energy minimization. Our CNN-CRF model minimizes the following typical CRF-type energy defined on the pixel graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ of an image Ω with the usual 4-connected neighborhood structure $\mathcal{E}$:

$$\min_{x \in \mathcal{L}} \sum_{i \in \mathcal{V}} f_i(x_i) + \sum_{i \sim j \in \mathcal{E}} f_{ij}(x_i, x_j). \tag{6.1}$$

The solution x^* of (6.1) is a member of the set of mappings $\mathcal{L} : \mathcal{V} \rightarrow \{0, \dots, d-1\}^{|\mathcal{V}|}$ representing a disparity map of Ω of range d. Here, both, the data-term $f_i(x_i)$ and the regularizer $f_{ij}(x_i, x_j)$ are each represented as a CNN. The optimization of the CRF energy is performed via a massively parallel and highly efficient variant of dual decomposition. The whole system can be learned end-to-end [87]. In this work, however, we focus on the data term f_i and keep edge-weights and penalty function f_{ij} in (6.1) fix.

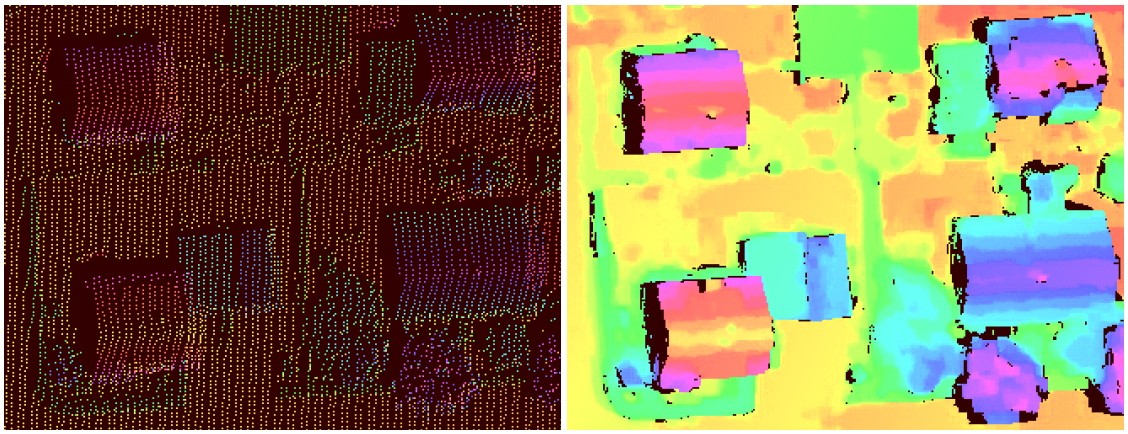
Generating the training data. To bootstrap our procedure, we directly use the publicly available model (`https://github.com/VLOGroup/cnn-crf-stereo`) with a 7 layered data term CNN, which was trained on the Middlebury Stereo 2014 dataset [169]. It has been shown in [87] that the model generalizes well to unknown scenes, which arguably makes it a good candidate for generating our initial training data. However, because the original training images are completely different from our aerial dataset, the reconstruction still contains outliers and erroneous regions. Therefore, directly using the resulting disparity images for training a new data term will rather harm the performance than improve our method. To mitigate this problem, our training procedure has to distinguish between regions, where it can trust the generated ground truth and where not.

Filtering the generated data. We use the common left-right consistency check to filter unreliable matches. Therefore, we first compute two disparity maps, d_l and d_r, for each image pair, where either the left image (d_l) or the right one (d_r) serves as the reference frame. For our filter we then require that matching points in the left and right image are in mutual correspondence for both disparity maps. More precisely, a pixel x survives the left-right consistency check if

$$|d_l(x) + d_r(x + d_l(x))| < \epsilon, \tag{6.2}$$

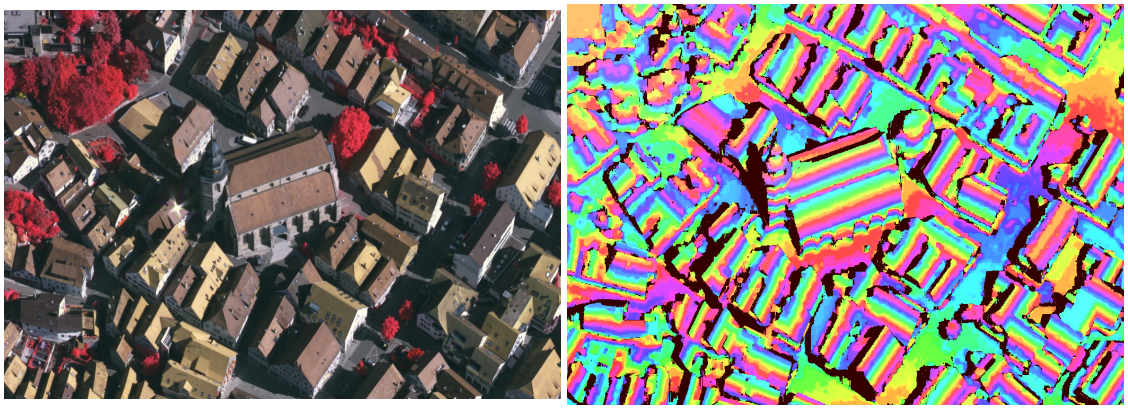
where ϵ is a threshold that is set to 0.9 in our experiments. This simple check gets rid of most of the wrong pixels and is, in our experience, already sufficient to retrain our model.

Training. As stated in Section 6.3, the model consists of two networks, one for the data-term f_i and one for the regularizer f_{ij}. From our experience, training edge costs for our regularizer requires the edges also to be represented in the training data. However, pixel near occlusions rarely survive our consistency check and are, thus, underrepresented in our self-supervised training data. Consequently, we keep the edge costs fixed and only retrain the network represented by f_i for the aerial images. In particular, we generate a one-hot encoding of our ground truth disparity maps and perform maximum likelihood training, i.e. we minimize the following loss function



(a) Lidar (b) Our Result

Figure 6.3: Close-up comparison between our computed depth values and the laser depth values.



(a) Reference image (b) Our result

Figure 6.4: Detailed view of a church. Note how accurately the church tower and the facade is reconstructed.

w.r.t. the parameters θ of the network:

$$L(f(\theta), f^*) = -\sum_{i\in\Omega}\sum_{d\in D} f^*_{i,d} \log f_{i,d} = -\sum_{i\in\Omega} \log f_{i,d^*}, \tag{6.3}$$

where f is the correlation volume predicted by the model, f^* is the one-hot encoding of the ground truth disparity map. The second equality comes from the fact that the one-hot encoding puts all the probability mass to the ground truth disparity d^*.

6.4 Experiments

In this section we evaluate the effectiveness of our self learning algorithm in the context of aerial images and street scenes for autonomous driving. We compare the depth maps generated by the

well-known Semi-Global Matching algorithm [2, 68] with the pre-trained CNN-CRF model and our model, refined via self-supervised training.

6.4.1 Vaihingen Dataset

We evaluate our method on the Vaihingen dataset of the ISPRS Urban classification and 3D reconstruction benchmark [166]. The Vaihingen dataset consists of 20 aerial images of size 7680×13824 pixels. In each image of the dataset the blue channel has been replaced by the response of an infrared camera, which leads to further deviation between the pre-trained and refined model. Nevertheless, we could observe similar behavior for the Toronto dataset of the same benchmark [166] where the color channels are RGB. All images are registered in a global coordinate system. Additionally a laser point cloud is provided, which we use for our evaluation. We perform all our experiments at half resolution.

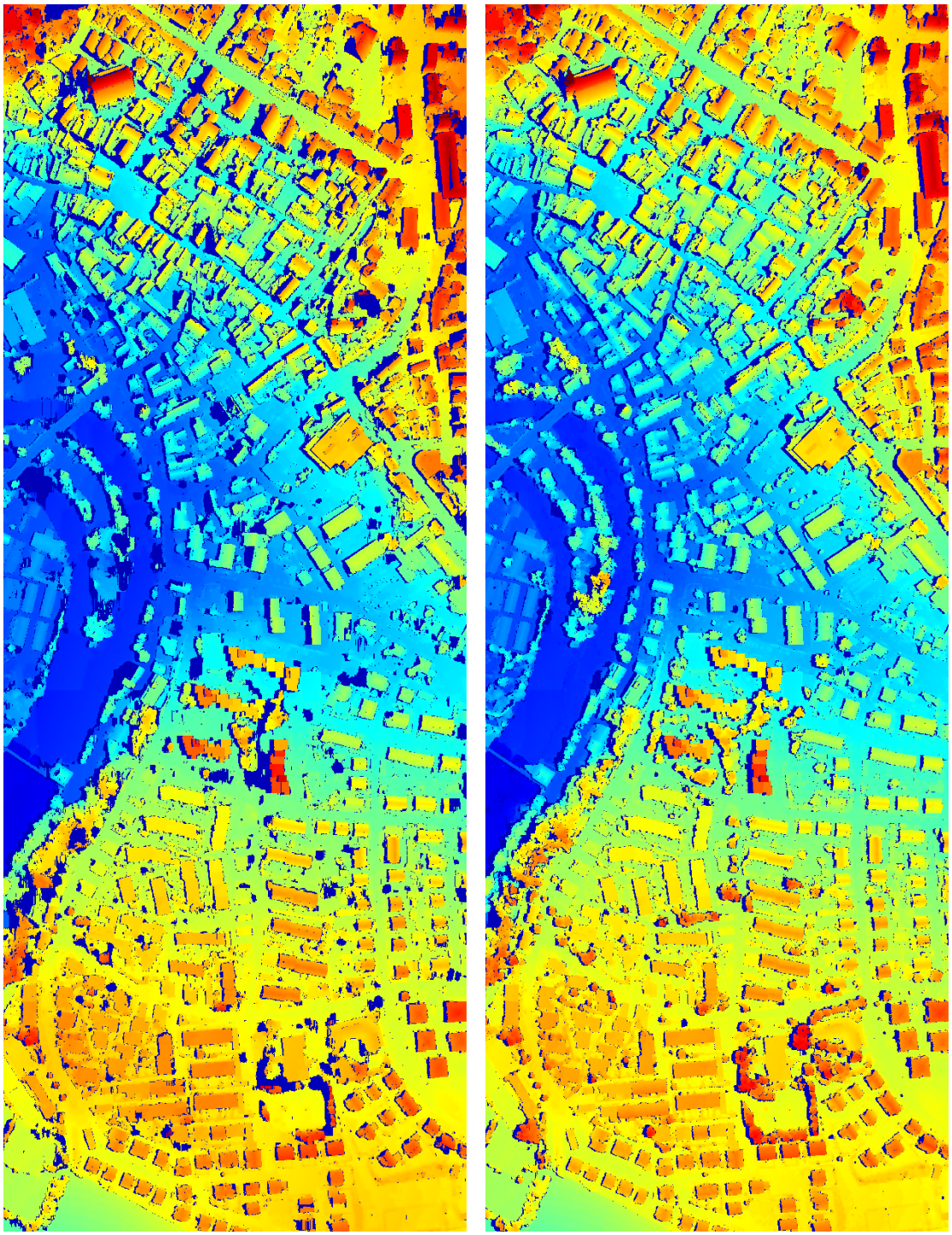
Both algorithms, SGM and CNN-CRF, require rectified input images. In order to limit the memory consumption during training, we additionally divide the images into parts.

Performance evaluation. We use the provided laser scanned depth values as our reference data to compare the different models. The pipeline for the evaluation consists of **(i)** computing the disparity map in pixel space for an image pair, **(ii)** using the disparity to compute the metric depth value for all pixels in the reference image, **(iii)** projecting the laser point cloud into the reference image and **(iv)** computing the metric difference for all valid pixels in pixel space. Additionally, we compute the recall of the reconstructed points, given by

$$recall = \frac{|\mathcal{P}_S \cap \mathcal{P}_L|}{|\mathcal{P}_L|}, \tag{6.4}$$

where P_S is the set of pixel with a valid (surviving the consistency check) disparity and P_L the set of pixels with a Laser measurement. A recall of 100% would mean that every pixel captured by the laser scanner is also captured by our model. We perform the evaluation using all available images and, therefore, report the numbers achieved on the whole dataset. Recall that we use the laser measurements only for evaluation.

Table 6.1 compares the recall and the accuracy achieved by the baseline SGM model, our pre-trained model used to bootstrap the training and our model after the first and the second training iteration. The accuracy is given as the percentage of pixel within a defined 3D distance to the laser measurements. In our setting one disparity value corresponds to a 3D displacement of 0.55 to 0.72 meters. Each iteration of the training increases both the recall and the overall accuracy. Our final model is able to increase the recall by 16.4 percent points and the accuracy between 2.6 and 12.7 percent points compared to the pre-trained version. This shows that self learning is a suitable option to use deep learning on stereo data without ground truth. Fig. 6.5 visually compares the depth map obtained from the pre-trained network with the one computed with the retrained model. Our model is able to close the gaps in the reconstruction during retraining. A closer inspection reveals that masked regions mainly occur near building-ground edges and correspond to occlusions and, hence, cannot survive the consistency check. This underlines our



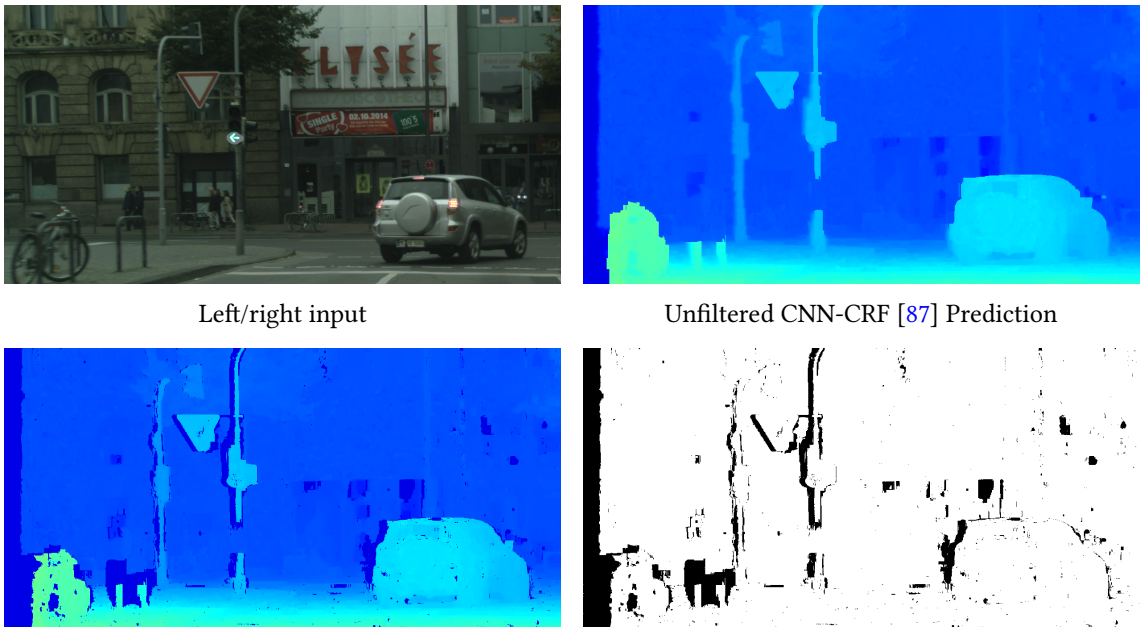
(a) Initial disparity map

(b) Self-learned disparity map

Figure 6.5: Visual comparison of disparity maps. Left: Generated training data. Right: Improved disparity map after retraining. Most of the (dark blue) artefacts are gone after the self-training. Color-coding from cold (small height) to warm (large height).

Model	Recall [%]	Accuracy [%]		
		0.3m	0.5m	1m
SGM	76.0	52.5	69.8	86.7
Pt-Net	87.7	62.9	76.4	87.1
Training 1	92.1	**65.2**	78.6	88.9
Training 2	**92.4**	64.5	**78.7**	**89.3**

Table 6.1: Evaluation of the models and comparison with the laser ground truth. The self-learned models increase the performance on the target domain significantly compared to SGM and the pre-trained network (Pt-Net) used for retraining.



Left/right input | Unfiltered CNN-CRF [87] Prediction

Filtered | Mask

Figure 6.6: Visualization of the two steps *prediction* and *filtering* of the self-learning circle. The white pixels in the mask are used as ground-truth during self-learning and the black pixels are filtered.

findings from Table 6.1, self-training can improve the accuracy and performance and lead to significantly denser reconstructions.

Fig. 6.3 compares the density of points between our computed depth map and the laser depth values projected into the image space. We are able to densely reconstruct the scene, whereas the laser provides only a sparse depth map. Fig. 6.4 shows a detailed reconstruction of our algorithm. In this visualization the color-coding is chosen to highlight high-frequency variations in depth. Here, especially the tower and the arches in the facade of the church prove that our model can deliver highly precise reconstruction from aerial images.

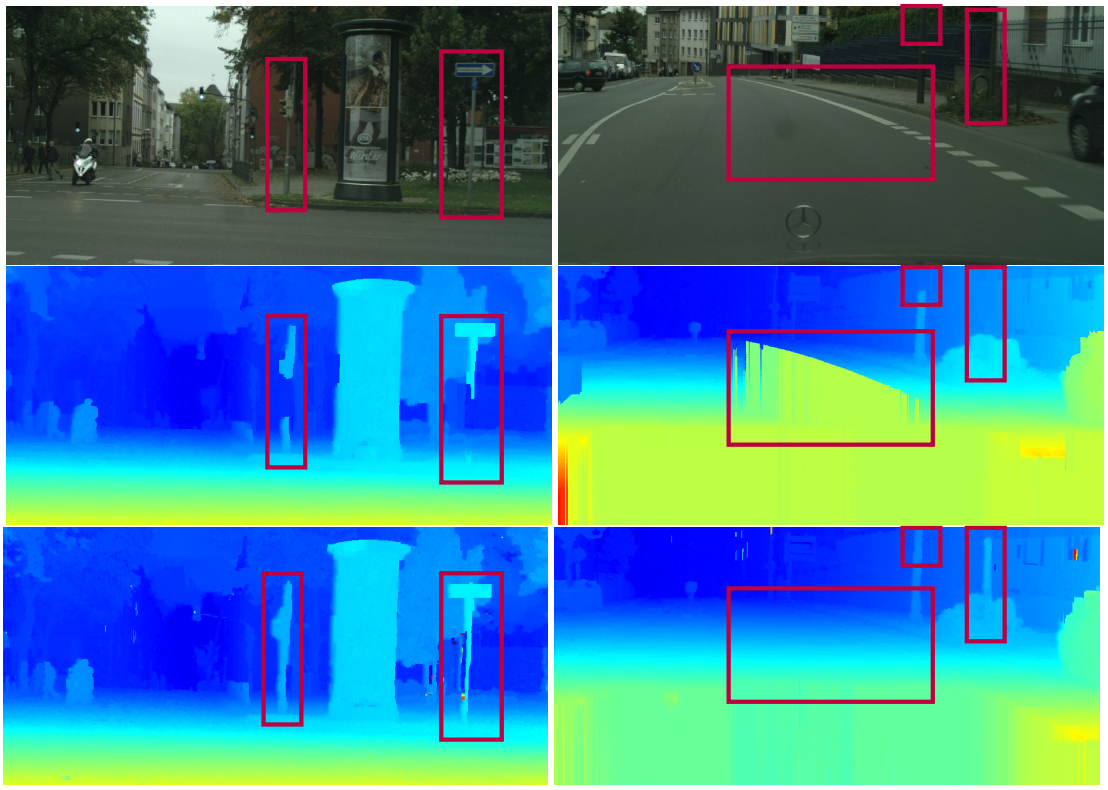


Figure 6.7: Comparison of initial disparity map and self-learned disparity map. From top (input image), to middle (initial disparity map) to bottom (self-learned disparity map). Note especially the highlighted boxes, where the self-learned results removed artifacts on the street and recovered fine details such as the poles.

6.4.2 Cityscapes Dataset

Additionally to the aerial dataset, we present qualitative results on the Cityscapes dataset [39]. This dataset consists of street scenes specifically for autonomous driving. It provides both the left and the right image of the mounted stereo system and thus we can apply the same self-learning procedure as we did for the aerial dataset. However, here we can only conduct a qualitative evaluation, because the Cityscapes dataset does not contain any groundtruth data. Figure 6.6 shows a visualization of the two main steps of the self-learning circle shown in Fig. 6.2. We can see that our left-right filtering step removes occluded pixels as well as other unreliable regions. Figure 6.7 shows the benefit of our self-learning approach. In general we are able to recover finer information compared to the initialization. This can be *e.g.* seen in Fig. 6.7 in the left column, where the we successfully recovered the poles of the traffic signs/lights. The right column shows an example where th initial disparity map contains a large incorrect regions, which can be successfully recovered as visualized in the bottom-right image.

6.5 Conclusion & Future Work

We have shown that, without the requirement of any labeled training data, state-of-the-art machine learning approaches for stereo matching can be used to compute high quality depth maps from aerial images as well as from street images used in autonomous driving. Starting from a pre-trained version, our proposed self-supervised learning framework constructs the training data with a previous version of the learning algorithm itself and additionally relies on conservative consistency checking to reject most of the potential outliers. Our experiments indicate that this concept works for large scale aerial images, whose imaging characteristics are quite far from the initial dataset used for pre-training. Nevertheless, the perceptual quality as well as the raw performance numbers are increased significantly compared to baseline models.

7 Summary and Outlook

In this thesis, we have developed methods for combining deep Convolutional Neural Networks (CNNs) with classical optimization approaches. This allowed us to get the best out of both worlds: The expressiveness and robustness of deep learning and the efficiency and interpretability of optimization. Using this paradigm, we have i) proposed methods to seamlessly combine *CNNs* with discrete optimization problems arising in Conditional Random Fields (CRFs), ii) combined *CNNs* with continuous optimization problems stemming from variational approaches and iii) proposed a method to learn powerful hybrid models without the need of any manually labeled training data.

7.1 Summary

We have shown how a *CRF* can be integrated as the final "layer" into deep learning based on the Structured Support Vector Machine (SSVM). This formulation is theoretically sound and has the advantage that the *CRF* is no longer used as a post-processing step, but fully integrated into the learning process. Therefore, this has allowed us to enable end-to-end learning of the joint model and thus we have been able to get rid of the long chain of manually tuned post-processing steps. We have shown that our formulation allows not only to learn an appropriate data-term for the *CRF*, but additionally also the pairwise term. Both terms can be visualized and thus, we got insights about what our hybrid model learns. The experiments confirmed our expectation: Including prior knowledge via the *CRF* together with end-to-end learning yields significantly smaller models in terms of parameters while maintaining or even outperforming *CNN*-only competitors. The *SSVM* can be used for jointly training *CNN+CRF* models *without* knowing the details of the *CRF* inference algorithm. However, one limitation is that the *SSVM* can only be used if the *CRF* is the last "layer" in the model. To overcome this limitation, we have proposed an orthogonal approach allowing to put *CRF*-layers at *any position* in the *CNN*. To this end, we have derived a generic *CRF* inference framework based on highly efficient dynamic programming building blocks. We have shown how to instantiate a number of different *CRF* inference algorithms using

the dynamic programming routine as a central building block. Thus, we have proposed an efficient algorithm to compute the *exact gradient* for this building block. This enabled us to seamlessly integrate all *CRF* inference algorithms in our framework into deep *CNN* models. In difference to the *SSVM* approach it is here also possible to integrate the *CRF* layer as an intermediate layer in the *CNN*. We have shown that our proposed *CRF* layer can be used generically and demonstrated its benefits on the stereo, optical flow and semantic segmentation tasks. The resulting models are interpretable and significantly smaller than *CNN*-only methods. Hence, we have provided two alternative approaches for training composite *CNN+CRF* models end-to-end. On the one hand the *SSVM* approach has the advantage that we do not need to know details about the inference algorithm, but has the drawbacks that the approach only yields an approximate sub-gradient and the *CRF* can only be used as the last layer in the model. On the other hand, the proposed generic *CRF* framework allows to train the hybrid models with the exact gradient and therefore eases the training. Furthermore, the *CRF* layer can be used at any position in the network including intermediate layers as well. A drawback is the required knowledge about the exact steps of the *CRF* inference algorithm.

The hybrid *CNN+CRF* models described above solve discrete optimization problems and thus the result is naturally a discrete value for every pixel. This is, however, not optimal for the geometrical stereo problem, which is a continuous problem. In order to keep the robustness of the *CRF* in the cost-volume space, we have proposed to learn a continuous-valued refinement on top of the *CRF*. We have therefore shown how to exploit the color image, the initial (discrete) disparity map and the pixel-wise confidences in a collaborative space with a learned variational network. To this end, our model is able to transfer and align object edges and depth discontinuities and yields a refined disparity map as the output. We have demonstrated that the proposed module is generally applicable to different initial disparity maps. Our method is able to remove the discretization artifacts and yields smooth disparity maps as a result.

The methods presented above work very well, however, all of them need a significant amount of training data in order to learn the parameters for the models. Unfortunately, due to the geometric constraints it is not possible to label the training images manually. Thus, generation of labeled training data is a difficult and also expensive problem for the stereo task. We have proposed a method to completely avoid the need of a training data set. To this end, we have shown how to exploit the geometric properties of the stereo problem in order to automatically generate training data. We have demonstrated the applicability of our method on an aerial dataset for which we did not use any ground-truth training data. The experiments have shown that it is possible to learn a high quality stereo model without the use of ground-truth data and that we significantly outperform classical methods even on datasets where no training data is available.

7.2 Conclusion & Outlook

The presented models in this thesis have given deep insights into the strengths and weaknesses of deep learning and classic optimization. Based on that it turned out that i) optimization often provides an elegant and efficient structure and ii) deep learning works amazingly well to calculate

suitable features and representations for a particular problem. Taking these observations into account can also be beneficial for future research. In the following paragraphs, we will point to some interesting directions for future research.

Occlusions Although our models incorporate prior knowledge modeled in a principled way, we did not explicitly model *occlusions* in our models. However, occlusions are an inherent challenge in correspondence problems such as stereo and optical flow. They result from pixels only visible in the reference image but not in the second image. One possibility to include occlusions in a principled way can be realized with a *CRF*. To this end, we can easily include one additional label which we interpret as "occlusion". Furthermore, the *CRF* can also be used to model the geometrical constraints regarding occlusions directly in the inference algorithm. This could be trained similarly as proposed in Chapters 3 and 4.

Discrete Optical Flow Another interesting direction for future research is dense, discrete optical flow. The method proposed in Chapter 4 uses a fixed two-dimensional search window. However, the label space grows quadratically and thus makes optimization intractable. While current state-of-the-art models rely on a coarse-to-find warping scheme, one might consider coarse-to-fine optical flow schemes in the discrete setting as well. Instead of capturing large flow vectors with image warping, the idea is to move the search window with the estimate from the previous level. This has the advantage i) of avoiding the warping artifacts because we sample the image always on the original (non-warped) image and ii) that the search window size can be significantly reduced without limiting the maximal possible displacements. Using *e.g.* a special version of the BP layer could make such a dense, discrete optical flow tractable.

Multi-View Stereo In this thesis we have mainly focused on the canonical 2-view stereo task. However, the proposed methods actually form a very general framework of how graphical models can be trained together with *CNNs*. This concept could also be extended to the general Multi-View-Stereo (MVS) task, where we have multiple views framing the same scene as opposed to the 2-views in the canonical stereo. This is especially interesting, since currently the best performing *MVS* methods do not use learning at all. This is mainly due to the large image sizes and the required view selection. The latter is usually a heuristic in the optimization itself and it is therefore unclear how to integrate it into a learning framework. However, it is clear that learned features and learning in general will significantly improve *MVS*. The vision would be a fully learnable system including *view selection*, *matching*, *regularization* and *fusion* to get a fused point-cloud as the result.

Unsupervised and Self-Supervised Learning One of the main trends in the next few years will be unsupervised and self-supervised learning. The literature has shown that deep models benefit significantly from more training data. However, simply labeling more data is probably not the right way. Instead, statistical tools such as mutual information estimation could pave the way for unsupervised learning to be as powerful as supervised learning.

A

List of Acronyms

Adam	Adaptive Momentum
AI	Artificial Intelligence
ANN	Artificial Neural Network
AutoML	Automated Machine Learning
BCA	Brightness Constancy Assumption
CNN	Convolutional Neural Network
CRF	Conditional Random Field
DL	Deep Learning
DMM	Dual Minorize-Maximize
DNN	Deep Neural Network
DP	Dynamic Programming
ERM	Empirical Risk Minimization
FISTA	Fast Iterative Shrinkage and Thresholding Algorithm
GPU	Graphics Processing Unit
MAP	Maximum-A-Posteriori
MCMC	Markov Chain Monte Carlo
ML	Machine Learning
MRF	Markov Random Field
MVS	Multi-View-Stereo
NAS	Neural Architecture Search
NCC	Normalized Cross Correlation
ODE	Ordinary Differential Equation
OFC	Optical Flow Constraint
PPA	Proximal Point Algorithm
RGB	Red-Green-Blue
SAD	Sum of Absolute Differences
SIFT	Scale Invariant Feature Transform

SSD	Sum of Squared Differences
SSVM	Structured Support Vector Machine
VN	Variational Network
WTA	Winner Takes All

B

List of Publications

My work at the Institute of Computer Graphics and Vision led to the following peer-reviewed publications.

Frame-To-Frame Consistent Semantic Segmentation

Manuel Rebol and **Patrick Knöbelreiter**. *Joint Austrian Computer Vision and Robotics Workshop (OAGM), 2020.*

Abstract: In this work, we aim for temporally consistent semantic segmentation throughout frames in a video. Many semantic segmentation algorithms process images individually which leads to an inconsistent scene interpretation due to illumination changes, occlusions and other variations over time. To achieve a temporally consistent prediction, we train a convolutional neural network (CNN) which propagates features through consecutive frames in a video using a convolutional long short term memory (ConvLSTM) cell. Besides the temporal feature propagation, we penalize inconsistencies in our loss function. We show in our experiments that the performance improves when utilizing video information compared to single frame prediction. The mean intersection over union (mIoU) metric on the Cityscapes validation set increases from 45.2% for the single frames to 57.9% for video data after implementing the ConvLSTM to propagate features trough time on the ESPNet. Most importantly, inconsistency decreases from 4.5% to 1.3% which is a reduction by 71.1%. Our results indicate that the added temporal information produces a frame-to-frame consistent and more accurate image understanding compared to single frame processing.

Belief Propagation Reloaded: Learning BP-Layers for Labeling Problems

Patrick Knöbelreiter, Christian Sormann, Alexander Shekhovtsov, Friedrich Fraundorfer and Thomas Pock. *Conference on Computer Vision and Pattern Recognition (CVPR), 2020.*

Abstract: It has been proposed by many researchers that combining deep neural networks with graphical models can create more efficient and better regularized composite models. The main difficulties in implementing this in practice are associated with a discrepancy in suitable learning objectives as well as with the necessity of approximations for the inference. In this work we

take one of the simplest inference methods, a truncated max-product Belief Propagation, and add what is necessary to make it a proper component of a deep learning model: connect it to learning formulations with losses on marginals and compute the backprop operation. This BP-Layer can be used as the final or an intermediate block in convolutional neural networks (CNNs), allowing us to design a hierarchical model composing BP inference and CNNs at different scale levels. The model is applicable to a range of dense prediction problems, is well-trainable and provides parameter-efficient and robust solutions in stereo, flow and semantic segmentation.

Learned Collaborative Stereo Refinement

Patrick Knöbelreiter and Thomas Pock. *German Conference on Pattern Recognition (GCPR), 2019.*

Abstract: In this work, we propose a learning-based method to denoise and refine disparity maps of a given stereo method. The proposed variational network arises naturally from unrolling the iterates of a proximal gradient method applied to a variational energy defined in a joint disparity, color, and confidence image space. Our method allows to learn a robust collaborative regularizer leveraging the joint statistics of the color image, the confidence map and the disparity map. Due to the variational structure of our method, the individual steps can be easily visualized, thus enabling interpretability of the method. We can therefore provide interesting insights into how our method refines and denoises disparity maps. The efficiency of our method is demonstrated by the publicly available stereo benchmarks Middlebury 2014 and Kitti 2015.

Efficient Multi-Task Learning of Semantic Segmentation and Disparity Estimation

Robert Harb and **Patrick Knöbelreiter**. *Joint Austrian Computer Vision and Robotics Workshop (OAGM), 2019.*

Abstract: We propose a jointly trainable model for semantic segmentation and disparity map estimation. In this work we utilize the fact that the two tasks have complementary strength and weaknesses. Traditional depth prediction algorithms rely on low-level features and often have problems at large textureless regions, while for semantic segmentation these regions are easier to capture. We propose a CNN-based architecture, where both tasks are tightly interconnected to each other. The model consists of an encoding stage which computes features for both tasks, semantic segmentation and disparity estimation. In the decoding stage we explicitly add the semantic predictions to the disparity decoding branch and we additionally allow to exchange information in the intermediate feature representations. Furthermore, we set the focus on efficiency, which we achieve by the usage of previously introduced ESP building blocks. We evaluate the model on the commonly used KITTI dataset.

Learning Energy Based Inpainting for Optical Flow

Christoph Vogel, **Patrick Knöbelreiter** and Thomas Pock. *Asian Conference on Computer Vision (ACCV), 2018.*

Abstract: Modern optical flow methods are often composed of a cascade of many independent steps or formulated as a black box neural network that is hard to interpret and analyze. In this

work we seek for a plain, interpretable, but learnable solution. We propose a novel inpainting based algorithm that approaches the problem in three steps: feature selection and matching, selection of supporting points and energy based inpainting. To facilitate the inference we propose an optimization layer that allows to backpropagate through 10K iterations of a first-order method without any numerical or memory problems. Compared to recent state-of-the-art networks, our modular CNN is very lightweight and competitive with other, more involved, inpainting based methods.

Self-Supervised Learning for Stereo Reconstruction on Aerial Images

Patrick Knöbelreiter, Christoph Vogel and Thomas Pock. *International Geoscience and Reomte Sensing Symposium (IGARSS), 2018.*

Abstract: Recent developments established deep learning as an inevitable tool to boost the performance of dense matching and stereo estimation. On the downside, learning these networks requires a substantial amount of training data to be successful. Consequently, the application of these models outside of the laboratory is far from straight forward. In this work we propose a self-supervised training procedure that allows us to adapt our network to the specific (imaging) characteristics of the dataset at hand, without the requirement of external ground truth data. We instead generate interim training data by running our intermediate network on the whole dataset, followed by conservative outlier filtering. Bootstrapped from a pre-trained version of our hybrid CNN-CRF model, we alternate the generation of training data and network training. With this simple concept we are able to lift the completeness and accuracy of the pre-trained version significantly. We also show that our final model compares favorably to other popular stereo estimation algorithms on an aerial dataset.

Robot Localisation and 3D Position Estimation using a Free-Moving Camera and Cascaded Convolutional Neural Networks

Justinas Miseikis, **Patrick Knöbelreiter**, Inka Brijacak, Saeed Yahyanejad, Kyrre Glette, Ole Jakob Elle and Jim Torresen. *International Conference on Advanced Intelligent Mechatronics (AIM), 2018.*

Abstract: Many works in collaborative robotics and humanrobot interaction focuses on identifying and predicting human behaviour while considering the information about the robot itself as given. This can be the case when sensors and the robot are calibrated in relation to each other and often the reconfiguration of the system is not possible, or extra manual work is required. We present a deep learning based approach to remove the constraint of having the need for the robot and the vision sensor to be fixed and calibrated in relation to each other. The system learns the visual cues of the robot body and is able to localise it, as well as estimate the position of robot joints in 3D space by just using a 2D color image. The method uses a cascaded convolutional neural network, and we present the structure of the network, describe our own collected dataset, explain the network training and achieved results. A fully trained system shows promising results in providing an accurate mask of where the robot is located and a good estimate of its joints positions in 3D. The accuracy is not good enough for visual servoing applications yet, however, it can be

sufficient for general safety and some collaborative tasks not requiring very high precision. The main benefit of our method is the possibility of the vision sensor to move freely. This allows it to be mounted on moving objects, for example, a body of the person or a mobile robot working in the same environment as the robots are operating in.

Scalable Full Flow with Learned Binary Descriptors

Gottfried Munda, Alexander Shekhovtsov, **Patrick Knöbelreiter** and Thomas Pock. *German Conference on Pattern Recognition (GCPR), 2018.*

Abstract: We propose a method for large displacement optical flow in which local matching costs are learned by a convolutional neural network (CNN) and a smoothness prior is imposed by a conditional random field (CRF). We tackle the computation- and memory-intensive operations on the 4D cost volume by a min-projection which reduces memory complexity from quadratic to linear and binary descriptors for efficient matching. This enables evaluation of the cost on the fly and allows to perform learning and CRF inference on high resolution images without ever storing the 4D cost volume. To address the problem of learning binary descriptors we propose a new hybrid learning scheme. In contrast to current state of the art approaches for learning binary CNNs we can compute the exact non-zero gradient within our model. We compare several methods for training binary descriptors and show results on public available benchmarks.

End-to-end Training of Hybrid CNN-CRF Models for Stereo

Patrick Knöbelreiter, Christian Reinbacher, Alexander Shekhovtsov and Thomas Pock. *Conference on Computer Vision and Pattern Recognition (CVPR), 2017.*

Abstract: We propose a novel and principled hybrid CNN+ CRF model for stereo estimation. Our model allows to exploit the advantages of both, convolutional neural networks (CNNs) and conditional random fields (CRFs) in an unified approach. The CNNs compute expressive features for matching and distinctive color edges, which in turn are used to compute the unary and binary costs of the CRF. For inference, we apply a recently proposed highly parallel dual block descent algorithm which only needs a small fixed number of iterations to compute a high-quality approximate minimizer. As the main contribution of the paper, we propose a theoretically sound method based on the structured output support vector machine (SSVM) to train the hybrid CNN+CRF model on large-scale data end-to-end. Our trained models perform very well despite the fact that we are using shallow CNNs and do not apply any kind of post-processing to the final output of the CRF. We evaluate our combined models on challenging stereo benchmarks such as Middlebury 2014 and Kitti 2015 and also investigate the performance of each individual component.

End-to-end Training of Hybrid CNN-CRF Models for Semantic Segmentation using Structured Learning

Aleksander Colovic, **Patrick Knöbelreiter**, Alexander Shekhovtsov and Thomas Pock. *Computer Vision Winter Workshop (CVWW), 2017.*

Abstract: In this work we tackle the problem of semantic image segmentation with a combination of convolutional neural networks (CNNs) and conditional random fields (CRFs). The CRF takes contrast sensitive weights in a local neighborhood as input (pairwise interactions) to encourage consistency (smoothness) within the prediction and align our segmentation boundaries with visual edges. We model unary terms with a CNN which outperforms non data driven models. We approximate the CRF inference with a fixed number of iterations of a linearprogramming relaxation based approach. We experiment with training the combined model end-to-end using a discriminative formulation (structured support vector machine) and applying stochastic subgradient descend to it.

Learning joint Demosaicing and Denoising based on Sequential Energy Minimization

Teresa Klatzer, Kerstin Hammernik, **Patrick Knöbelreiter** and Thomas Pock. *International Conference on Computational Photography (ICCP), 2016.*

Abstract: Demosaicing is an important first step for color image acquisition. For practical reasons, demosaicing algorithms have to be both efficient and yield high quality results in the presence of noise. The demosaicing problem poses several challenges, e.g. zippering and false color artifacts as well as edge blur. In this work, we introduce a novel learning based method that can overcome these challenges. We formulate demosaicing as an image restoration problem and propose to learn efficient regularization inspired by a variational energy minimization framework that can be trained for different sensor layouts. Our algorithm performs joint demosaicing and denoising in close relation to the real physical mosaicing process on a camera sensor. This is achieved by learning a sequence of energy minimization problems composed of a set of RGB filters and corresponding activation functions. We evaluate our algorithm on the Microsoft Demosaicing data set in terms of peak signal to noise ratio (PSNR) and structured similarity index (SSIM). Our algorithm is highly efficient both in image quality and run time. We achieve an improvement of up to 2.6 dB over recent state-of-the-art algorithms.

Bibliography

[1] K. Alahari, C. Russell, and P. H. S. Torr. Efficient piecewise learning for conditional random fields. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. (page 85, 107)

[2] Samuel Audet, Yoriyuki Kitta, Yuta Noto, Ryo Sakamoto, and Takagi Akihiro. libsgm. `https://github.com/fixstars/libSGM`. (page 167)

[3] Christian Bailer, Kiran Varanasi, and Didier Stricker. CNN based patch matching for optical flow with thresholded hinge loss. *CoRR*, 2016. URL `http://arxiv.org/abs/1607.08064`. (page 85)

[4] Jonathan T. Barron and Ben Poole. The fast bilateral solver. In *European Conference on Computer Vision (ECCV)*, pages 617–632, 2016. (page 96, 138, 146, 148)

[5] Amir Beck. *First-order methods in optimization*. SIAM, 2017. (page 2, 45)

[6] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sciences*, pages 183–202, 2009. (page 56, 138)

[7] Richard Bellman. Dynamic programming. *Science*, pages 34–37, 1966. (page 25)

[8] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8): 1798–1828, 2013. (page 64)

[9] Kristin P Bennett and Emilio Parrado-Hernández. The interplay of optimization and machine learning research. *Journal of Machine Learning Research*, 7(Jul):1265–1281, 2006. (page 2)

[10] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math expression compiler. In *Python for Scientific Computing Conference*, 2010. (page 93)

[11] Stan Birchfield and Carlo Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 401–406, 1998. (page 83)

[12] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. (page 13, 18, 65)

[13] Maros Blaha, Christoph Vogel, Audrey Richard, Jan Dirk Wegner, Thomas Pock, and Konrad Schindler. Large-scale semantic 3d reconstruction: An adaptive multi-resolution model for multi-class volumetric labeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (page 163)

[14] Andrew Blake, Pushmeet Kohli, and Carsten Rother. *Markov random fields for vision and image processing*. Mit Press, 2011. (page 18)

[15] Michael Bleyer and Margrit Gelautz. Simple but effective tree structures for dynamic programming-based stereo matching. In *In VISAPP*, pages 415–422, 2008. (page 107, 111, 114)

[16] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. (page 45)

[17] Yuri Boykov and Marie-Pierre Jolly. Interactive organ segmentation using graph cuts. In *Medical Image Computing and Computer-Assisted Intervention*, pages 276–286, 2000. URL `http://www.csd.uwo.ca/faculty/yuri/Papers/miccai00.pdf`. (page 86)

[18] Yuri Boykov and Marie-Pierre Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *IEEE International Conference on Computer Vision (ICCV)*, pages 105–112, 2001. URL `http://www.csd.uwo.ca/faculty/yuri/Papers/iccv01.pdf`. (page 83)

[19] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1124–1137, 2004. (page 44)

[20] Yuri Boykov, Olga Veksler, and Ramin Zabih. Markov random fields with efficient approximations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 648–655, 1998. (page 44)

[21] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1222–1239, 2001. (page 44)

[22] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a siamese time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 7 (04):669–688, 1993. (page 84)

[23] G. Brown, M. Hua and Winder S. Discriminative learning of local image descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010. (page 85)

[24] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *European Conference on Computer Vision (ECCV)*, pages 25–36, 2004. (page 59, 137, 138)

[25] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *European Conference on Computer Vision (ECCV)*, pages 611–625, 2012. (page 127)

[26] Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Proceedings of Advances in Neural Information Processing Systems*, pages 402–408, 2001. (page 1)

[27] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, pages 120–145, 2011. (page 56, 57, 58, 137)

[28] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5410–5418, 2018. (page 119, 120, 123, 126, 127, 152)

[29] L.-C. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun. Learning Deep Structured Models. In *ICML*, 2015. (page 85, 86, 89, 91)

[30] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014. (page 84)

[31] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015. URL `http://arxiv.org/abs/1412.7062`. (page 44, 108)

[32] Yunjin Chen, Wei Yu, and Thomas Pock. On learning optimized reaction diffusion processes for effective image restoration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5261–5269, 2015. (page 139, 142, 155)

[33] Zhuoyuan Chen, Xun Sun, Liang Wang, Yinan Yu, and Chang Huang. A deep visual correspondence embedding model for stereo matching costs. In *IEEE International Conference on Computer Vision (ICCV)*, pages 972–980, 2015. (page 82, 83, 84, 88)

[34] Xinjing Cheng, Peng Wang, and Ruigang Yang. Learning depth with convolutional spatial propagation network. *CoRR*, abs/1810.02695, 2018. (page 108)

[35] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1251–1258, 2017. (page 72)

[36] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv*, 2015. (page 70)

[37] Peter Clifford. Markov random fields in statistics. *Disorder in physical systems: A volume in honour of John M. Hammersley*, 19, 1990. (page 19)

[38] Gregory F Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405, 1990. (page 29)

[39] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (page 62, 170)

[40] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (page 129, 130, 131, 132)

[41] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. (page 1)

[42] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. (page 1)

[43] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1422–1430, 2015. (page 64)

[44] Justin Domke. Parameter learning with truncated message-passing. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2937–2943, 2011. (page 107, 108)

[45] Justin Domke. Learning graphical model parameters with approximate marginal inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(10):2454–2467, October 2013. (page 107, 108, 109)

[46] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. (page 127)

[47] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, 2015. (page 65)

[48] A. Dosovitskiy, P. Fischery, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, Dec 2015. (page 84)

[49] Frederik Eaton and Zoubin Ghahramani. Choosing a variable to clamp: Approximate inference using conditioned belief propagation. In *International Conference on Artificial Intelligence and Statistics*, volume 5, pages 145–152, April 2009. (page 108)

[50] Alexander Effland, Erich Kobler, Karl Kunisch, and Thomas Pock. An optimal control approach to early stopping variational methods for image restoration. In *Journal of Mathematical Imaging and Vision*, 2019. (page 155)

[51] Gal Elidan, Ian McGraw, and Daphne Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. *arXiv preprint arXiv:1206.6837*, 2012. (page 40)

[52] Gabriele Facciolo, Carlo de Franchis, and Enric Meinhardt. MGM: A significantly more global matching for stereovision. In *British Machine Vision Conference*, 2015. (page 83, 84, 86)

[53] Pedro F Felzenszwalb and Daniel P Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79, 2005. (page 36)

[54] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient belief propagation for early vision. *International Journal of Computer Vision*, 70(1):41–54, 2006. (page 39)

[55] Pedro F Felzenszwalb and Olga Veksler. Tiered scene labeling with dynamic programming. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3097–3104, 2010. (page 129)

[56] Vojtěch Franc and Pavel Laskov. Learning maximal margin markov networks via tractable convex optimization. *Control Systems and Computers*, pages 25–34, April 2011. (page 85)

[57] Pascal Fua. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine vision and applications*, 6(1):35–49, 1993. (page 64)

[58] S Gehrke A, K Morin B, M Downey A, N Boehrer C, and T Fuchs C. Semi-global matching: An alternative to lidar for dsm generation? 2012. (page 163)

[59] Willard J. Gibbs. *Elementary principles in statistical mechanics*. Cambridge University Press, 1902. (page 24)

[60] Spyros Gidaris and Nikos Komodakis. Detect, replace, refine: Deep structured prediction for pixel wise labeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5248–5257, 2017. (page 138)

[61] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018. (page 64)

[62] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011. (page 70)

[63] Allan Gut. *Probability: a graduate course*, volume 75. Springer Science & Business Media, 2013. (page 11)

[64] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009. (page 2)

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. (page 70)

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016. (page 70, 73, 139)

[67] Tom Heskes. Convexity arguments for efficient minimization of the bethe and kikuchi free energies. *Journal of Artificial Intelligence Research*, 26:153–190, 2006. (page 40)

[68] Heiko Hirschmüller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 807–814, 2005. (page 83, 84, 163, 167)

[69] Heiko Hirschmüller. Semi-global matching-motivation, developments and applications. *Photogrammetric Week*, 2011. (page 86)

[70] Heiko Hirschmüller, Maximilian Buder, and Ines Ernst. Memory efficient semi-global matching. In *International Society for Photogrammetry and Remote Sensing (ISPRS)*, 2012. (page 163)

[71] Heiko Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, February 2008. (page 4, 107, 108, 111, 117)

[72] Berthold KP Horn and Brian G Schunck. Determining optical flow. In *Techniques and Applications of Image Understanding*, volume 281, pages 319–331. International Society for Optics and Photonics, 1981. (page 3, 57, 58)

[73] X. Hu and P. Mordohai. A quantitative evaluation of confidence measures for stereo vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 2121–2133, 2012. (page 144)

[74] E. Ilg, T. Saikia, M. Keuper, and T. Brox. Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation. In *European Conference on Computer Vision (ECCV)*, 2018. (page 127)

[75] Hiroshi Ishikawa. Exact optimization for markov random fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1333–1336, 2003. (page 44)

[76] Sham Kakade, Yee Whye Teh, and Sam T. Roweis. An alternate objective function for markovian fields, 2002. (page 107)

[77] Jörg H. Kappes, Bjoern Andres, Fred A. Hamprecht, Christoph Schnörr, Sebastian Nowozin, Dhruv Batra, Sungwoong Kim, Bernhard X. Kausler, Jan Lellmann, Nikos Komodakis, and Carsten Rother. A comparative study of modern inference techniques for discrete energy minimization problem. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013. (page 44)

[78] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (page 4, 117, 119, 120, 122)

[79] Joseph Keshet. *Optimizing the Measure of Performance*. 2014. (page 107)

[80] Sameh Khamis, Sean Fanello, Christoph Rhemann, Adarsh Kowdle, Julien Valentin, and Shahram Izadi. Stereonet: Guided hierarchical refinement for real-time edge-aware depth prediction. In *European Conference on Computer Vision (ECCV)*, pages 8–14, 2018. (page 6, 110, 117, 123, 137, 138, 146, 148, 150)

[81] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (page 77, 78, 119, 146)

[82] Alexander Kirillov, Dmitrij Schlesinger, Walter Forkel, Anatoly Zelenin, Shuai Zheng, Philip H. S. Torr, and Carsten Rother. Efficient likelihood learning of a generic CNN-CRF model for semantic segmentation. *CoRR*, 2015. URL `http://arxiv.org/abs/1511.05067`. (page 85, 107)

[83] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 971–980, 2017. (page 70)

[84] Patrick Knöbelreiter and Thomas Pock. Learned collaborative stereo refinement. In *German Conference on Pattern Recognition (GCPR)*, 2019. (page 6, 7, 63, 64, 108, 137)

[85] Patrick Knöbelreiter, Christian Reinbacher, Alexander Shekhovtsov, and Thomas Pock. End-to-end training of hybrid cnn-crf models for stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2339–2348, 2017. (page 63, 64, 107, 123, 146, 152, 153)

[86] Patrick Knöbelreiter, Christoph Vogel, and Thomas Pock. Self-supervised learning for stereo reconstruction on aerial images. In *International Geoscience and Remote Sensign Symposium*, 2018. (page 64)

[87] Patrick Knöbelreiter, Christian Reinbacher, Alexander Shekhovtsov, and Thomas Pock. End-to-End Training of Hybrid CNN-CRF Models for Stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. (page 7, 21, 105, 135, 164, 165, 169)

[88] Patrick Knöbelreiter, Christian Sormann, Alexander Shekhovtsov, Friedrich Fraundorfer, and Thomas Pock. Belief Propagation Reloaded: Learning BP-Layers for Labeling Problems. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. (page 37)

[89] Erich Kobler, Teresa Klatzer, Kerstin Hammernik, and Thomas Pock. Variational networks: Connecting variational methods and deep learning. In *German Conference on Pattern Recognition (GCPR)*, pages 281–293, 2017. (page 139, 142)

[90] Pushmeet Kohli and Philip H. S. Torr. Measuring uncertainty in graph cut solutions – efficiently computing min-marginal energies using dynamic graph cuts. In *European Conference on Computer Vision (ECCV)*, pages 30–43. Springer-Verlag, 2006. (page 110)

[91] Pushmeet Kohli, Ladicky Lubor, and Philip H.S. Torr. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision*, 82(3):302–324, 2009. (page 22)

[92] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009. (page 18)

[93] V. Kolmogorov and R. Zabih. *Graph Cut Algorithms for Binocular Stereo with Occlusions*, pages 423–437. Springer US, 2006. (page 82)

[94] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006. (page 40, 86)

[95] Vladimir Kolmogorov, Pascal Monasse, and Pauline Tan. Kolmogorov and Zabih's Graph Cuts Stereo Matching Algorithm. *Image Processing On Line*, 4:220–251, 2014. doi: 10.5201/ipol.2014.97. (page 117, 120)

[96] N. Komodakis, N. Paragios, and G. Tziritas. MRF optimization via dual decomposition: Message-passing revisited. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, 2007. (page 43, 87)

[97] Nikos Komodakis. Efficient training for pairwise or higher order CRFs via dual decomposition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1841–1848, 2011. (page 85, 89, 91)

[98] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Proceedings of Advances in Neural Information Processing Systems*, pages 109–117, 2011. (page 22, 23)

[99] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Proceedings of Advances in Neural Information Processing Systems*, 2012. (page 84)

[100] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1097–1105, 2012. (page 1, 65)

[101] Georg Kuschk and Daniel Cremers. Fast and accurate large-scale stereo reconstruction using variational methods. In *IEEE International Conference on Computer Vision Workshop*, pages 700–707, 2013. (page 138)

[102] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001. (page 21)

[103] Emanuel Laude, Thomas Möllenhoff, Michael Moeller, Jan Lellmann, and Daniel Cremers. Sublabel-accurate convex relaxation of vectorial multilabel energies. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, pages 614–627, Cham, 2016. Springer International Publishing. (page 82)

[104] Steffen L. Lauritzen. *Graphical Models*. Number 17 in Oxford Statistical Science Series. Oxford Science Publications, 1998. ISBN 0-19-852219-3. (page 109)

[105] 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. *Neural Computation*, 1(4): 541–551, Dec 1989. (page 1, 65, 76)

[106] Yann LeCun and C. Cortes. The mnist database of handwritten digits, 1998. URL `http://yann.lecun.com/exdb/mnist/`. (page 61)

[107] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. (page 71)

[108] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012. (page 70)

[109] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial intelligence and statistics*, pages 562–570, 2015. (page 118)

[110] JP Lewis. Fast normalized cross-correlation. (page 4)

[111] Mengtian Li, Alexander Shekhovtsov, and Daniel Huber. Complexity of discrete energy minimization problems. In *European Conference on Computer Vision (ECCV)*, pages 834–852, 2016. (page 86)

[112] Zhengfa Liang, Yiliu Feng, Yulan Guo, Hengzhu Liu, Wei Chen, Linbo Qiao, Li Zhou, and Jianfeng Zhang. Learning for disparity estimation through feature constancy. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2811–2820, 2018. (page 138)

[113] Guosheng Lin, Chunhua Shen, Ian D. Reid, and Anton van den Hengel. Efficient piecewise training of deep structured models for semantic segmentation. *CoRR*, 2015. URL `http://arxiv.org/abs/1504.01013`. (page 85, 107)

[114] Sifei Liu, Shalini De Mello, Jinwei Gu, Guangyu Zhong, Ming-Hsuan Yang, and Jan Kautz. Learning affinity via spatial propagation networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1520–1530. 2017. (page 108)

[115] Ziwei Liu, Xiaoxiao Li, Ping Luo, Chen-Change Loy, and Xiaoou Tang. Semantic image segmentation via deep parsing network. In *IEEE International Conference on Computer Vision (ICCV)*, December 2015. (page 85)

[116] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015. (page 44, 65, 73, 147)

[117] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. (page 5)

[118] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981. (page 6, 57, 58)

[119] W. Luo, A. Schwing, and R. Urtasun. Efficient deep learning for stereo matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (page 82, 83, 84, 88, 94, 95, 97, 98, 101, 163)

[120] Wenjie Luo, Alexander G Schwing, and Raquel Urtasun. Efficient deep learning for stereo matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5695–5703, 2016. (page 7, 123)

[121] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013. (page 70)

[122] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013. (page 70)

[123] Dmitrii Marin, Meng Tang, Ismail Ben Ayed, and Yuri Boykov. Beyond gradient descent for regularized segmentation losses. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. (page 44, 108)

[124] Daniel Maurer, Michael Stoll, and Andrés Bruhn. Order-adaptive and illumination-aware variational optical flow refinement. In *British Machine Vision Conference*, 2017. (page 138)

[125] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (page 121)

[126] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. (page 84, 95, 98, 163)

[127] John McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon. A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, 1955. URL `https://web.archive.org/web/20080930164306/http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html`. (page 59)

[128] Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda Shapiro, and Hannaneh Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *European Conference on Computer Vision (ECCV)*, 2018. (page 118, 129, 130, 131)

[129] Tim Meinhardt, Michael Moeller, Caner Hazirbas, and Daniel Cremers. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (page 139)

[130] Talya Meltzer, Amir Globerson, and Yair Weiss. Convergent message passing algorithms-a unifying view. *arXiv preprint arXiv:1205.2625*, 2012. (page 40)

[131] Arthur Mensch and Mathieu Blondel. Differentiable dynamic programming for structured prediction and attention, 2018. (page 108)

[132] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3061–3070, 2015. (page 1, 5, 94, 96, 146)

[133] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969. (page 67)

[134] T. Möllenhoff, E. Laude, M. Moeller, J. Lellmann, and D. Cremers. Sublabel-accurate relaxation of nonconvex energies. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (page 102)

[135] Gottfried Munda, Alexander Shekhovtsov, Patrick Knöbelreiter, and Thomas Pock. Scalable full flow with learned binary descriptors. In *German Conference on Pattern Recognition (GCPR)*, pages 321–332, 2017. (page 118)

[136] Kevin Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. *arXiv preprint arXiv:1301.6725*, 2013. (page 37)

[137] Kevin P Murphy. A probabilistic perspective. *Text book*, 2012. (page 59)

[138] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o(1/k2). *Doklady Akademii Nauk*, 269:543–547, 1983. (page 53, 54)

[139] Yurii Nesterov. On an approach to the construction of optimal methods of minimization of smooth convex functions. *Ekonomika i Mateaticheskie Metody*, 24(3):509–517, 1988. (page 156)

[140] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2004. (page 2, 54)

[141] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision (ECCV)*, pages 69–84, 2016. (page 64)

[142] Sebastian Nowozin. Constructing composite likelihoods in general random fields. In *ICML Workshop on Infering: Interactions between Inference and Learning*, July 2013. URL `https://www.microsoft.com/en-us/research/publication/constructing-composite-likelihoods-in-general-random-fields/`. (page 85)

[143] Sebastian Nowozin and Christoph H Lampert. *Structured learning and prediction in computer vision*. Now publishers Inc, 2011. (page 18, 35, 44)

[144] P. Ochs, R. Ranftl, T. Brox, and T. Pock. Techniques for Gradient Based Bilevel Optimization with Nonsmooth Lower Level Problems. *Journal of Mathematical Imaging and Vision*, 2016. (page 85)

[145] Peter Ochs, René Ranftl, Thomas Brox, and Thomas Pock. Bilevel optimization with nonsmooth lower level problems. In Jean-François Aujol, Mila Nikolova, and Nicolas Papadakis, editors, *Scale Space and Variational Methods in Computer Vision*, pages 654–665. Springer International Publishing, 2015. (page 85)

[146] Christopher J. Pal, Jerod J. Weinman, Lam C. Tran, and Daniel Scharstein. On learning conditional random fields for stereo - exploring model structures and approximate inference. *International Journal of Computer Vision*, 99(3):319–337, 2012. (page 84, 85, 107)

[147] Jiahao Pang, Wenxiu Sun, Jimmy SJ. Ren, Chengxi Yang, and Qiong Yan. Cascade residual learning: A two-stage convolutional neural network for stereo matching. In *IEEE International Conference on Computer Vision Workshop*, pages 887–895, 2017. (page 138)

[148] G. Papandreou and A. Yuille. Perturb-and-map random fields: Reducing random sampling to optimization, with applications in computer vision. In S. Nowozin, P.V. Gehler, J. Jancsary, and C.H. Lampert, editors, *Advanced Structured Prediction*. MIT-Press, 2014. (page 110)

[149] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, pages 127–239, 2014. (page 141)

[150] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2536–2544, 2016. (page 63)

[151] Judea Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In *AAAI*, pages 133–136, 1982. (page 109)

[152] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-73-7. (page 26, 110)

[153] Thomas Pock and Antonin Chambolle. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1762–1769, 2011. (page 57)

[154] Matteo Poggi, Fabio Tosi, and Stefano Mattoccia. Learning from scratch a confidence measure. In *British Machine Vision Conference*, 2016. (page 164)

[155] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. (page 53)

[156] Eric T. Psota, Jedrzej Kowalczuk, Mateusz Mittek, and Lance C. Perez. Map disparity estimation using hidden Markov trees. In *IEEE International Conference on Computer Vision (ICCV)*, December 2015. (page 84)

[157] Rongjun Qin, Jiaojiao Tian, and Peter Reinartz. 3d change detection – approaches and applications. In *International Society for Photogrammetry and Remote Sensing (ISPRS)*, 2016. (page 163)

[158] R. Ranftl, S. Gehrig, T. Pock, and H. Bischof. Pushing the limits of stereo using variational stereo estimation. In *IEEE Intelligent Vehicles Symposium*, pages 401–407, 2012. (page 138)

[159] René Ranftl and Thomas Pock. A deep variational model for image segmentation. In Xiaoyi Jiang, Joachim Hornegger, and Reinhard Koch, editors, *German Conference on Pattern Recognition (GCPR)*, pages 107–118. Springer International Publishing, 2014. (page 85)

[160] René Ranftl, Kristian Bredies, and Thomas Pock. Non-local total generalized variation for optical flow estimation. In *European Conference on Computer Vision (ECCV)*, pages 439–454, 2014. (page 82, 138)

[161] Jérôme Revaud, Philippe Weinzaepfel, Zaïd Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1164–1172, 2015. (page 138)

[162] Gernot Riegler, Matthias Rüther, and Horst Bischof. Atgv-net: Accurate depth super-resolution. In *European Conference on Computer Vision (ECCV)*, pages 268–284, 2016. (page 108, 138)

[163] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, 2015. (page 73, 147)

[164] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. In *Psychological Reviews*, 1958. (page 65, 67)

[165] Stefan Roth and Michael J Black. Fields of experts. *International Journal of Computer Vision*, 2009. (page 140)

[166] Franz Rottensteiner, Gunho Sohn, Markus Gerke, and Jan Dirk Wegner. Isprs test project on urban classification and 3d building reconstruction. *Commission III-Photogrammetric Computer Vision and Image Analysis*, 2013. (page 167)

[167] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002. (page 59)

[168] Bogdan Savchynskyy. Discrete graphical models–an optimization perspective. *arXiv preprint arXiv:2001.09017*, 2020. (page 18, 28, 42)

[169] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nesic, X. Wang, and P. Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In *German Conference on Pattern Recognition (GCPR)*, pages 31–42, 2014. (page 7, 21, 62, 63, 94, 96, 147, 165)

[170] Daniel Scharstein. Learning conditional random fields for stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007. (page 84, 85)

[171] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 2002. (page 3, 5, 39, 82, 136)

[172] Michail I Schlesinger. Syntactic analysis of two-dimensional visual signals in noisy conditions. *Cybernetics*, 12(4):612–628, 1976. (page 44)

[173] Alexander G. Schwing and Raquel Urtasun. Fully connected deep structured networks. *CoRR*, 2015. URL `http://arxiv.org/abs/1503.02351`. (page 85)

[174] Akihito Seki and Marc Pollefeys. Patch based confidence prediction for dense disparity map. In *British Machine Vision Conference*, volume 10, 2016. (page 95)

[175] Akihito Seki and Marc Pollefeys. Sgm-nets: Semi-global matching with neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. (page 108)

[176] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014. (page 59)

[177] Alexander Shekhovtsov, Christian Reinbacher, Gottfried Graber, and Thomas Pock. Solving dense image matching in real-time using discrete-continuous optimization. In *Computer Vision Winter Workshop*, 2016. (page 83, 84, 86)

[178] Alexander Shekhovtsov, Christian Reinbacher, Gottfried Graber, and Thomas Pock. Solving dense image matching in real-time using discrete-continuous optimization. *Computer Vision Winter Workshop*, 2016. (page 43, 44, 138)

[179] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative Learning of Deep Convolutional Feature Point Descriptors. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. (page 83)

[180] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. (page 70, 73)

[181] David Sontag and Tommi Jaakkola. Tree block coordinate descent for MAP in graphical models. In *Artificial Intelligence and Statistics*, pages 544–551, 2009. (page 114, 115, 116)

[182] David Sontag, Talya Meltzer, Amir Globerson, Tommi S Jaakkola, and Yair Weiss. Tightening lp relaxations for map using message passing. *arXiv preprint arXiv:1206.3288*, 2012. (page 40)

[183] Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7):787–800, 2003. (page 39)

[184] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3104–3112, 2014. (page 65)

[185] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):1068–1080, June 2008. (page 37, 44, 111)

[186] Marshall Tappen and William T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical mrf parameters. In *IEEE International Conference on Computer Vision (ICCV)*, pages 900–906, 2003. (page 37, 44, 107, 110, 111, 114)

[187] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 25–32, 2004. (page 89)

[188] Theano Development Team. Convolution arithmetic, 2020. URL `http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html`. (page 71)

[189] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *IEEE International Conference on Computer Vision (ICCV)*, pages 839–846. IEEE, 1998. (page 5)

[190] Jonathan J. Tompson, Arjun Jain, Yann Lecun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In Z. Ghahramani, M. Welling, C. Cortes, N.d. Lawrence, and K.q. Weinberger, editors, *Proceedings of Advances*

in Neural Information Processing Systems, pages 1799–1807. Curran Associates, Inc., 2014. (page 85)

[191] Alessio Tonioni, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. Unsupervised adaptation for deep stereo. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (page 164)

[192] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005. URL `http://dl.acm.org/citation.cfm?id=1046920.1088722`. (page 85, 89)

[193] Stepan Tulyakov, Anton Ivanov, and Francois Fleuret. Weakly supervised learning of deep metrics for stereo reconstruction. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (page 164)

[194] Stepan Tulyakov, Anton Ivanov, and Francois Fleuret. Practical deep stereo (pds): Toward applications-friendly deep stereo matching. In *Proceedings of Advances in Neural Information Processing Systems*, pages 5871–5881, 2018. (page 122, 123, 136, 152)

[195] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, pages 1747–1756, 2016. (page 108)

[196] Raviteja Vemulapalli, Oncel Tuzel, Ming-Yu Liu, and Rama Chellapa. Gaussian conditional random field network for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3224–3233, 2016. (page 22, 108)

[197] SVN Vishwanathan, Nicol N Schraudolph, Mark W Schmidt, and Kevin P Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *ICML*, pages 969–976, 2006. (page 44)

[198] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967. (page 28)

[199] Christoph Vogel and Thomas Pock. A primal dual network for low-level vision problems. In *German Conference on Pattern Recognition (GCPR)*, 2017. (page 139)

[200] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, 115(1):1–28, 2015. (page 1)

[201] Christoph Vogel, Patrick Knöbelreiter, and Thomas Pock. Learning energy based inpainting for optical flow. In *Asian Conference on Computer Vision (ACCV)*, 2018. (page 108, 138)

[202] Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008. (page 40, 44)

[203] Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. Tree-reweighted belief propagation algorithms and approximate ml estimation by pseudo-moment matching. In *International Conference on Artificial Intelligence and Statistics*, 2003. (page 114, 115)

[204] Shenlong Wang, Sanja Fidler, and Raquel Urtasun. Proximal deep structured models. In *Proceedings of Advances in Neural Information Processing Systems*, pages 865–873, 2016. (page 139)

[205] Jerod J Weinman, Lam Tran, and Christopher J Pal. Efficiently learning random fields for stereo vision with sparse message passing. In *European Conference on Computer Vision (ECCV)*, pages 617–630, 2008. (page 44)

[206] Tomáš Werner. A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7), 2007. (page 87)

[207] Tomas Werner. A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1165–1179, 2007. (page 40)

[208] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019. (page 73)

[209] Oliver Woodford, Philip Torr, Ian Reid, and Andrew Fitzgibbon. Global stereo reconstruction under second-order smoothness priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009. (page 82)

[210] Gengshan Yang, Joshua Manela, Michael Happold, and Deva Ramanan. Hierarchical deep stereo matching on high-resolution images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5515–5524, 2019. (page 123)

[211] Qingqing Yang, Pan Ji, Dongxiao Li, Shaojun Yao, and Ming Zhang. Fast stereo matching using adaptive guided filtering. *Image and Vision Computing*, 32(3):202 – 211, 2014. (page 114)

[212] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on information theory*, 51(7):2282–2312, 2005. (page 40)

[213] Zhichao Yin, Trevor Darrell, and Fisher Yu. Hierarchical discrete distribution decomposition for match density estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. (page 126, 127)

[214] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. (page 72)

[215] Alan L Yuille. Cccp algorithms to minimize the bethe and kikuchi free energies: Convergent alternatives to belief propagation. *Neural computation*, 14(7):1691–1722, 2002. (page 40)

[216] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In *European Conference on Computer Vision (ECCV)*, volume 801, 1994. (page 5, 83, 164)

[217] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l1 optical flow. In *German Conference on Pattern Recognition (GCPR)*, 2007. (page 137, 138)

[218] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. (page 83, 84)

[219] Jure Žbontar and Yann LeCun. Computing the stereo matching cost with a convolutional neural network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1592–1599, June 2015. (page 88)

[220] Jure Žbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 2016. (page 6, 7, 65, 81, 82, 83, 84, 88, 94, 95, 96, 97, 98, 101, 123, 152, 163)

[221] Feihu Zhang, Victor Prisacariu, Ruigang Yang, and Philip HS Torr. Ga-net: Guided aggregation net for end-to-end stereo matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 185–194, 2019. (page 108, 119, 120, 122, 126, 127)

[222] Ke Zhang, Jiangbo Lu, and Gauthier Lafruit. Cross-based local stereo matching using orthogonal integral images. *IEEE transactions on circuits and systems for video technology*, 19(7):1073–1079, 2009. (page 5)

[223] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision (ECCV)*, pages 649–666, 2016. (page 63)

[224] Richard Zhang, Phillip Isola, and Alexei A Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1058–1067, 2017. (page 63)

[225] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalog Du, Chang Huang, and Philip H. S. Torr. Conditional random fields as recurrent neural networs. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. (page 22, 84, 85)

[226] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1529–1537, 2015. (page 44, 105, 108)

[227] Chao Zhou, Hong Zhang, Xiaoyong Shen, and Jiaya Jia. Unsupervised learning of stereo matching. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. (page 64, 164)

[228] Ding-Xuan Zhou. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2):787–794, 2020. (page 1)

[229] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998. (page 155)

[230] C Lawrence Zitnick and Takeo Kanade. A cooperative algorithm for stereo matching and occlusion detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7): 675–684, 2000. (page 6)