Methods for Learning Structured Prediction in Semantic Segmentation of Natural Images

Andreas Christian Müller

Methods for Learning Structured Prediction in Semantic Segmentation of Natural Images

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelm Universität Bonn

vorgelegt von

Andreas Christian Müller

aus

Offenbach am Main

Bonn, September 2013

Angefertigt mit Genehmigung

der Mathematisch-Naturwissenschaftlichen

Fakultät

der Rheinischen Friedrich-Wilhelms-Universität

Bonn

1. Gutachter Prof. Dr. Sven Behnke

2. Gutachter Prof. Dr. Jürgen Gall

Tag der Promotion: 19.08.2014

Erscheinungsjahr: 2014

Zusammenfassung

Automatische Segmentierung und Erkennung von semantischen Klassen in natürlichen Bildern ist ein wichtiges offenes Problem des maschinellen Sehens. In dieser Arbeit untersuchen wir drei möglichen Ansätze der Erkennung: ohne Überwachung, mit Überwachung auf Ebene von Bildern und mit Überwachung auf Ebene von Pixeln.

Diese Arbeit setzt sich aus drei Teilen zusammen. Im ersten Teil der Arbeit schlagen wir einen Clustering-Algorithmus vor, der eine neuartige, informationstheoretische Zielfunktion optimiert. Wir zeigen, dass der vorgestellte Algorithmus üblichen Standardverfahren aus der Literatur gegenüber klare Vorteile auf vielen verschiedenen Datensätzen hat. Clustering ist ein wichtiger Baustein in vielen Applikationen des machinellen Sehens, insbesondere in der automatischen Segmentierung.

Der zweite Teil dieser Arbeit stellt ein Verfahren zur automatischen Segmentierung und Erkennung von Objektklassen in natürlichen Bildern vor, das mit Hilfe von Supervision in Form von Klassen-Vorkommen auf Bildern in der Lage ist ein Segmentierungsmodell zu lernen.

Der dritte Teil der Arbeit untersucht einen der am weitesten verbreiteten Ansätze zur semantischen Segmentierung und Objektklassensegmentierung, Conditional Random Fields, verbunden mit Verfahren der strukturierten Vorhersage. Wir untersuchen verschiedene Lernalgorithmen des strukturierten Lernens, insbesondere im Zusammenhang mit approximativer Vorhersage. Wir zeigen, dass es möglich ist trotz des Vorhandenseins von Kreisen in den betrachteten Nachbarschaftsgraphen exakte strukturierte Modelle zur Bildsegmentierung zu lernen. Mit den vorgestellten Methoden bringen wir den Stand der Kunst auf zwei komplexen Datensätzen zur semantischen Segmentierung voran, dem MSRC-21 Datensatz von RGB-Bildern und dem NYU V2 Datensatz von RGB-D Bildern von Innenraum-Szenen. Wir stellen außerdem eine Software-Bibliothek vor, die es erlaubt einen weitreichenden Vergleich der besten Lernverfahren für strukturiertes Lernen durchzuführen. Unsere Studie erlaubt uns eine Charakterisierung der betrachteten Algorithmen in einer Reihe von Anwendungen, insbesondere der semantischen Segmentierung und Objektklassensegmentierung.

Abstract

Automatic segmentation and recognition of semantic classes in natural images is an important open problem in computer vision. In this work, we investigate three different approaches to recognition: without supervision, with supervision on level of images, and with supervision on the level of pixels. The thesis comprises three parts.

The first part introduces a clustering algorithm that optimizes a novel information-theoretic objective function. We show that the proposed algorithm has clear advantages over standard algorithms from the literature on a wide array of datasets. Clustering algorithms are an important building block for higher-level computer vision applications, in particular for semantic segmentation.

The second part of this work proposes an algorithm for automatic segmentation and recognition of object classes in natural images, that learns a segmentation model solely from annotation in the form of presence and absence of object classes in images.

The third and main part of this work investigates one of the most popular approaches to the task of object class segmentation and semantic segmentation, based on conditional random fields and structured prediction. We investigate several learning algorithms, in particular in combination with approximate inference procedures. We show how structured models for image segmentation can be learned exactly in practical settings, even in the presence of many loops in the underlying neighborhood graphs. The introduced methods provide results advancing the state-of-the-art on two complex benchmark datasets for semantic segmentation, the MSRC-21 Dataset of RGB images and the NYU V2 Dataset or RGB-D images of indoor scenes. Finally, we introduce a software library that allows us to perform extensive empirical comparisons of state-of-the-art structured learning approaches. This allows us to characterize their practical properties in a range of applications, in particular for semantic segmentation and object class segmentation.

Acknowledgements

First, I would like to thank my advisor Sven Behnke, who allowed me to pursue my PhD in his department and who provided funding for my studies. I would also like to thank Jürgen Gall for agreeing to be my second reader.

During my work on this dissertation, my ideas were shaped by many of my fellow students and researchers. First and foremost I am in debt to Hannes Schulz, whose feedback proved to be invaluable during my studies. I would like to thank Christoph Lampert for hosting me as a visiting researcher at the IST Austria. His guidance and advice helped me in directing my further research. My thanks are also extended to Sebastian Nowozin for his collaboration, many helpful discussions, and advice. I would also like to thank Carsten Rother for allowing me to work with him at Microsoft Research Cambridge and hosting me there.

The B-IT research school kindly provided funding for parts of my research, which I am also thankful for.

I have been lucky to be a part of the SCIKIT-LEARN community, whose developers have been a constant source of encouragement and inspiration to me. My understanding of machine learning as well as software engineering has been greatly affected by the continuing exchange with Olivier Grisel, Lars Buitinck, Gaël Varoquax, Mathieu Blondel, Gilles Louppe, Arnaud Joly and others. In particular I want to thank Vlad Niculae for his help during the formation of this work.

I also thank my parents and my sister for their support.

Last but not least I am grateful to Anna Müller for her support, encouragement and company. Thank you for giving me the strength needed for this endeavour.

Contents

1 Introduction

> Essentially, all models are wrong,
> but some are useful.
>
> *George E. P. Box*

In computer vision research, the goal is to automatically extract information from a given image or image sequence. In particular discerning semantic information, that is interpreting an image, is a prominent research topic. While much progress has been made in recent years, computer vision systems still lag behind human vision in most tasks that require semantic information. These tasks can often be formulated in terms of *semantic classes*, meaning categories of parts, objects or scenes. Examples include answering questions such as "Is this a picture of a beach?", "How many cars are there in this image?" or even "What objects lie on the table?". These questions illustrate a range of possible tasks involving semantic categories, such as classifying images of single objects, localization and counting of object classes, and parsing a scene fully into objects and object classes together with their relations. While humans can distinguish tens of thousands of object classes, and have little trouble in interpreting complex scenes, current methods are often restricted to a much smaller number of classes and only have limited capabilities to model interactions or relations. We believe that *context* is one of the most important cues when it comes to classifying objects, and therefore understanding scenes. Therefore, we target dense labeling of scenes, taking object relations into account. The task of densely labeling image pixels into classes is called object class segmentation or semantic segmentation.

We choose this task in particular for the following reasons:

- Pixel-level annotation provides highly detailed information about the scene.
- Joint estimation of multiple classes allows for the use of context.
- In contrast to category-agnostic segmentation approaches, object class segmentation has an unambiguous true labeling.
- A variety of manually annotated datasets is publicly available.

Applications of semantic segmentation and scene understanding include automatic image interpretation for retrieval, autonomous driving and mobile robotics. Besides these applications, due to the abundance of camera data and the proliferation of mobile computing, we expect semantic annotation of images to be a key component in future technologies.

In the following we distinguish between the task of semantic segmentation, which usually distinguishes unstructured "stuff" classes such as road and grass, and object class segmentation, which denotes the segmentation of very structured classes, such as cars, planes and people. We consider four different datasets in this thesis: the object class segmentation datasets GRAZ-02 [Marszatek and Schmid, 2007] and PASCAL VOC 2010 [Everingham et al., 2010], and the semantic segmentation datasets MSRC-21 [Shotton et al., 2006] and NYU V2 [Silberman et al., 2012]. Examples can be found in Figure 1.1. Both tasks have the same ultimate goal of parsing, and therefore understanding, images in terms of semantic classes. However they employ different mechanisms to represent and process.

One of the bottlenecks in learning object class segmentation and semantic segmentation is the availability of training data. While unlabeled image data, and even data with semantic "tags" is available in practically unlimited quantities, semantic annotation on pixel level is scarce and only available through laborious manual annotation. Chapter 3 introduces an approach to cope with this shortcoming of object class segmentation approaches by introducing a method to learn segmentation automatically from image level annotations.

The main part of this thesis investigates the use of structured learning [Taskar et al., 2003, Tsochantaridis et al., 2006] algorithms to the task of semantic image segmentation. Both topics have received much attention in the computer vision and machine learning communities lately [Ladicky et al., 2009, Krähenbühl and Koltun, 2012, Branson et al., 2013, Blake et al., 2011]. Unfortunately, learning

Figure 1.1: Examples from the MSRC-21 (top) and PASCAL VOC (bottom) datasets with ground-truth annotation. MSRC-21 contains mostly texture classes, such as tree, building, street and sky, but also objects, like cars in this example. PASCAL VOC contains only object classes, such as person, cat, table and bottle, and an additional background class (black).

structured prediction in computer vision applications is still little understood. We focus on the use of conditional random fields (CRFs), which have shown promising results for computer vision applications. Using the paradigm of structural support vector machines (SSVMs), it is possible to learn conditional random fields to directly minimize a loss of interest. In particular, CRFs allow to combine different cues, possibly produced using different paradigms in a principled manner. One of the main difficulties with CRF approaches to computer vision problems is that context in images is usually represented as a neighborhood graph of pixels or superpixels. These graphs, by nature, contain many cycles, making inference intractable in general. Consequently, learning algorithms have to rely on approximate inference, with often unknown consequences to learning.

There have been several previous studies on learning structural support vector machines, and learning for conditional random fields. The impact of approximate inference was first investigated by Finley and Joachims [2008], applying structural support vector machines to multi-label data. Later, different works investigated how to combine approximate inference and learning in a single frame-

work. Meshi et al. [2010], Komodakis [2011], and Hazan and Urtasun [2010] approached the problem using duality, and formulate learning and inference as a joint optimization problem. Stoyanov et al. [2011], and later Jancsary et al. [2013] and Krähenbühl and Koltun [2013] formulated learning structured prediction as optimizing a *prediction engine*, that takes into account all aspects of the model, in particular the inference algorithm used. In this work, on the other hand, we follow the well-established algorithms for learning structural support vector machines, and investigate how we can use the available inference algorithms to obtain good results within a reasonable time-frame.

Nowozin et al. [2010] provided a detailed evaluation of different aspects of learning object class segmentation, that is somewhat orthogonal to this work. Their work considers the choice of features, number of superpixels and pairwise potentials for conditional maximum likelihood learning of tree-structured CRFs. Nowozin et al. [2010] also compared conditional maximum likelihood learning with maximum margin learning, finding little difference in accuracy. We focus our work on the more popular neighborhood graphs, which do not allow for efficient inference. Therefore, conditional maximum likelihood learning is intractable in our setting. We focus on the use of maximum-margin methods and their practicality for semantic segmentation and object class recognition. For comparison, we also evaluate these algorithms in settings where exact inference is possible. More recently, Lucchi et al. [2013] proposed a novel algorithm for efficiently learning structured prediction for semantic segmentation, using approximate inference. In Chapter 6, we develop an algorithm that runs in similar time to the one proposed by Lucchi et al. [2013], which is able to learn a CRF to the exact optimum on the same dataset.

Some recent works use alternatives to the CRF approach to object class segmentation, most notably Li et al. [2010] and Xia et al. [2012]. Li et al. [2010] use a pool of candidate segments, which are ranked according to how object-like they are. A generic ranking is followed by a per-class ranking, which outputs whole-object hypotheses. The work was later extended using a more holistic probabilistic approach by Li et al. [2013]. Xia et al. [2012] used sparse coding of object masks on multiple scales together with a bag-of-word model. Their objective jointly optimizes per-class shape masks and image-based per-instance masks. While both approaches are highly promising, they are out of the scope of this work.

1.1 List of Contributions

This thesis contains the following contributions:

- Introducing a clustering algorithm that improves upon widely used approaches from the literature. Our algorithm yields better clusterings in terms of known classes on a wide range of standard datasets.
- Demonstrating a scalable algorithm for weakly supervised object class segmentation. The proposed method is able to learn to segment complex object classes using image annotation alone.
- Providing a general and efficient open source software framework for structured prediction.
- Analysing max-margin learning algorithms with exact and approximate inference in different applications. We give a thorough evaluation of all major SSVM learning algorithms in a wide array of application.
- Showing that exact learning for semantic segmentation and object class segmentation is possible in practice, even in loopy graphs. We combine fast inference, caching and inference algorithms which certify optimality to learn a 1-slack SSVM.
- Learning 3D relations of semantic structure categories for indoor scenes. We extend the CRF approach to learning spacial relations from RGB-D data and improve upon the state-of-the-art in semantic annotation on the NYU V2 dataset of indoor scenes.

1.2 Thesis Outline

Before we delve into semantic segmentation and object class recognition, we first investigate a general clustering algorithm in Chapter 2. Clustering is an important step in most semantic segmentation pipelines, in at least two places: bottom-up segmentation and creation of dictionaries for feature learning. We introduce a novel information theoretic algorithm that compares favorably with algorithms from the literature. While we do not apply our algorithm to the task of bottom-up segmentation, this is a promising avenue for future research.

We introduce an algorithm for semi-supervised learning of object class segmentation in Chapter 3, motivated by the difficulty of obtaining annotated training data for semantic segmentation.

The central topic of this thesis, learning structured prediction for semantic segmentation, is introduced in Chapter 4. This chapter also introduces our software library for implementing structured learning and prediction algorithms.

Chapter 5 gives a quantitative comparison of the most widely used structured prediction algorithms in diverse settings. In particular, we investigate learning behavior when exact inference is intractable, including experiments for semantic segmentation on the popular Pascal VOC dataset and the MSRC-21 dataset.

The problem of learning with approximate inference is investigated in Chapter 6. We develop a strategy for efficient caching and a combination of inference algorithms that allows us to learn SSVMs for semantic image segmentation exactly, even though the involved factor graphs contain many loops. We demonstrate our algorithm on the Pascal VOC 2010, where we are competitive with comparable approaches, and MSRC-21 datasets where we improve upon the state-of-the-art.

Finally, Chapter 7 applies the methods developed in Chapter 4 and Chapter 6 to the problem of semantic annotation with structure classes in RGB-D data. We demonstrate that we are able to learn meaningful spatial relations, and outperform state-of-the-art methods on the NYU V2 datasets.

1.3 Publications

The main material of this thesis has either been published in conference proceedings or has been submitted to conferences or journals. We now list the relevant publications.

Chapter 2 *Information Theoretic Clustering using Minimum Spanning Trees*
Andreas C. Müller, Sebastian Nowozin and Christoph H. Lampert. Published in the proceedings of the German Conference on Pattern Recognition.

Chapter 3 *Multi-Instance Methods for Partially Supervised Image Segmentation*
Andreas C. Müller and Sven Behnke. Published in the proceedings of the IARP Workshop on Partially Supervised Learning.

Chapter 4 *PyStruct - Structured Prediction in Python*

Andreas C. Müller and Sven Behnke. Submitted to the Journal of Machine Learning Research, Open Source Software track.

Chapter 6 *Learning a Loopy Model for Semantic Segmentation Exactly*

Andreas C. Müller and Sven Behnke. arXiv preprint 1309.4061, Submitted to the International Conference on Computer Vision Theory and Applications.

Chapter 7 *Learning Depth-Sensitive Conditional Random Fields for Semantic Segmentation*

Andreas C. Müller and Sven Behnke. Submitted to the International Conference on Robotics and Automation.

2 Information Theoretic Clustering

Before we start our investigation of semantic segmentation and object class segmentation, we look into a general purpose clustering algorithm. In clustering, the goal is to divide data points into homogeneous subsets, called clusters. Many different formulations of the clustering problem are given in the literature. In the context of this work, clustering plays an important role in many respects:

- Clustering, as a purely unsupervised method, is on one end of the spectrum of algorithms we investigate. When using per-image or per-pixel supervision as in the later chapters, we can use clustering to calibrate our expectation of what semi-supervised and supervised algorithms should be able to achieve.

- Clustering algorithms build the basis of most superpixel algorithms, and better clustering algorithms can lead to better superpixel algorithms.

- As many other computer vision algorithms, our segmentation methods depends on bag-of-feature representations of segments or images. These are built using a vocabulary of visual words that is usually created via clustering.

So not only is clustering one of the fundamental problems in machine learning, it is also an important building block for other methods in this work. Most algorithms are based on ad-hoc criteria such as intra-cluster similarity and inter-cluster dissimilarity. An alternative approach is to formalize clustering using an information theoretic framework, where one considers inputs as well as cluster assignments as random variables. The goal is then to find an assignment of data points to clusters that maximizes the mutual information between the assignments and the observations.

In the following, we rely on a non-parametric estimator of the data entropy to find clusterings of maximum mutual information. The use of non-parametric estimates allows a data-driven approach, without making strong assumptions on the form of the data distribution. As a consequence, we obtain a very flexible model that, for example, allows non-convex clusters. The resulting objective is easy to evaluate, but difficult to optimize over. We overcome this by proposing an efficient approximate optimization based on Euclidean minimum spanning trees. Because the estimator and the optimization are both parameter-free, the only free parameter of the algorithm is the number of clusters, which makes it very easy to use in practice. The non-parametric entropy estimate we are using is applicable only if the data distribution is absolute continuous and therefore can not be applied if the data lies on a proper submanifold. We show how to overcome this obstacle in practice by using an estimate of the intrinsic dimensionality of the data. The contributions of this chapter are:

- Proposing the use of a minimum spanning tree based entropy estimator in information theoretic clustering.
- Giving a fast algorithm for a relaxed version of the resulting problem.
- Showing the practicality on a number of synthetic and real datasets.
- Extending the clustering to data on submanifolds by estimating intrinsic dimensionality.

2.1 Related Work

The most commonly used clustering algorithm is the k-means algorithm, originally investigated by MacQueen [1967] and most commonly implemented using Lloyd's algorithm [MacQueen, 1967, Lloyd, 1982]. While k-means often works well in practice, one of its main drawbacks is the restriction in cluster shape. Clusters are given by the Voronoi tessellation of the cluster means and therefore are always convex. Another drawback is the non-determinism of the procedure, caused by the dependence on random initialization.

Another widely used method is spectral clustering [Shi and Malik, 2000, Ng et al., 2002], which solves a graph partitioning problem on a similarity graph

constructed from the data. While spectral clustering is much more flexible than k-means it is quite sensitive to the particular choice of graph construction and similarity measure. It is also computationally expensive to compute, because clustering n points requires computing the eigenvalues and -vectors of an $n \times n$ matrix.

Information theoretic approaches to clustering were first investigated in the context of document classification. In this setting, training examples are described by a discrete distribution over words, leading to the task of *distributional clustering*, which was later related to the Information Bottleneck method by Slonim and Tishby [1999]. This setting was described in detail by Dhillon et al. [2003]. In distributional clustering, it is assumed that the distribution of the data is known explicitly (for example as word counts), which is not the case in our setting.

Later, Banerjee et al. [2005] introduced the concept of Bregman Information in the clustering context, generalizing mutual information of distributions, and showed how this leads to a natural formulation of several clustering algorithms. Agakov and Barber [2006] constructed a soft clustering by using a parametric model of $p(Y \mid X)$. The framework of mutual information based clustering was extended to non-parametric entropy estimates by Faivishevsky and Goldberger [2010]. They use a nearest neighbor based estimator of the mutual information, called MeanNN, that takes into account all possible neighborhoods, therefore combining global and local influences. The approximate mutual information is maximized using local search over labels.

Clustering algorithms based on minimum spanning trees (MSTs) have been studied early on in the statistics community, due to their efficiency. One of the earliest methods is single-link agglomerative clustering [Gower and Ross, 1969]. Single-link agglomerative clustering can be understood as a minimum spanning tree-based approach in which the largest edge is removed until the desired number of components is reached. Zahn [1971] refined this criterion by cutting edges that are longer than other edges in the vicinity. This approach requires tuning several constants by hand. More recently, Grygorash et al. [2006] proposed a hierarchical MST-based clustering approach that iteratively cuts edges, merges points in the resulting components, and rebuilds the spanning tree. We limit our discussion to the most widely used algorithm from [Gower and Ross, 1969].

2.2 Clustering using Non-Parametric Entropy Estimates

In general, the goal of clustering can be formulated as follows: given a finite collection of samples $\mathbf{x} = (x_1, \ldots, x_n)$, we want to assign cluster-memberships $\mathbf{y} = (y_1, \ldots, y_n), y_i \in \{1, \ldots k\}$ to these samples. We adopt the viewpoint of information theoretic clustering of Gokcay and Principe [2002], where the x_i are considered i.i.d. samples from a distribution $p(X)$, and the y_i are found such that the mutual information $I(X, Y)$ between the distribution $p(X)$ and the assigned labels $p(Y)$ is maximized. We can rewrite this objective as

$$I(X,Y) = D_{\mathrm{KL}}\big(p(X,Y) \parallel p(X)p(Y)\big) = H(X) - \sum_{y=1}^{k} p(Y{=}y)H(X \mid Y{=}y) \tag{2.1}$$

where

- $D_{\mathrm{KL}}\big(p(X) \parallel q(X)\big) = \int_{\mathcal{X}} p(X) \ln\left(\frac{p(X)}{q(X)}\right) dX$ is the Kullback-Leibler divergence,
- $H(X) = \int_{\mathcal{X}} p(X) \ln\big(p(X)\big)\, dX$ is the differential entropy, and
- $H(X \mid Y{=}y) = \int_{\mathcal{X}} p(X \mid Y{=}y) \ln\big(p(X \mid Y{=}y)\big)\, dX$ is the conditional differential entropy.

Expressing the mutual information in terms of the entropy is convenient, since the objective then decomposes over the values of Y. Additionally, $H(X)$ is independent of the distribution of Y and therefore does not influence the search over $\mathbf{y}$.

Because we are given only a finite sample from $p(X)$, there is no way to exactly compute $I(X, Y)$, and this is still true if we fix a set of cluster indicators y_i. Possible ways to overcome this are:

1. Fit a parametric model $\hat{p}(X, Y \mid \theta)$ to the observations.
2. Use a non-parametric model $\hat{x}$ to approximate $p(X, Y)$.
3. Estimate $H(X \mid Y)$ directly using a non-parametric estimate.

We choose the third option, as it is the most flexible while avoiding the curse of dimensionality that comes with using non-parametric density estimates.

Let $\mathbf{x}_y$ be the set of x_i with label y. Given a non-parametric density estimator H_{est} we have $H_{est}(\mathbf{x}_y) \approx H(X \mid Y{=}y)$, leading to the clustering problem

$$\max_{\mathbf{y}} \quad -\sum_{y=1}^{k} p(Y{=}y) H_{\text{est}}(\mathbf{x}_y), \tag{2.2}$$

where the probability $p(Y{=}y)$ is given by the empirical frequency of y:

$$p(Y=y) = \frac{n_y}{n}, \quad \text{with} \quad n_y = \big|\{i \mid y_i = y\}\big|.$$

2.2.1 Minimum Spanning Tree Based Entropy Estimation

From now on, we assume that $\mathcal{X} = \mathbb{R}^d$ and $p(X)$ is absolute continuous. This setting allows the use of the non-parametric entropy estimate of Hero III and Michel [1999], that constructs a minimum spanning tree of the data and obtains an estimate of the data entropy from the logarithm of the length of the spanning tree. More precisely, the entropy estimate of a dataset $\mathbf{x} = (x_1, \ldots, x_n)$ is given by

$$H_{mst}(\mathbf{x}) = d\log(L) - (d-1)\log(n) + \log(\beta_d) \tag{2.3}$$

where L is the length of a minimum spanning tree $T(\mathbf{x})$ of $\mathbf{x}$ and β_d is an unknown, but data-independent constant. The estimator H_{mst} is consistent in the sense that $H_{mst}(\mathbf{x}) \to H(X)$ for $n \to \infty$ [Hero III and Michel, 1999]. Using Equation 2.3 as a non-parametric entropy estimate in Equation 2.2 yields the problem to maximize $\hat{I}(\mathbf{x}, \mathbf{y})$ with

$$\begin{aligned}
\hat{I}(\mathbf{x}, \mathbf{y}) &:= -\sum_{y=0}^{k} p(y)\Big[d\log(L_y) - (d-1)\log n_y\Big] + C, && (2.4)\\
&= -\sum_{y=0}^{k} p(y)\Big[d\log(\bar{L}_y) + \log n_y\Big] + C' && (2.5)\\
&= -d\sum_{y=0}^{k} p(y)\log(\bar{L}_y) - \sum_{y=0}^{k} p(y)\log p(y) + C''. && (2.6)
\end{aligned}$$

Here n_y is the cardinality of $\mathbf{x}_y$, L_y is the length of the minimum spanning tree $T(\mathbf{x}_y)$ and C, C' and C'' are constants independent of $\mathbf{y}$. We defined $\bar{L}_y := \frac{L_y}{n_y}$, the mean edge length per node in $T(\mathbf{x}_y)$.

Equation 2.6 has a natural interpretation: The first term penalizes long spanning trees, weighted by the size of the cluster. The second term favors a high entropy of $p(y)$, leading to balanced clusters. Note that there is a natural trade-off between enforcing intra-cluster similarity, expressed through L and the balancing of cluster sizes. This trade-off is similar to formulating an objective in terms of a loss and a regularizer. In contrast to the "loss + regularizer" setup, where the trade-off needs to be specified by the user, the trade-off in Equation 2.6, given by the factor d, is a direct consequence of the entropy estimator.

The reliance on the dimensionality of the ambient space $\mathbb{R}^d$ can be seen as the requirement that d is actually the intrinsic dimensionality of the data. This requirement is made explicit in our assumptions of an absolute continuous data density: If the support of $p(X)$ was a lower-dimensional sub-manifold of $\mathbb{R}^d$, $p(X)$ could not be absolute continuous.

2.2.2 Finding Euclidean Minimum Spanning Tree Clusterings

The objective given by Equation 2.4 is a non-linear combinatorial optimization problem. It has two properties that make it hard to optimize:

1. The objective depends in a non-linear way on L_y. This makes linear programming techniques, that proved successful for other combinatorial tasks, not directly applicable.

2. L_y is defined in terms of minimum spanning trees. This set is hard to characterize, as changing the cluster membership of a single node may change the two minimum spanning trees involved completely.

For the above reasons, we propose a simple procedure to approximately solve Equation 2.4. Consider a graph G with nodes $\mathbf{x}$, an arbitrary set of edges, and edge weights given by the Euclidean distances between points. The connected components of G induce a clustering $\mathbf{y}(G)$ of $\mathbf{x}$, by assigning x_i and x_j the same

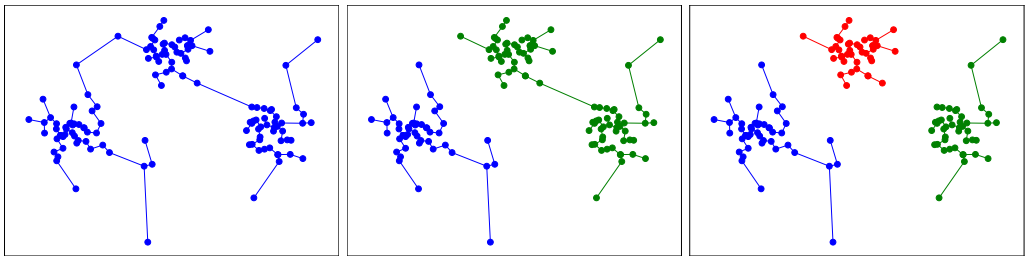

Figure 2.1: Illustration of the optimization algorithm for $k = 3$ on synthetic dataset. *Left*: Euclidean minimum spanning tree of the data. *Center*: The edge that yields the best two-cluster partition in terms of Equation 2.4 was removed, yielding two connected components. *Right*: Another edge from the forest was removed, resulting in the desired number of three components. Note that the edges that are removed are not the longest edges but form a trade-off between edge length and cluster size.

cluster if and only if they are in the same connected component of G. Define

$$\hat{I}(G) := -\sum_{y=0}^{k} p(y)\Big[d\log(L_{G,y}) - (d-1)\log n_y\Big], \tag{2.7}$$

where y enumerates the connected components $G_0, \ldots, G_k$ of G, $n_y = |V(G_y)|$ is the number of nodes in G_y and $L_{G,y} = \sum_{e \in E(G_y)} w(e)$ is the sum of the weights of all edges in the connected component G_y. Then $\hat{I}(G) \geq \hat{I}\big(\mathbf{y}(G)\big)$, by the definition of the minimum spanning tree, and equality holds if and only if G_y is the minimum spanning tree of its nodes for all y. We try to find a graph G with k components, such that $\hat{I}(G)$ is maximal. We can restrict ourself to optimizing over the set $\mathcal{F}$ of forests over $\mathbf{x}$ with k components, as adding edges inside connected components only decrease the objective. Thus we can formulate the clustering problem equivalently as

$$\max_{G \in \mathcal{F}} \hat{I}(G). \tag{2.8}$$

Optimization over forests remains hard, and we further restrict ourself to solutions of the form $\mathcal{G} := \{F \in \mathcal{F} \mid F \text{ subgraph of } T(\mathbf{x})\}$ for a given minimum spanning tree $T(\mathbf{x})$, leading to the problem $\max_{G \in \mathcal{G}} \hat{I}(G)$. This restriction allows for a very fast, combinatorial optimization procedure.

For the two class case, optimization of the above objective can be solved exactly and efficiently by searching over all of $\mathcal{G}$. This amounts to searching for the edge e

Algorithm 1 Information Theoretic MST-based Clustering

Input: Points $\mathbf{x}$, desired number of clusters k.
Output: Clustering $\mathbf{y}$ of $\mathbf{x}$

$G \leftarrow T(\mathbf{x})$
for $i = 0, \ldots, k-1$ **do**
  for $G_j, j = 0, \ldots, i$ connected components of G **do**
    $e_j \leftarrow \text{SplitCluster}(G_j)$
  $l \leftarrow \underset{j}{\text{argmax}}\ \hat{I}(G_j \setminus e_j)$
  $G \leftarrow G \setminus e_l$

function SplitCluster(G)
  Pick arbitrary root x_0 of G.
  for node x starting from leaves **do**
    $w_x \leftarrow \sum_{c \in \text{children}(x)} w_c + d(x, c)$
    $n_x \leftarrow 1 + \sum_{c \in \text{children}(x)} n_c$
  for node x **do**
    $w'_x \leftarrow w_{x_0} - w_x$
  for $e \in E(G), e = (c, p)$, p parent of c **do**
    $v_c \leftarrow w'_p + w_p - w_c - d(p, c)$
    $m_c \leftarrow n - n_c$
    objective(e) $\leftarrow dm_c \ln(m_c) - (d-1)m_c \ln(v_c) + dn_c \ln(n_c) - (d-1)n_c \ln(w_c)$
  $e^* \leftarrow \underset{e \in E(G)}{\text{argmax}}$ objective(e)

that maximizes $\hat{I}\big(T(\mathbf{x}) \setminus e\big)$. The naive algorithm that computes the objective for each edge separately has run time that is quadratic in the number of data points. To improve upon this, we use a dynamic programming approach as described in function SplitCluster of Algorithm 1 , which has only linear complexity. Using this algorithm, run time in the two cluster case is dominated by computing $T(\mathbf{x})$. We extend this algorithm to the case of more than two clusters in a greedy way: Starting with the full spanning tree of $\mathbf{x}$, we remove the edge yielding the lowest value of Equation 2.7 until the number of components equals the number of desired clusters. The overall procedure is summarized in Algorithm 1, which we call *Information Theoretic MST-based (ITM) clustering*. An illustration can be found in Figure 2.1

We use Prim's algorithm combined with a ball tree data structure [Omohundro, 1989] for distance queries to compute the minimum spanning tree of the data. While this procedure has no strong runtime guarantees, we found this faster in practice than specialized algorithms for euclidean minimum spanning trees, which achieve a better theoretical runtime of $O\big(n \log(n)\alpha(n)\big)$ [March, William B, Ram, Parikshit, and Gray, Alexander G, 2010]. Here α is the inverse of the Ackerman function. The dynamic programming solution of Algorithm 1 has a run time of $O(n)$ per removed edge, leading to an overall run time of $O\big(n \log(n)\alpha(n) + nk\big)$. The $O(nk)$ term comes from a worst case scenario, in which each step in the hierarchical clustering procedure only splits off a constant number of points. In a more realistic setting, we expect that the individual clusters are much smaller than the original dataset. In this case, the $O(nk)$ term would improve to $O\big(n \log(k)\big)$.

2.2.3 Estimating Intrinsic Dimensionality

While assuming totally continuous densities is very natural from a theoretical point of view, it can be a hindrance in practical applications. Often, the data is assumed to lie on a submanifold, embedded in a higher-dimensional space. In this case, the density is not totally continuous, and the dimensionality of the data can not be taken as the dimensionality of the embedding space.

A particularly drastic example is the case of the dataset having less samples than features. In this case, the data clearly lies even on a linear subspace of the input space, and the dimensionality of the input space does not accurately reflect the intrinsic dimensionality of the data. To overcome this problem, we use the estimate of intrinsic dimensionality analyzed by Massoud Farahmand et al. [2007]. In their method, for each data point x, a local estimate $\hat{d}(x)$ of the dimensionality at x is computed as

$$\hat{d}(x) = \frac{\ln 2}{\ln\big(r_k(x)/r_{\lfloor k/2 \rfloor}(x)\big)}. \tag{2.9}$$

Here k is a fixed integer and $r_k(x)$ is the distance of x from its kth neighbor. We follow Massoud Farahmand et al. [2007] and set $k = \lceil 2 \ln n \rceil$. The final estimate $\hat{d}$ is then computed by averaging the estimates over all x

$$\hat{d} = \frac{1}{n} \sum_{x \in X} \min\Big(\hat{d}(x),\ d\Big). \tag{2.10}$$

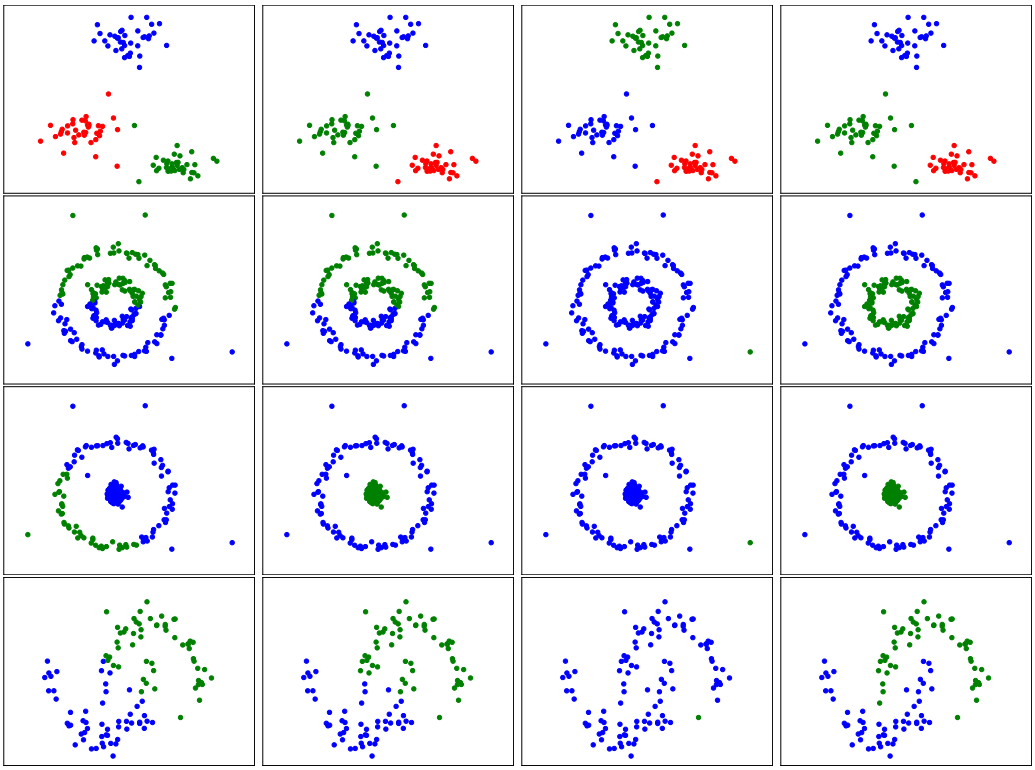

Figure 2.2: Comparison of k-means (left), MeanNN (center left), single link (center right) and ITM (right) on four synthetic datasets. Without the need to tune parameters, ITM can adjust to different cluster shapes. MeanNN is able to recover non-convex clusters (third row) but often produces similar results to k-means (second and last row). Single link clustering is very sensitive to noise, as it does not take cluster size into account.

We compute the density estimate once, prior to clustering, and then plug the estimate $\hat{d}$ into equation 2.7 in place of d. We found this estimate to work robustly and give sensible results for all datasets we investigated. As we already used a ball tree data structure to build the minimum spanning trees, we can reuse this structure to compute r_k. Consequently, estimating the dimensionality resulted only in little computational overhead.

2.3 Experiments

We compared ITM to the popular k-means algorithm [MacQueen, 1967, Lloyd, 1982], to the MeanNN algorithm of Faivishevsky and Goldberger [2010] and to single-link agglomerative clustering [Gower and Ross, 1969]. The similari-

Algorithm	Objective	Det.	Complexity
k-means	$\sum_y \sum_{i,y_i=y} \|x_i - \mu_y\|^2$	No	$O(nk)$ per iteration
MeanNN	$\sum_y \log\left(\frac{1}{\|\mathbf{x}_y\|} \sum_{i,j,y_i=y_j=y} \|x_i - x_j\|^2\right)$	No	$O(n^2)$ per iteration
Single Link	–	Yes	$O(n \log n)$
ITM	$\sum_{y=0}^{k} dp(y) \log(\bar{L}_y) + p(y) \log p(y)$	Yes	$O\big(\alpha(n)n \log n + nk\big)$

Table 2.1: Comparing properties of related algorithms. Det. stands for Deterministic

ties between single-link agglomerative clustering and the proposed MST-based optimization make it a good baseline for tree-based clustering approaches.

A comparison of ITM, MeanNN and the baseline methods, k-means and single link agglomerative clustering, in terms of their objective, optimization and complexity can be found in Table 2.1. We implemented the ITM clustering procedure as well as MeanNN in Python. We used the k-means implementation available in the SCIKIT-LEARN library [Pedregosa et al., 2011]. The source code is available online*.

2.3.1 Experimental Setup

For both k-means and MeanNN, we restart the algorithm ten times using different random initializations, keeping the result with the best objective value. As ITM is deterministic there is no need for random restarts. All of the algorithms we compare work with a fixed number of clusters, which we set to the number of classes in the dataset for all experiments.

As single link agglomerative clustering is sensitive to outliers, we set a hard limit of five on the minimum number of samples per cluster for the quantitative analysis.

2.3.2 Qualitative Results

Figure 2.2 shows qualitative results on three synthetic datasets. For well separated, convex clusters, the four algorithms produce very similar clusters (see top row). If

*`https://github.com/amueller/information-theoretic-mst`

the structure of the data is more complex, the advantage of the proposed method is apparent. Note that there was no need to specify any other parameters than the number of clusters to produce these results. It is also noteworthy that the results of MeanNN are very close to those produces by k-means in most cases. This similarity can be explained by the close relation of the objective functions, listed in Table 2.1.

2.3.3 Quantitative Results

We present results on several standard datasets from the UCI repository, selecting datasets that span a wide range of combinations of number of samples, features and clusters. To satisfy the assumption of absolute continuity of the data distribution, we restrict ourself to data with continuous features.

We evaluated the experiments using the *adjusted Rand index (ARI)* [Hubert and Arabie, 1985] a popular measure of cluster quality [Gomes et al., 2010, Kamvar et al., 2003]. The Rand index [Rand, 1971] between two clusterings counts how many pairs of points two clusterings agree on. The adjusted Rand index contains a calibration against chance performance. We also measured *normalized mutual information (NMI)* [Strehl and Ghosh, 2003], but do not report it here, as it resulted in an identical ranking of the clustering algorithms.

Table 2.2 summarizes the results. The two entropy-based methods (MeanNN, ITM) have a clear advantage of the other methods, with ITM finding better clusterings than MeanNN in the majority of cases. We see that ITM does well when the intrinsic dimensionality of the data matches the feature dimension, but degraded otherwise (see FACES and USPS). Estimating the intrinsic dimensionality of the data overcomes this weakness, and improves results in most cases. For all but one dataset, either ITM or ITM with estimated intrinsic dimensionality gives the best results of all considered algorithms. The single link agglomerative clustering procedure produces reasonable results on datasets with little noise and well-separated clusters, but fails otherwise. The run time of computing the ITM clustering was dominated by the computation of the MST of the data.

Dataset				Results				
Description	n	d	k	k-means	MeanNN	SL	ITM	ITM ID
DIGITS	1797	64	10	0.62	0.67	0.10	**0.85**	0.73
FACES	400	4096	40	0.41	0.49	0.08	0.02	**0.54**
IRIS	150	4	3	0.72	0.75	0.55	**0.88**	**0.88**
USPS	9298	256	10	0.52	0.54	0.00	0.44	**0.64**
VEHICLE	846	18	4	0.10	0.09	0.00	**0.10**	**0.10**
VOWEL	990	10	11	0.17	0.19	0.00	**0.20**	0.19
WAVEFORM	5000	21	2	**0.37**	0.30	0.00	0.23	0.23
MNIST	70000	784	10	0.37	N/A[†]	0.00	0.50	**0.77**

Table 2.2: Adjusted Rand Index of k-means, MeanNN, single link agglomerative clustering and ITM on several datasets (higher is better). ITM ID refers to ITM using the estimated intrinsic dimensionality. The best score for each dataset is printed in bold.
[†]We were unable to make MeanNN scale to 70000 data points, as storing the whole pairwise distance matrix seems necessary.

2.4 Summary

In this chapter we proposed the use of a minimum spanning tree based, non-parametric entropy estimator in information theoretic clustering, ITM. Thereby we extended the work of Faivishevsky and Goldberger [2010] to a more flexible and efficient entropy estimate. We proposed an approximate optimization method by formulating the clustering problem as a search over graphs. The resulting algorithm is deterministic and has sub-quadratic run time. Empirical comparisons showed that the proposed method outperforms standard algorithms and the non-parametric entropy based clustering of Faivishevsky and Goldberger [2010] on multiple benchmark datasets. We demonstrated that ITM is able to detect non-convex clusters, even in the presence of noise. In contrast to other algorithms that can handle non-convex clusters, ITM has no tuning parameters, as the objective presents a natural trade-off between balancing cluster sizes and enforcing intra-cluster similarity. A limitation of the proposed algorithm is that it is based on the assumption of an absolute continuous data distribution. We show that this limitation can be overcome in practice by estimating the intrinsic dimensionality of the data.

3 Weakly Supervised Object Segmentation

Most algorithms for semantic image segmentation and object-class segmentation work with strong supervision: a pixel-wise labeling of training images. In this chapter we investigate a method that works with annotation which is much easier to obtain: whole image labels. While we do not reach the accuracy of competing fully supervised approaches, our efficient, weakly supervised method is potentially able to scale to much larger datasets, without the need for time-consuming manual annotation on pixel level.

Recently, several approaches have been proposed for weakly supervised semantic segmentation. While these are close to our work, there are several important distinctions. We address the task of object-class segmentation which concerns object categories, while semantic segmentation approaches often focus on "stuff" categories like "sky" and "road" which are more easily detected using simple texture features. In contrast to, for example, Vezhnevets et al. [2011], who build a joint model over all segmentation decisions, our approach is in principle applicable to large datasets, the regime where weak annotation is most useful.

In our approach we work with a set of candidate segments, generated using constrained parametric min-cuts [Carreira and Sminchisescu, 2010]. The procedure yields segments that are overlapping, object-like regions which serve as candidates for object locations.

We formulate weakly supervised multi-class image segmentation as a multi-instance problem based on these candidate segments. In multi-instance learning [Dietterich et al., 1997] each training example is given as a multi-set of instances, called a bag. Each instance is represented as a feature vector x and a label y. A bag is labeled positive if it contains at least one positive example and negative otherwise.

During training only the labels of the training bags, not of the instances inside the bags, are known. The goal is to learn a classifier for unseen bags. Formally, let $\mathcal{X}$ be the set of instances. To simplify notation we assume that bags are simply sets, not multi-sets. Then a bag is an element of the power set $2^{\mathcal{X}}$ and the task is to learn a function

$$f_{MI}\colon 2^{\mathcal{X}} \to \{-1, +1\}. \tag{3.1}$$

Training examples are tuples (X_i, y_i) of bags $X_i \subset \mathcal{X}$ and labels $y_i \in \{-1, +1\}$. It is assumed that the f_{MI} function stems from a so-called underlying concept, given by an (unknown) function $f_I\colon \mathcal{X} \to \{-1, +1\}$, with

$$f_{MI}(X) = \max_{x \in X} f_I(x). \tag{3.2}$$

Multi-instance learning is a natural formulation for image classification and has been successfully applied to this task [Zhou and Zhang, 2006]. We propose to go a step further and apply multi-instance learning to the task of object-class segmentation in natural images by also classifying instances, not only bags. In this we follow the work of Li and Sminchisescu [2010] and Zha et al. [2008], who not only learned f_{MI}, but also f_I. In our model each image forms a bag, while the candidate segments correspond to the instances contained in the bag. During learning only presence of object classes is needed as bag-level supervision. By learning f_I, we are then able to predict for individual segments whether they contain the object class of interest, thereby obtaining a segmentation of the object. To measure the performance of our algorithm we use a dataset that not only contains image-level annotation, but also pixel-level annotation of object. This allows us to judge the success of learning on instance level.

3.1 Related Work

3.1.1 Object Segment Proposals

Most work on multi-class segmentation focuses on strong supervision on super-pixel level. There is still little work on using candidate segments. The method we use for generating candidate segments is Constraint Parametric Min-Cuts (CPMC) of Carreira and Sminchisescu [2010]. This method creates a wide variety

of overlapping segments. Support vector regression (SVR) is trained on these segments to estimate the overlap of segments with ground truth object-class labeling from the PASCAL VOC dataset [Everingham et al., 2010]. This provides a ranking of candidate segments according to how "object-like" they are, which allows for selecting only a limited number of very object-like segments. The method performed well on a variety of datasets, building the basis of a very successful entry to the PASCAL VOC segmentation challenge [Li et al., 2010]. A similar approach to whole-object segment proposals was investigated by Endres and Hoiem [2010], but they did not compare their results with the state-of-the-art approach of Carreira and Sminchisescu [2010].

3.1.2 Multi-Instance Methods

Multi-instance learning was formally introduced by Dietterich et al. [1997]. Since then, many algorithms were proposed to solve the multi-instance learning problem [Andrews et al., 2003, Gärtner et al., 2002, Zhou et al., 2009, Li et al., 2009, Zhang and Goldman, 2002, Mangasarian and Wild, 2008, Leistner et al., 2010, Chen et al., 2006]. We discuss only those that are relevant to the present treatment.

Gärtner et al. [2002] introduced the concept of a multi-instance kernel on bags, defined in terms of a kernel on instances. The basic principle of the multi-instance kernel is similar to a soft-max over instances in each bag. This can be viewed as approximating the kernel value of the "closest pair" given by two bags. Gärtner et al. [2002] showed that the multi-instance kernel is able to separate bags if and only if the original kernel on instances is able to separate the underlying concepts. The method of multi-instance kernels has a particular appeal in that it transforms a multi-instance problem into a standard classification problem by using an appropriate kernel. The downside of this approach is that it does not directly label instances, only bags.

Zhou et al. [2009] explicitly addressed the fact that instances are not independent within a bag , leading to an algorithm that can take advantage of possible correlations. Computational costs of their algorithm does not scale well with the number of instances, although a heuristic algorithm is proposed to overcome this restriction. Zhou et al. [2009] demonstrated only a slight advantage of their algorithm over the MI-kernel of Gärtner et al. [2002], so we use the MI-kernel for better scalability.

Li and Sminchisescu [2010] computed likelihood ratios for instances, giving a new convex formulation of the multi-instance problem. Using these likelihood ratios, classification can be performed directly on the instances, provided an appropriate threshold for classifying instances as positive is known. We circumvent this problem by applying the same classifier to instances and bags, thereby obtaining hard class decisions for each instance.

3.1.3 Semantic Scene Segmentation using Weak Annotation

Learning semantic segmentation from image-level annotation was first investigated in Verbeek and Triggs [2007], using a semi-supervised conditional random field on patches. Verbeek and Triggs [2007] evaluated their approach on the MSRC-9 datasets. More recently, similar approaches were proposed by Vezhnevets et al. [2011] and Vezhnevets and Buhmann [2010]. Vezhnevets et al. [2011] independently developed a multiple-instance based approach to segmentation, and report impressive results on the MSRC-21 dataset.

While semantic segmentation is closely related to the task of multi-class image segmentation that we are considering in this chapter, there are important distinctions: In semantic segmentation, each pixel has a semantic annotation, also containing non-object "stuff" classes like "sky", "grass" and "water". In multi-class image segmentation, the focus is on objects, with possibly large parts of the image being labeled as unspecific "background". The unspecific background class contains much more clutter than for example "grass" and is therefore much harder to model. Additionally, object classes themselves are much harder to capture using low-level textural information only. This makes disseminating the distinctive features in multi-class object recognition much more challenging, and requires a more holistic approach to recognition than these patch-based or superpixel-based approaches.

3.2 Multi-Instance Kernels for Image Segmentation

3.2.1 Constraint Parametric Min-Cuts (CPMC)

To generate proposal segments, we use the Constraint Parametric Min-Cuts (CPCM) method of Carreira and Sminchisescu [2010]. In CPMC, initial segments are constructed using graph cuts on the pixel grid. The energy function for these cuts uses pixel color and the response of the global probability of boundary (gPb) detector [Maire et al., 2008]. As much as ten thousand initial segments are generated from foreground and background seeds. A fast rejection based on segment size and ratio cut [Wang and Siskind, 2003] reduces these to about 2000 overlapping segments per image. Then, the segments are ranked according to a measure of object-likeness that is based on region and Gestalt properties. This ranking is computed using an SVR model [Carreira and Sminchisescu, 2010], which is available online. For computing the global probability of boundary (gPb), we used the CUDA implementation of Catanzaro et al. [2009], which provides a speedup of two orders of magnitude over the original implementation.

3.2.2 Multi-Instance Learning using MI-Kernels

Since scalability is very important in real-world computer vision applications, and natural images might need hundreds of segments to account for all possible object boundaries, we use the efficient multi-instance kernel [Gärtner et al., 2002]. Multi-instance kernels are a form of set kernels that transform a kernel on instance level to a kernel on bag level. We reduce the multi-instance multi-class problem to a multi-instance problem by using the one-vs-all approach.

With k_I denoting a kernel on instances $x, x' \in \mathcal{X}$, the corresponding multi-instance kernel k_{MI} on bags $X, X' \in 2^{\mathcal{X}}$ is defined as

$$k_{MI}(X, X') := \sum_{x \in X, x' \in X'} k_I^p(x, x'), \tag{3.3}$$

where $p \in \mathbb{N}$ is a free parameter [Gärtner et al., 2002]. As we use the RBF-kernel k_{rbf} as kernel on $\mathcal{X}$ and powers of RBF-kernels are again RBF-kernels, we do not consider p explicitly in the following.

We normalize the kernel k_{MI} [Gärtner et al., 2002] using

$$k(X, X') := \frac{k_{MI}(X, X')}{\sqrt{k_{MI}(X, X) k_{MI}(X', X')}}. \tag{3.4}$$

Training an SVM with this kernel produces a bag-level classifier for each class, which we refer to as MIK. This procedure is very efficient since the resulting kernel matrix is of size of the number of bags, which is much smaller than a kernel matrix of size of the number of instances, as is commonly used in the literature [Andrews et al., 2003, Nguyen, 2010]. Another advantage over other methods is the use of a single convex optimization, whereas other approaches often use iterative algorithms [Andrews et al., 2003] or need to fit complex probabilistic models [Zha et al., 2008].

While using MIK has many advantages, it produces only an instance-level classifier. We propose to transform a bag-level classifier f_{MI} as given by the SVM and Equation 3.3 into an instance-level classifier by setting $f_I(x) := f_{MI}(\{x\})$, in other words, by considering each instance as its own bag.

3.2.3 Segment Features

To describe single segments, we make use of densely computed SIFT [Lowe, 2004] and ColorSIFT [van de Sande et al., 2010] features on multiple scales, from which we compute bag-of-visual-word histograms. Additionally, we use a pyramid of histograms of oriented gradients [Dalal and Triggs, 2005] on the segments. We use RBF-kernels for all of the features, constructing one MI-kernel per feature. These are then combined using multiple kernel learning to produce a single kernel matrix. This kernel matrix can then be used for all classes, making classification particularly efficient.

3.2.4 Combining Segments

The framework described above yields an image-level and a segment-level classification. To obtain a pixel-level object-class segmentation, we have to combine these. Since we do not make use of the ground truth segmentation during training, we cannot learn an optimal combination as Li et al. [2010] did, but perform a simple majority vote instead.

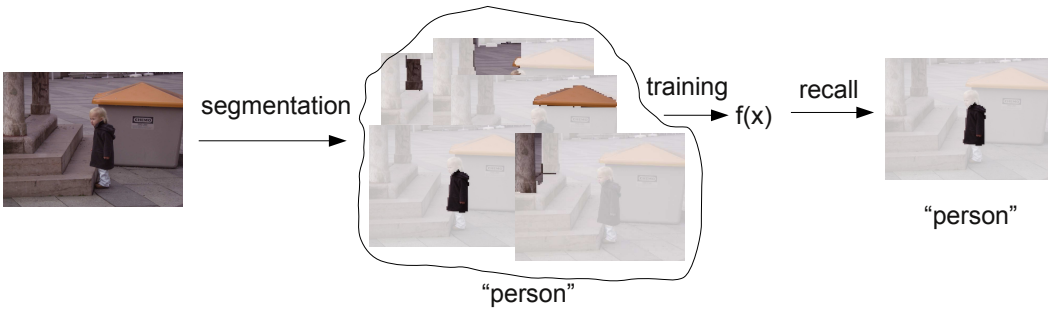

Figure 3.1: Schematic overview. See text for details.

We merge segments into pixel-level class labels by setting the label y_x of a pixel x according to:

$$y_x = \underset{y \in Y}{\operatorname{argmax}} |\{S_i | x \in S_i \wedge y_{S_i} = y\}|, \tag{3.5}$$

Here Y is the set of possible object classes, S_i enumerates all segments within an image and y_{S_i} is the label of segment S_i. In other words each pixel is assigned the class with the highest number of class segments containing it. This simple heuristic yields good results in practice.

3.3 Experiments

3.3.1 Instance-Level Predictions using MI-Kernel

To assess the validity of instance-level predictions using multi-instance kernels, we transform f_I back to an instance-level classifier, using the multi-instance learning assumption (Equation 3.2). We refer to these instance-based MIK predictions as MIK-I. In all experiments, the parameters of the MI-Kernel and SVM are adjusted using MIK and then used with both MIK and MIK-I. This facilitates very fast parameter search since MIK is very efficient to compute. Note that we cannot adjust parameters using instance prediction error, as we assume no instance labels to be known.

We compared the performance of MIK, MIK-I and state-of-the-art MI methods on the Musk benchmark datasets [Dietterich et al., 1997] in Table 3.1. Results were obtained using 10-fold cross-validation. While the computational complexity of MIK-I is very low compared to the other methods, it achieves competitive

	SVM-SVR	EMDD	mi-SVM	MI-SVM	MICA	MIK	MIK-I
Musk1	87.9	84.9	87.4	77.9	84.3	88.0	88.0
Musk2	85.4	84.8	83.6	84.3	90.5	89.3	85.2

Table 3.1: Bag-level accuracy (in percent) of various MIL algorithms on the standard Musk datasets. All but MIK provide instance-level labeling.

	Musk1		Musk2	
	accuracy	witness-rate	accuracy	witness-rate
mi-SVM	87.4	100	83.6	83.9
SVM-SVR	87.9	100	85.4	89.5
MIK-I	88.0	99	85.2	62.3

Table 3.2: Bag-level Accuracy of MIL algorithms on the Musk datasets and the empirical witness rates of the classifiers (both in percent).

results. Using instance-level labels results in a slight loss of accuracy of MIK-I, compared to MIK. Interestingly, even though the model was not trained to provide any instance-level labels, the performance is still competitive.

For multi-class image segmentation, it is beneficial to have a low witness rate, that is only a few instances are assumed to be positive in a positive bag. Since an object might not be very prominent in an image, only a fraction of segments might correspond to the object. Table 3.2 compares the witness rates of MIK-I, miSVM [Andrews et al., 2003] and SVR-SVM [Li and Sminchisescu, 2010] on the Musk datasets. MIK-I is able to achieve similar accuracy with much less witnesses than the other methods. Note that Musk1 consists of very small bags while Musk2 contains significantly larger bags, more similar to the setup concerning images and segments.

3.3.2 Partially Supervised Image Segmentation on Graz-02

We evaluate the performance of the proposed algorithm for object-class segmentation on the challenging Graz-02 dataset [Marszatek and Schmid, 2007]. This dataset contains 1096 images of three object classes, bike, car and person. Each image may contain multiple instances of the same class, but only one class is present per image.

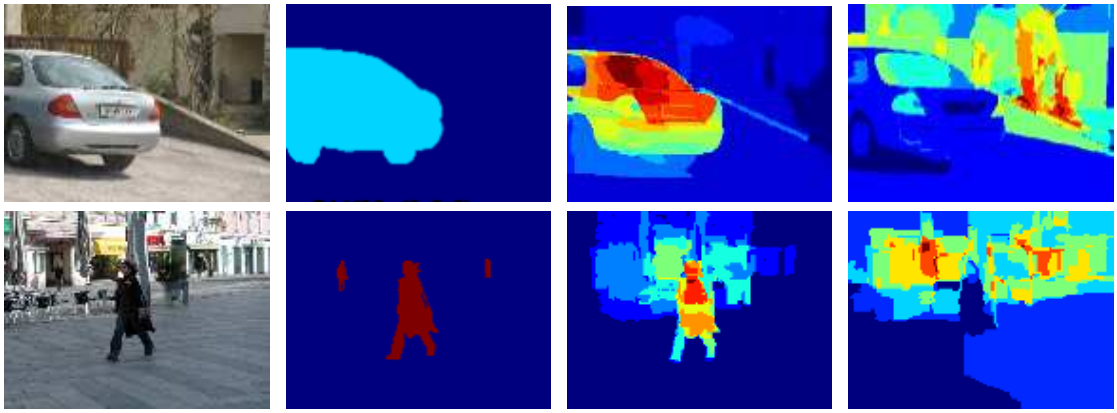

Figure 3.2: Qualitative results on the GRAZ-02 dataset. Top: Results on category "car". Bottom: Results on category "person". From left to right: original image, ground truth segmentation, segment votes for correct class, segment votes against correct class (red many, blue few votes).

	car	bike	person
MIL-MKL (our approach)	30	45	43
Best strongly supervised approaches [Fulkerson et al., 2009, Schulz and Behnke, 2011]	72	72	66

Table 3.3: Pixel-level accuracy (in percent) on the GRAZ-02 dataset.

We adjusted parameters on a hold-out validation set using bag-level information and used the training and test sets as specified in the dataset. We train one multiple kernel learning (MKL) SVM per class using MIK and predict class labels on segment level using MIK-I. This yields a classification of each segment into one of four classes: car, bike, person, or background. We merge segments into pixel-level class labels as described in Section 3.2.4.

Per-class pixel accuracies are reported in Table 3.3; some qualitative results are shown in Figure 3.2. The overall accuracy on images labels, which is the task that was actually trained, is 90%. While the performance of our multiple-instance based approach is far from current methods that use pixel-level annotations, whose pixel-level accuracy is around 70% [Fulkerson et al., 2009, Schulz and Behnke, 2011], it can serve as a baseline for research on weakly supervised methods for object-class segmentation.

3.4 Summary

We proposed an algorithm for object-class segmentation using only weak supervision based on multiple-instance learning. In our approach each image is represented as a bag of object-like proposal segments.

We described a way to extent bag-level predictions made by the multi-instance kernel method to instance level while remaining competitive with the state-of-the-art in bag label prediction.

We evaluated the proposed object-class segmentation method on the challenging GRAZ-02 dataset. While not reaching the performance of methods using strong supervision, our result can work as a baseline for further research into weakly supervised object-class segmentation.

4 Learning Conditional Random Fields

Many classical computer vision applications such as stereo, optical flow, semantic segmentation and visual grouping can be naturally formulated as image labeling tasks. Arguably the most popular way to approach such labeling problems is via graphical models, such as Markov random fields (MRFs) and conditional random fields (CRFs). MRFs and CRFs provide a principled way of integrating local evidence and modeling spatial dependencies, which are strong in most image-based tasks. While in earlier approaches, model parameters were set by hand or using cross-validation, more recently parameters are often learned using a max-margin approach.

Most models employ linear energy functions of unary and pairwise interactions, trained using structural support vector machines (SSVMs). While linear energy functions lead to learning problems that are convex in the parameters, complex constraints complicate their optimization.

In recent years there has been a wealth of research in methods for learning structured prediction, as well as in their application in areas such as natural language processing and computer vision (see Nowozin and Lampert [2011] for an introduction and Blake et al. [2011] for a recent survey). In this chapter, we first introduce the concepts and algorithms used in structured prediction, in particular in maximum margin methods. Then, we review the use of CRFs in computer vision, and introduce our methods. Finally we give a description of our open source implementation of structured learning algorithms, PyStruct.

4.1 Basic Concepts in Structured Prediction

Structured prediction can be defined as making a prediction $f(x)$ by maximizing a compatibility function g between an input x and possible labels y [Nowozin and Lampert, 2011]:

$$f(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, g(x, y) \tag{4.1}$$

Finding y in the above equation is often referred to as inference or prediction. We will use the most common approach of using a linear parametrization of g, which leads to

$$f(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\ \theta^T \Phi(x, y). \tag{4.2}$$

Here, y is a *structured* label, Φ is a joint feature function of x and y, and θ contains the parameters of the model. *Structured* means that y is more complicated than a single output class, for example a label for each word in a sentence or a label for each pixel in an image. Learning structured prediction means learning the parameters θ from training data. The particular model that is used is completely encoded in $\Phi(x, y)$, which manifests the relation between input x and output y. As $\mathcal{Y}$ is typically very large, it is crucial to exploit the particular form of $\Phi(x, y)$ to solve the prediction problem of Equation 4.2.

While y could be complicated like a parse tree or the geometric configuration of a molecule, many settings, such as the image segmentation setting we are interested in, can be reduced to the case where y is a vector of discrete labels $\mathcal{Y} = \{1, \ldots, k\}^n$. In the following, we only discuss this multivariate case. In general, n is often different for different inputs x, such as images with different numbers of pixels. We ignore this in our notation to simplify the presentation.

4.1.1 Factor Graphs and the Relation to Graphical Models

In the case when y is multivariate, a very general and widely used method to specify the structure of a model, and therefore Φ, is using *factor graphs*. A factor graph is a bipartite graph $(\mathcal{V}, \mathcal{F}, \mathcal{E})$, consisting of variable nodes $\mathcal{V}$, factor nodes $\mathcal{F}$ and edges $\mathcal{E}$ connecting variables to factors. The *scope* N_F of a factor $F \in \mathcal{F}$ is defined as

$$N_F = \{v \in \mathcal{V} \,|\, (v, F) \in \mathcal{E}\} \tag{4.3}$$

The variable nodes of the factor graph correspond to the entries of the variables y, that is $\mathcal{V} = \{1, \ldots, n\}$, and each factor node is associated with a *factor* or *potential function* ψ_F. A factor graph represents a function*

$$g(x, y) = \sum_{F \in \mathcal{F}} \psi_F(x, y_{N_F}) \tag{4.4}$$

Here, y_{N_F} denotes the entries of y indexed by N_F.

The benefit of using the factor graph representation is that it decomposes the function over subsets of the variables of interest y_i. This allows us to apply efficient optimization procedures for the prediction problem in Equation 4.2 by exploiting the graph structure of the factor graph. To obtain a linear function from Equation 4.4 as in Equation 4.2, we can simply let each ψ be of the form

$$\psi_F(x, y_{N_F}) = \theta_F^T \Phi_F(x, y_{N_F}). \tag{4.5}$$

The most common form by far is

$$\psi_F(x, y_{N_F}) = \theta_{F, y_{N_F}}^T \phi_F(x), \tag{4.6}$$

where $\phi_F(x)$ is a vector representation of the input x, and there are different parameter vectors $\theta_{F, y_{N_F}}$ for each possible assignment of $y_{N_F} \in \mathcal{Y}_{N_F}$. Both, Equation 4.5 and Equation 4.6 are instantiations of the general linear form Equation 4.2. To see this, for Equation 4.5 we simply concatenate the individual components for all $f \in \mathcal{F}$:

$$\theta = \bigoplus_{F \in \mathcal{F}} \theta_F \tag{4.7}$$

$$\Phi(x, y) = \bigoplus_{F \in \mathcal{F}} \Phi_F(x, y_{N_F}). \tag{4.8}$$

*Traditionally factor graphs represent *products* of factors. To simplify presentation, we work directly in the log-domain of the more standard product representation.

Writing down Φ and θ for the form Equation 4.6 is a little less compact:

$$\theta = \bigoplus_{F \in \mathcal{F}} \bigoplus_{y_{N_F} \in \mathcal{Y}_{N_F}} \theta_{F, y_{N_F}} \tag{4.9}$$

$$\Phi(x, y) = \bigoplus_{F \in \mathcal{F}} \left(\phi_F(x) \otimes e_{y_{N_F}} \right), \tag{4.10}$$

Here $e_{y_{N_F}} \in \mathbb{R}^{|\mathcal{Y}_{N_F}|}$ is the indicator for a given variable setting y_{N_F}. In words, Φ_F is built simply by creating a vector of $|\mathcal{Y}_{N_F}|$ times the size of ϕ_F, which is zero everywhere, except for the position corresponding to y_{N_F}.

This approach to structured prediction is closely related to approaches using *probabilistic graphical models*. Probabilistic graphical models are a tool to express factorizations of probability distributions. Similar to Equation 4.4, the joint probability distribution over a multi-variate random variable y can be expressed using a factor-graph:

$$p(y|x) = \frac{1}{Z_x} \prod_{F \in \mathcal{F}} \exp\left(\psi_F(x, y_{N_F})\right). \tag{4.11}$$

Here

$$Z_x = \sum_{y' \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \exp\left(\psi(x, y'_{N_F})\right) \tag{4.12}$$

is the normalization constant of the conditional distribution over y. If f is chosen as in Equation 4.6, then the resulting distribution belongs to the exponential family, the class of probability distributions most commonly used in the graphical model literature.

The most probable prediction y is given as $\operatorname{argmax}_{y \in \mathcal{Y}} p(y|x)$. As Z is independent of y, and by the monotonicity of the logarithm, maximizing $p(y|x)$ is equivalent to maximizing $g(x, y)$ over y in Equation 4.4. Therefore, from a prediction standpoint, the two formulations are equivalent.

During learning, the presence of the factor Z in Equation 4.11 introduces additional complications. As we only address the problem of making predictions, not modeling probabilities, there are no clear benefits from the probabilistic approach. Consequently, we work with the more direct structured prediction approach of Equation 4.2 and Equation 4.4 instead.

4.2 Learning Max-Margin Structured Prediction

Maximum margin learning has become one of the most popular methods to learn classifiers and structural models in computer vision and text processing. There are several reasons for the popularity of linear maximum margin approaches:

Loss-sensitivity In contrast to most probabilistic approaches, maximum margin learning approaches can directly minimize a user-specified loss.

Feasibility If the loss decomposes over the factor graph that specifies g, then learning is feasible as soon as the maximization over y in Equation 4.2 can be carried out.

Generalization The maximum margin principle yields generalization bounds using the effective complexity [Taskar et al., 2003], that are generally tighter than corresponding VC-theoretical bounds.

Strong Convexity The resulting optimization problem is strongly convex, leading to efficient optimization and unique solutions.

For learning, a dataset $(x^1, y^1), \ldots, (x^k, y^k)$ is given, together with a loss

$$\Delta \colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}. \tag{4.13}$$

The parameters θ are learned by minimizing the loss-based soft-margin objective

$$\min_{\theta} \frac{1}{2}||\theta||^2 + C \sum_i \ell(x^i, y^i, \theta) \tag{4.14}$$

with regularization parameter C. Here, ℓ is a hinge-loss-like upper bound on the empirical Δ-risk:

$$\ell(x^i, y^i, \theta) = [\max_{y \in \mathcal{Y}} \Delta(y^i, y) + \theta^T \Phi(x^i, y) - \theta^T \Phi(x^i, y^i)]_+. \tag{4.15}$$

This is an instance of regularized empirical risk minimization, with a piecewise linear, convex upper bound on the loss. Finding the y that corresponds to a maximum in Equation 4.15 is a central part of all maximum-margin based learning algorithms, and is referred to as loss-augmented prediction. For complex models, such as the ones used for image segmentation, this optimization often dominates

the learning process in terms of computational complexity. Therefore, it is often desirable to find learning algorithms that converge with as little optimizations of the loss-augmented prediction problem as possible.

There are several popular algorithms to solve Equation 4.14. We briefly review three standard algorithms, and a very recent one: the 1-slack and n-slack cutting plane algorithms, a stochastic primal subgradient algorithm, and recently proposed stochastic dual coordinate descent method, which we now describe in detail. We also give simplified known convergence rates in terms of calls to the loss-augmented prediction.

Additionally, we discuss practical implications and implementation. One particularly interesting aspect is the difference between *sequential* (or *online*) and *batch* algorithms. Batch algorithms process the whole dataset before adjusting parameters, while sequential algorithms process one sample at a time, and adjust parameters incrementally. In image segmentation tasks, inference is often costly, making loss-augmented prediction the most expensive step in learning. This often leads to longer run-times for batch algorithms. On the other hand, loss-augmented prediction in batch algorithms is *embarrassingly parallel*, allowing the use of multiple processors with almost linear speed improvements.

4.2.1 Stochastic Subgradient Descent

Arguably the most straight-forward way to approach Equation 4.14 is using subgradient descent. In light of the complexity of solving the loss-augmented prediction problem in Equation 4.15, it is natural to work in a stochastic setting (see Ratliff et al. [2007]). Given a model through Φ and a set of parameters θ, a subgradient considering a single training example (x^i, y^i) can be computed simply by solving the loss-augmented prediction problem:

$$\begin{aligned} &\frac{d}{d\theta}\left[\frac{1}{2}||\theta||^2 + C\ell(x^i, y^i, \theta)\right] \ni C\left[\Phi(x^i, \hat{y}) - \Phi(x^i, y^i)\right] + \theta \\ &\text{with } \hat{y} \in \operatorname*{argmax}_{y \in \mathcal{Y}} \Delta(y^i, y) + \theta^T \Phi(x^i, y) \end{aligned} \tag{4.16}$$

The most commonly used update has the simple form

$$\theta_{t+1} = (1 - \eta_t)\theta_t - \eta_t C\left[\Phi(x^i, \hat{y}) - \Phi(x^i, y^i)\right] \tag{4.17}$$

Here η_t is a sequence of step sizes. In practice the choice of η_t often strongly influences the convergence behavior. Many practitioners adopt the sequence proposed for binary SVMs in the Pegasos algorithm [Shalev-Shwartz et al., 2011]:

$$\eta_t = \frac{C}{t}, \tag{4.18}$$

which has been found to work well in many settings. Shalev-Shwartz et al. [2011] showed that this schedule achieves a convergence rate of $O(\frac{\ln T}{T})$. Lacoste-Julien et al. [2013] and Shamir and Zhang [2012] recently showed independently that a rate of $O(\frac{1}{T})$ can be achieved using a novel averaging scheme:

$$\bar{\theta}_T = \frac{2}{(T+1)(T+2)} \sum_{t=0}^{T} (t+1)\theta_t. \tag{4.19}$$

This t-weighted averaging can be computed on-the-fly as

$$\bar{\theta}_t = \frac{t}{t+2}\bar{\theta}_t + \frac{2}{t+2}\theta_{t+1}. \tag{4.20}$$

Implementation of the stochastic subgradient algorithm (with or without averaging) is straight-forward, but unfortunately detecting convergence is often tricky. It is possible to use mini-batches instead of processing one sample at a time to make use of multiple processors for loss-augmented prediction. Unfortunately, this negatively affects the number of iterations needed, and did not provide a benefit in our experiments.

4.2.2 The n-Slack Cutting Plane Method

The n-slack cutting plane method [Tsochantaridis et al., 2006] reformulates Equation 4.14 into a quadratic objective with a combinatorial number of constraints:

$$\begin{aligned} \min_{\theta,\xi_1,\ldots,\xi_k} \; & \frac{1}{2}||\theta||^2 + C\sum_{i=1}^{k}\xi_i \\ \text{s.t. for } i = 1,\ldots,k \; & \forall \hat{y} \in \mathcal{Y}: \\ & \theta^T[\Phi(x^i, y^i) - \Phi(x^i, \hat{y})] \geq \Delta(y^i, \hat{y}) - \xi_i \end{aligned} \tag{4.21}$$

Algorithm 2 n-Slack Cutting Plane Training of Structural SVMs

Input: training samples $\{(x^1, y^1), \ldots, (x^k, y^k)\}$, regularization parameter C, stopping tolerance ϵ.

Output: parameters θ, slack $(\xi_1, \ldots, \xi_k)$

1: $\mathcal{W}_i \leftarrow \varnothing, \xi_i \leftarrow 0$ for $i = 1, \ldots, k$

2: **repeat**

3: **for** i=1, . . . , k **do**

4: $\hat{y} \leftarrow I(x^i, y^i, \theta) := \underset{\hat{y} \in \mathcal{Y}}{\operatorname{argmax}} \Delta(y^i, \hat{y}) - \theta^T[\Phi(x^i, y^i) - \Phi(x^i, \hat{y})]$

5: **if** $\Delta(y^i, \hat{y}) - \theta^T[\Phi(x^i, y^i) - \Phi(x^i, \hat{y})] \geq \xi_i + \epsilon$ **then**

6: $\mathcal{W}_i \leftarrow \mathcal{W}_i \cup \{\hat{y}\}$

7: $(\theta, \xi_1, \ldots, \xi_k) \leftarrow \underset{\theta, \xi_1, \ldots, \xi_k}{\operatorname{argmin}} \frac{||\theta||^2}{2} + C \sum_{i=1}^{k} \xi_i$

$$\text{s.t. for } i = 1, \ldots, k \; \forall \hat{y} \in \mathcal{W}_i :$$

$$\theta^T[\Phi(x^i, y^i) - \Phi(x^i, \hat{y}^i)] \geq \Delta(y^i, \hat{y}^i) - \xi_i$$

8: **until** no $\mathcal{W}_i$ changes anymore.

As it is not feasible to deal with all constraints, only a working set $\mathcal{W}$ of active constraints is maintained, using the cutting plane method. The algorithm starts with an empty working set, and repeatedly iterates over the training data. For each sample, the most violated constraint is added to $\mathcal{W}$, and the quadratic program is solved again, with the new set of constraints. The algorithm terminates when no constraint can be found that is violated more than ϵ, which guarantees a suboptimality of at most ϵ. Tsochantaridis et al. [2006] showed a convergence rate of $O(\frac{1}{T^2})$ with respect to calls to the QP solver. In the worst-case, only a single new constraint could be found in one pass over the dataset, which leads to $O(\frac{1}{kT^2})$ in terms of calls to loss-augmented prediction. The recent work of Lacoste-Julien et al. [2013], however, suggests a rate of $O(\frac{1}{T})$, which empirically seems more plausible. This analysis does not include the cost of solving the QP, which depends on the size of the dataset and the number of iterations. The complete procedure is described in Algorithm 2.

The n-slack cutting plane is a sequential algorithm that processes each sample individually. While this allows fast process of the optimization with respect to the number of calls to loss-augmented prediction, individual steps become more and more costly. The number of the constraints is usually a multiple of the number of training samples, which leads to very large QP problems, even for medium sized datasets. This makes the algorithm often slow in practice. Solving the quadratic program can be accelerated using several techniques. We found that aggressively removing constraints that are inactive or contribute little to the solution often makes the difference between the algorithm being practical or not. Another possible technique is to update the quadratic program only every r samples, for some small integer r (Joachims et al. [2009] suggest $r = 100$). While this strategy on its own did not provide a large benefits in our experiments, it allows for parallel loss-augmented prediction on these mini-batches of size r.

4.2.3 The 1-Slack Cutting Plane Method

The 1-slack cutting plane method [Joachims et al., 2009] solves the following reformulation of Equation Equation 4.14:

$$\begin{aligned} &\min_{\theta,\xi} \frac{1}{2}||\theta||^2 + C\xi \\ &\text{s.t. } \forall \hat{\mathbf{y}} = (\hat{y}^1, \ldots, \hat{y}^n) \in \mathcal{Y}^n : \\ &\qquad \theta^T \sum_{i=1}^{n} [\Phi(x^i, y^i) - \Phi(x^i, \hat{y}^i)] \geq \sum_{i=1}^{n} \Delta(y^i, \hat{y}^i) - \xi \end{aligned} \tag{4.22}$$

Informally, the 1-slack formulation corresponds to joining all training samples into a single training example $(\mathbf{x}, \mathbf{y})$ that has no interactions between variables corresponding to different data points, and then applying the n-slack algorithm with a single data point. A detailed description can be found in Algorithm 3. By construction, only a single constraint is added in each iteration of Algorithm 3, leading to very small working sets $\mathcal{W}$. This has the advantage of producing a QP that easier to solve, as it contains far less variables than in the n-slack algorithm. The down-side of this is that the loss-augmented prediction problem has to be solved much more often until convergence. This is reflected in a convergence rate of $O(\frac{1}{kT})$, which scales with the inverse of the dataset size.

Joachims et al. [2009], who introduced the method, proposed two enhancements to make the algorithm more efficient:

Constraint Pruning As in the n-slack algorithm, members of the working set $\mathcal{W}$ can become inactive during learning. If a constraint has been inactive for a number of iterations, it is removed from $\mathcal{W}$, leading to smaller problem sizes.

Inference Caching In the 1-slack algorithm, each constraint is created using a combination of loss-augmented prediction results. Therefore, each of these predictions can be part of multiple constraints during learning. To exploit this, we maintain a set C^i of the last r results of loss-augmented prediction for each training example (x^i, y^i) (line 5 in Algorithm 3). For generating a new constraint $(\hat{y}^1, \ldots, \hat{y}^n)$, we find

$$\hat{y}^i \leftarrow \operatorname*{argmax}_{\hat{y} \in C^i} \sum_{i=1}^{n} \Delta(y^i, \hat{y}) - \theta^T \sum_{i=1}^{n} [\Phi(x^i, y^i) - \Phi(x^i, \hat{y})] \tag{4.23}$$

by enumeration of C^i and continue until line 8. Only if $\xi' - \xi < \epsilon$, that is the produced constraint is not violated strongly enough, we return to line 5 and actually invoke the loss augmented prediction I.

4.2.4 The BCFW Algorithm

Very recently, Lacoste-Julien et al. [2013] derived a very performant new algorithm. Starting from the Frank-Wolfe algorithm applied to the dual, they derived a block-coordinate version (BCFW) where each block corresponds to the constraints associated with a single training example. In this formulation, closed-form line search is possible, yielding a simple-to-implement algorithm. The algorithm processes a single training example at a time, and can be applied while remaining completely in the primal domain, making it applicable to very large datasets. Shalev-Shwartz and Zhang [2012] derived an alternative view of the algorithm, viewing it as coordinate descent in the dual with an additional proximal term. The BCFW algorithm has the same theoretical convergence rate of $O(\frac{1}{T})$ as averaged stochastic subgradient decent, but has two distinguishing advantages:

Algorithm 3 1-Slack Cutting Plane Training of Structural SVMs

Input: training samples $\{(x^i, y^i), \ldots, (x^i, y^i)\}$, regularization parameter C, stopping tolerance ϵ.
Output: parameters θ, slack ξ

1: $\mathcal{W} \leftarrow \emptyset$
2: **repeat**
3:
$$(\theta, \xi) \leftarrow \underset{\theta,\xi}{\operatorname{argmin}} \frac{||\theta||^2}{2} + C\xi$$
$$\text{s.t. } \forall \hat{\mathbf{y}} = (\hat{y}^1, \ldots, \hat{y}^k) \in \mathcal{W} :$$
$$\theta^T \sum_{i=1}^{k} [\Phi(x^i, y^i) - \Phi(x^i, \hat{y}^i)] \geq \sum_{i=1}^{k} \Delta(y^i, \hat{y}^i) - \xi$$
4: **for** i=1, . . . , k **do**
5: $$\hat{y}^i \leftarrow I(x^i, y^i, \theta) := \underset{\hat{y} \in \mathcal{Y}}{\operatorname{argmax}} \sum_{i=1}^{k} \Delta(y^i, \hat{y}) - \theta^T \sum_{i=1}^{k} [\Phi(x^i, y^i) - \Phi(x^i, \hat{y})]$$
6: $\mathcal{W} \leftarrow \mathcal{W} \cup \{(\hat{y}^i, \ldots, \hat{y}^i)\}$
7: $$\xi' \leftarrow \sum_{i=1}^{k} \Delta(y^i, \hat{y}^i) - \theta^T \sum_{i=1}^{k} [\Phi(x^i, y^i) - \Phi(x^i, \hat{y}^i)]$$
8: **until** $\xi' - \xi < \epsilon$

Stopping Criterion Both views of BCFW give rise to a dual objective, which can be used as a theoretically sound stopping criterion.

Learning Rate While there are several theoretical results on choosing the step size in stochastic subgradient descent, choosing a concrete schedule is often problematic in practice. By using analytic line-search, BCFW removes the need for any step size parameter, making application of the algorithm much simpler. The only disadvantage of BCFW compared to SSGD is the larger memory requirement. While SSGD only needs to store a single copy of the parameters θ, BCFW needs to store a separate (though possibly sparse) copy for each training sample.

In practice, the BCFW algorithm is commonly used with the weighted averaging described in Equation 4.20, as this is known to improve performance in the closely related stochastic subgradient algorithms, and empirically improves convergence. The detailed procedure as given by Lacoste-Julien et al. [2013] is shown in Algorithm 4.

Algorithm 4 BCFW

Input: training samples $\{(x^i, y^i), \ldots, (x^i, y^i)\}$, regularization parameter C, stopping tolerance ϵ.

Output: parameters θ

1: $\theta_0, \theta_0^i, \bar{\theta}_0 \leftarrow \mathbf{0}, \quad \ell_0, \ell_0^i, t \leftarrow 0$

2: **repeat**

3: $t \leftarrow t + 1$

4: Pick i uniformly at random from $\{1, \ldots, k\}$.

5: Perform loss-augmented prediction on sample i:

$$\hat{y} \leftarrow I(x^i, y^i, \theta) := \operatorname*{argmax}_{\hat{y} \in \mathcal{Y}} \Delta(y^i, \hat{y}) - \theta^T [\Phi(x^i, y^i) - \Phi(x^i, \hat{y})]$$

6: Compute parameter and loss updates based on sample i:

$$\theta_s \leftarrow \tfrac{C}{n} \Phi(x, \hat{y})$$

$$\ell_s \leftarrow \tfrac{C}{n} \Delta(y^i, \hat{y})$$

7: Compute optimum step size η:

$$\eta \leftarrow \tfrac{(\theta_t^i - \theta_s)^T \theta_t + C(\ell_s - \ell_k^i)}{\|\theta_t^i - \theta_s\|^2} \text{ and clip to } [0, 1]$$

8: Update per-sample parameters and loss estimate:

$$\theta_{t+1}^i \leftarrow (1 - \eta)\theta_{t+1}^i + \eta \theta_s$$

$$\ell_{t+1}^i \leftarrow (1 - \eta)\theta_{t+1}^i + \eta \ell_s$$

9: Update global parameters and loss estimate:

$$\theta_{t+1} \leftarrow \theta_{t+1} + \theta_t^i - \theta_{t+1}^i$$

$$\ell_{t+1} \leftarrow \ell_{t+1} + \ell_t^i - \ell_{t+1}^i$$

10: Compute the weighted running average:

$$\bar{\theta}_{t+1} = \tfrac{k}{k+2} \bar{\theta}_k + \tfrac{2}{k+2} \theta_{k+1}$$

11: **until** $(\theta - \theta_s)^T \theta - \ell + \ell_s \leq \epsilon$

where θ_s and ℓ_s are recomputed over the whole dataset.

4.3 Conditional Random Fields for Semantic Segmentation

4.3.1 Fundamentals of Conditional Random Fields

Structured models based on factor graphs, as described in Section 4.1.1 are often referred to as Conditional Random Fields (CRFs), pointing to their probabilistic interpretation. CRFs have been established as an important tool in many areas of computer vision. Applications include dense stereo, optical flow, inpainting, denoising, image editing, low-level segmentation and semantic segmentation.

Most conditional random fields only apply unary and pairwise potential functions. In this case, Equation 4.4 becomes

$$g(x, y) = \sum_{v \in V} \psi_v(x, y_v) + \sum_{(v,w) \in E} \psi_{v,w}(x, y_v, y_w). \tag{4.24}$$

Here V enumerates variables and $E \subset V \times V$ is a set of edges which represent pairwise factors.

There are broadly two types of models that are used: models in which each pixel in an image is represented as a variable, and models in which pixels are first grouped together into superpixels and each superpixel is represented as a variable. In pixel-based CRFs edges are usually introduced between adjacent pixels, that is using either 4-neighborhoods or 8-neighborhoods. Superpixel-based approaches introduce edges between adjacent superpixels, that is those that share a boundary. Figure 4.1 shows a sample image together with extracted superpixels and the neighborhood graph.

By nature, models over pixels and models over superpixels create graphs with a large number of loops. This complicates prediction and learning, and is the central topic of Chapter 6.

The motivation for using superpixels is that it is often easy to group nearby pixels that belong to the same object just based on local color or texture cues. While it is hard to even define what makes a sensible segmentation of an image, it is much easier to define an *over-segmentation*. The only requirement for an over-segmentation is that each segment only contains pixels belonging to the same entity.

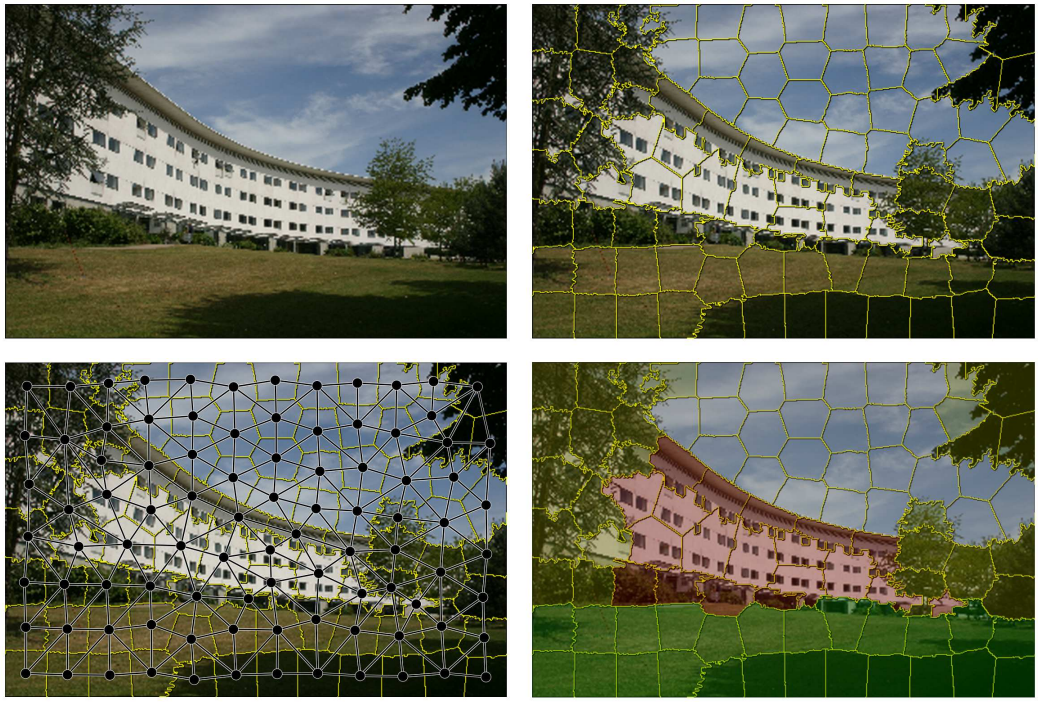

Figure 4.1: Schematic of CRFs over superpixels. From left to right: input image, superpixel segmentation using SLIC into about 100 superpixels, pairwise potential in the CRF, and desired labeling of superpixels.

Using superpixels instead of pixels has large computational advantages. A model that uses superpixels contains much fewer variables than one that works on pixel-level. Typical sizes for superpixels are in the hundreds of pixels, leading to a hundred-fold decrease in the number of variables. There are also semantic advantages. Using superpixels can constitute a form of regularization, including prior knowledge about the structure of images. Modeling interactions between superpixels also allows more distant parts of an image to interact, instead of modeling long-term relations only indirectly over neighbors. Finally, modeling decisions on a superpixel-level allows for more context for a local decision, while semantic decisions on pixel-level are not meaningful near object boundaries. The main disadvantage of superpixel-based approaches is that the initial over-segmentation can not be influenced by later reasoning. This can be a problem if fine structures need to be segmented for which local evidence does not suffice.

4.3.2 Data, Features and Superpixels

When evaluating learning algorithms in Chapter 5 and Chapter 6 we use the well-established PASCAL VOC 2010 and MSRC-21 datasets, shown in the introduction in Figure 1.1. As we focus on the learning algorithms in these chapters, we use features for unary potentials from the literature.

For both datasets we use the TextonBoost class probabilities provided by Krähenbühl and Koltun [2012]. For the PASCAL VOC dataset, the potentials provided by Krähenbühl and Koltun [2012] also include the responses of object detectors, to better capture the complex object classes. We average these potentials inside superpixels and use the resulting feature as input to our unary potentials. For the MSRC-21 dataset, we compute TextonBoost on two different scales, as suggested by Mottaghi et al. [2013]. We additionally extract SIFT and color descriptors and create bag-of-word descriptors for each superpixel. Following the approach of Lucchi et al. [2011] we augment these with a global bag-of-word descriptor for each image. We train a linear SVM using an approximation to the additive χ^2 kernel [Vedaldi and Zisserman, 2010] and use the response as an additional input to our CRF. This *piecewise training* simplifies learning and was found to have little effect on accuracy [Nowozin et al., 2010]. In total, there are 63 features for the MSRC-21 dataset, 21 for each scale of TextonBoost and an additional 21 for the bag-of-word model using an SVM.

We use the SLIC [Achanta et al., 2012] algorithm to create superpixel for all our experiments. It has been shown to provide competitive results with a minimum of computational complexity. Algorithmically, SLIC simply computes a k-means clustering over pixels. Each pixels is represented as 5D point, using three color channels and the x and y coordinates in the image. In a post-processing step, small segments are removed. To make clustering of so many points feasible, the search for the nearest cluster in k-means is restricted to a local neighborhood in the image. An example of superpixels computed with the SLIC algorithm can be found in Figure 4.1. The features and superpixel algorithm used for the RGB-D dataset NYU used in Chapter 7 will be discussed there.

4.3.3 Previous Work

Conditional random fields were first used in the context of semantic segmentation by He et al. [2004], who used contrastive divergence for learning parameters. Many other early models, such as the one of Shotton et al. [2006], did not learn parameters at all, but used *contrast-sensitive Potts potential*. These potentials have a smoothing effect over labels by encouraging neighboring variables to take the same value. The penalty for taking different values is dependent on color contrast between the pixels.

This smoothing approach was improved upon by Kohli et al. [2009], who introduces higher order potentials to enforce label consistency within larger regions. Later, Ladicky et al. [2009] proposed a hierarchical model over pixels and superpixels, including the higher order potentials of Kohli et al. [2009] and additional lateral and hierarchical connections. A similar approach, using superpixels as the finest resolution, and additionally modelling object class co-occurrences was suggested by Gonfaus et al. [2010]. However, all these models did not learn the potentials of the CRF model in a principled manner. The above approaches all learned unary potentials using a non-structured approach, for example SVMs or boosting, and then set pairwise and higher order potentials in the model by hand.

Szummer et al. [2008] on the other hand used a structured support vector machine approach to learn unary and pairwise parameters. They use the classical graph cut approach of Boykov et al. [2001] for inference. However, graph-cut inference is only applicable to submodular energies, and therefore severely restricts the expressiveness of the resulting model. Lucchi et al. [2011] investigated the importance of global constraints, using an approach similar to Szummer et al. [2008], but also learning global interactions—however, these did not improve performance, compared to simply including global descriptors into local classifiers.

The problem of approximate inference was addressed elegantly by Yao et al. [2012], who learn a joint model for scene classification, object localization and semantic segmentation. Their work is based on Hazan and Urtasun [2010], who integrate learning and inference in a joint optimization problem.

4.4 Casting Structured Prediction into Software

Unfortunately only few implementations for learning structured prediction are publicly available—many applications are based on the non-free implementation of Joachims et al. [2009]. In this section, we introduce our implementation, PyStruct, which aims at providing a high-quality code with an easy-to-use interface, in the high-level Python language. This allows practitioners to efficiently test a range of models, as well as allowing researchers to compare to baseline methods much more easily than this is possible with current implementations. PyStruct is BSD-licensed, allowing modification and redistribution of the code, as well as use in commercial applications. By embracing paradigms established in the scientific Python community and reusing the interface of the widely-used scikit-learn library [Pedregosa et al., 2011], PyStruct can be used in existing projects, replacing standard classifiers. The online documentation and examples help new users understand the somewhat abstract ideas behind structured prediction. We base the experiments in the rest of this work on the algorithms implemented in PyStruct. In the following, we briefly discuss implementation and features of the PyStruct library.

4.4.1 Library Structure and Content

Using the formulation of structured prediction introduced in Section 4.1, learning can be broken down into three sub-problems:

1. Encoding the structure of the problem in a joint feature function Φ.
2. Solving the loss-augmented prediction problem in Equation equation 4.15.
3. Optimizing the objective in Equation 4.14 with respect to θ.

The first two problems are usually tightly coupled, as the maximization in Equation 4.2 is usually only feasible by exploiting the structure of Φ, as described in Section 4.1.1. The last problem, finding θ, on the other hand, is usually treated as independent. PyStruct takes an object-oriented approach to decouple the task-dependent implementation of 2. and 3. from the general algorithms used to solve 1.

Estimating θ is done in `learner` classes, which currently support cutting plane algorithms for structural support vector machines (SSVMs), the BCFW algorithm for SSVMs, subgradient methods for SSVMs, the structured perceptron and latent

Package	1-SP	n-SP	SSGD	BCFW	L-SSVM	Perceptron	ML
PyStruct	✓	✓	✓	✓	✓	✓	×
SVM^{struct}	✓	✓	×	×	✓	×	×
Dlib	✓	×	×	×	×	✓	×
CRFsuite	×	×	×	×	×	✓	✓

Table 4.1: Comparison of learning algorithms implemented in popular structured prediction software packages. 1-CP stands for 1-slack Cutting Plane, n-CP for n-slack Cutting plane, SSGD for stochastic subgradient decent learning of SSVMs, BCFW is as described in section 4.2.4, L-SVM stands for latent variable SSVMs, and ML for maximum likelihood learning.

Package	Multi-Class	Multi-Label	Chain	Graph	LSVM	LDCRF
PyStruct	✓	✓	✓	✓	✓	✓
SVM^{struct}	✓	✓	×	×	×	×
Dlib	✓	×	✓	✓	×	×
CRFsuite	×	×	✓	×	×	×

Table 4.2: Comparison of models implemented in popular structured prediction software packages. LSVM stands for the latent multi-class SVM, LDCRF for latent dynamic conditional random fields.

variable SSVMs. See Section 4.2 for a detailed description of the algorithms. The cutting plane implementation uses the cvxopt package [Dahl and Vandenberghe, 2006] for quadratic optimization.

Encoding the structure of the problem is done using `model` classes, which compute Φ and encode the structure of the problem. PyStruct implements models for many common cases, such as multi-class and multi-label classification, conditional random fields with constant or data-dependent pairwise potentials, and several latent variable models. The maximization for finding y in Equation 4.2 is carried out using highly optimized implementations from external libraries. PyStruct includes support for using OpenGM [Kappes et al., 2013], LibDAI [Mooij, 2010], fusion moves [Rother et al., 2007, Lempitsky et al., 2010], and AD^3 [Martins et al., 2011]. It also includes an interface to a general purpose linear programming solver from cvxopt [Dahl and Vandenberghe, 2006].

Table 4.1 and Table 4.2 list learning algorithms and models that are implemented in PyStruct and compares them to other publicly available structured prediction libraries.

4.4.2 Project Goals

Modularity PyStruct separates the algorithms for parameter estimation and inference from the task-dependent formulation of Φ. This allows practitioners, for example in computer vision or natural language processing, to improve their model without changing any optimization code. On the other hand, researchers working on better inference or parameter learning can easily benchmark their improvements on a wide array of applications.

Completeness PyStruct aims at providing complete predictors that can be used directly in applications. It contains model formulation for many typical scenarios. This is in contrast to SVM$^{\text{struct}}$ that provides no models at all, requiring the user to develop significant amounts of code, even for simple tasks.

Efficiency While PyStruct focuses on usability, providing efficient and competitive implementations is important to allow fast prototyping and scaling to large datasets. PyStruct achieves the same runtime performance as the popular SVM$^{\text{struct}}$ model for cutting plane algorithms, and provides implementations of the BCFW and Subgradient methods that scale to large datasets.

Documentation and Examples PyStruct provides full documentation of all classes and functions. It also provides examples for many important applications, such as sequence tagging, multi-label classification and image segmentation. Furthermore, standard benchmarks are included as examples, which allows easy comparison with the literature.

Integration To improve usability, PyStruct is interoperable with other numeric and scientific Python projects, such as scikit-learn [Pedregosa et al., 2011], mahotas [Coelho, 2013], gensim [Řehůřek and Sojka, 2010], and scikit-image. This allows users to build powerful applications with little effort. In particular, most of the model-selection methods of scikit-learn can be used directly with PyStruct.

Testing PyStruct contains a testing-suite with 80% line-coverage. It also employs continuous integration to ensure stability and a seamless user experience.

Listing 1 Example of defining and learning a CRF model.

```
1 model = crfs.EdgeFeatureGraphCRF(
2             class_weight=inverse_frequency,
3             symmetric_edge_features=[0, 1],
4             antisymmetric_edge_features=[2],
5             inference_method='qpbo')
6
7 ssvm = learners.NSlackSSVM(model, C=0.01, n_jobs=-1)
8 ssvm.fit(X, Y)
```

4.4.3 Usage Example: Semantic Image Segmentation

We demonstrate the use of PyStruct on the task of semantic image segmentation, the main focus of this work. The example shows how to learn an n-slack support vector machine on a superpixel-based CRF on the Pascal VOC dataset. Details of the experiment can be found in Section 4.3. Each sample (corresponding on one entry of the list `X`) is represented as a tuple consisting of input features and a graph representation.

The source code is shown in Listing 1. Lines 1–5 declare a model using parametric edge potentials for arbitrary graphs. Here `class_weight` re-weights the Hamming loss according to inverse class frequencies. The parametric pairwise interactions have three features: a constant feature, color similarity, and relative vertical position. The first two are declared to be symmetric with respect to the direction of an edge, the last is antisymmetric. We use fusion moves for inference. Line 5 creates a `learner` object that will learn the parameters for the given model using the n-slack cutting plane method, and line 6 performs the actual learning. Using this simple setup, we achieve an accuracy (Jaccard index) of 30.3 on the validation set following the protocol of Krähenbühl and Koltun [2012], who report 30.2 using a more complex approach. Training the structured model takes approximately 30 minutes using a single i7 core.

4.4.4 Experiments

While PyStruct focuses on usability and covers a wide range of applications, it is also important that the implemented learning algorithms run in acceptable time. In this section, we compare our implementation of the 1-slack cutting plane

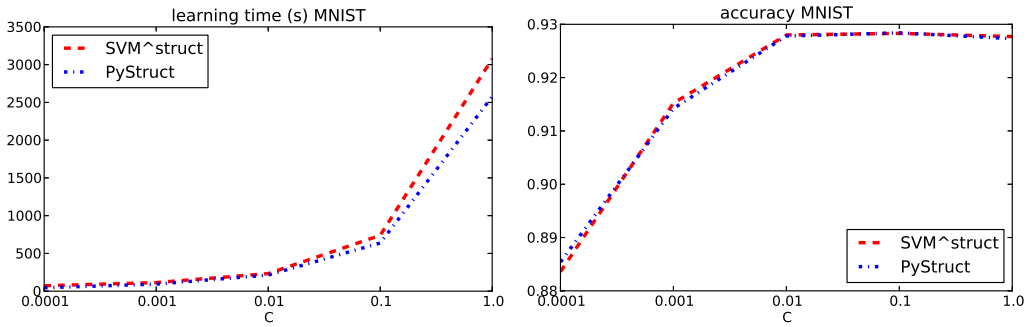

Figure 4.2: Runtime comparison of PyStruct and SVM$^{\text{struct}}$ for multi-class classification.

algorithm with the implementation in SVM$^{\text{struct}}$. We compare performance of the Crammer-Singer multi-class SVM with respect to learning time and accuracy on the MNIST dataset of handwritten digits. While multi-class classification is not very interesting from a structured prediction point of view, this problem is well-suited to benchmark the cutting plane solvers, as loss-augmented prediction is trivial.

Results are shown in Figure 4.2. We report learning times and accuracy for varying regularization parameter C. The MNIST dataset has $60\,000$ training examples, 784 features and 10 classes.The setup of the experiment is the same as in Chapter 5. The figure indicates that PyStruct has competitive performance, while using a high-level interface in a dynamic programming language.

4.5 Summary

In this chapter, we introduced basic concepts of structured prediction. In particular, we discussed Structured Support Vector Machines, a max-margin approach for training linear predictors for structured data. We gave a description of several popular learning algorithms, together with their theoretical runtime bounds and practical considerations. We discussed the use of CRFs for object class segmentation and semantic segmantion, and superpixel-based approaches.

We also introduced PyStruct, our modular structured learning and prediction library in Python. PyStruct is geared towards ease of use, while providing efficient implementations and is be the basis of our further experiments. PyStruct integrates itself into the scientific Python ecosystem, making it easy to use with existing libraries and applications. Currently, PyStruct focuses on max-margin and perceptron-based approaches. In the future, we plan to integrate other paradigms, such as sampling-based learning [Wick et al., 2011], surrogate objectives (for example pseudo-likelihood), and approaches that allow for a better integration of inference and learning [Meshi et al., 2010].

5 Empirical Comparison of Learning Algorithms

In this chapter, we provide an empirical evaluation of the learning algorithms described in Chapter 4. We use the open source implementations in PyStruct, and publish the evaluation code and datasets with the package.

5.1 Datasets and Models

We consider several qualitatively different datasets that have been widely used in the literature. The problems we consider are multi-class classification, sequence labeling, multi-label prediction, and general graph labeling.

5.1.1 Multi-Class Classification (MNIST)

The simplest task we use is multi-class classification. In this task, the inference problem is trivial, as it is assumed the target set is small enough to be enumerated efficiently. While this problem could be solved more efficiently with specialized algorithms, it nevertheless provides an initial insight into structured prediction algorithms. We choose the classical MNIST dataset of handwritten digits, consisting of 60000 training images and 10000 test images of the digits zero to nine. Figure 5.1 shows some examples. Each image is a 28×28 grey-level image, resulting in a 784-dimensional feature vector. We normalize the features between 0 and 1. The model we use for multi-class classification is the Crammer-Singer

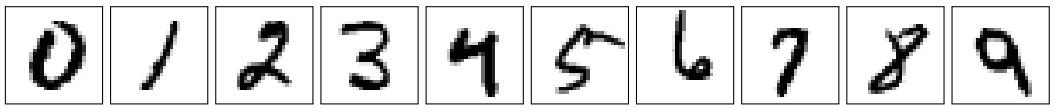

Figure 5.1: Visualization of samples from the ten classes in the MNIST dataset.

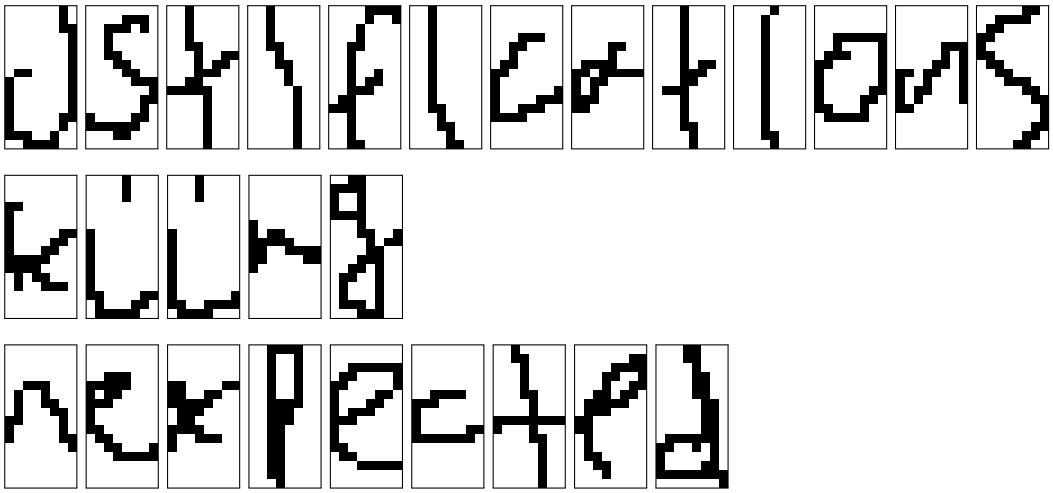

Figure 5.2: Visualization of some words from the OCR dataset. The first letters were removed by Taskar et al. [2003] as they were capitals. The words are: (j)ustifications, (s)kiing and (u)nexpected. The letters in parentheses are not part of the dataset.

formulation. We do not include a bias, leading to $784 \cdot 10 = 7840$ parameters in the model.

5.1.2 Sequence Labeling (OCR)

A classical application of structured prediction is sequence labeling. We choose the "OCR" dataset introduced in the seminal work of Taskar et al. [2003]. Each input consists of the segmented handwritten letters of a word in lower case. The task is to classify all letters in a word, that is, assign each segmented input letter to one of the classes "a" to "z". As the first letter of each word was capitalized, these were removed by Taskar et al. [2003], leading to somewhat odd-looking labels. The words are between three and fourteen letters long, and each letter is represented as a binary image of size 16, with a total of 6877 words. The dataset is divided into ten folds. We consider two setups: learning on one fold, and testing on the remaining nine folds, following Taskar et al. [2003], and learning on nine folds and testing on the remaining fold, following Lacoste-Julien et al. [2013]. We refer to the learning on one fold as OCR-SMALL, and learning on nine folds as OCR-BIG. We use a simple chain model with a single constant pairwise feature. This means that the unary potential has $16 \cdot 8 \cdot 26 = 768$ parameters, one for

each input feature and output class. The pairwise potential consists of a matrix of transition potentials with $26 \cdot 26 = 676$ entries. It is well-known that efficient exact inference in chains is possible using message passing algorithms.

5.1.3 Multi-Label Classification

Multi-label classification is a generalization of multi-class classification, in which each example can be associated with more than one class. In other words, the algorithm must decide for each sample and for each class whether the sample belongs to that class or not. Multi-label classification was first formulated as a structured prediction problem by Finley and Joachims [2008], who used to investigate the influence of approximate inference on the n-slack cutting plane algorithm. In their formulation each class is represented as a binary node in a factor graph—the states representing presence or absence of the class. A different factor of pairwise potentials is introduced between each pair of classes. We also consider a different model, where pairwise potentials are only introduced between specific nodes. We build a tree over the binary variables by computing the Chow-Liu Tree [Chow and Liu, 1968] of the targets. While this results in a less expressive model, a tree-shaped model allows for exact inference via message passing. We use two of the datasets used in Finley and Joachims [2008], the SCENE and YEAST datasets. We choose these two as these are real-world datasets for which the pairwise approach outlined above actually improves upon the baseline [Finley and Joachims, 2008]. The SCENE dataset has six labels, 294 input features, 1211 training samples and 1196 test samples. This leads to $204 \cdot 6 = 1224$ parameters for the unary potentials, $3 \cdot 5 \cdot 4 = 60$ parameters for the full pairwise potentials (four for each edge), and $5 \cdot 4 = 20$ parameters for the pairwise potentials of the tree-shaped model. Using four parameters for each edge is a slight over-parametrization that simplifies writing out the model. The YEAST dataset has 14 labels, 103 features, 1500 training samples and 971 test samples. The resulting numbers of parameters are $14 \cdot 103 = 1442$ parameters for the unary potentials, $7 \cdot 13 \cdot 4 = 364$ parameters for the full pairwise potential, and $13 \cdot 4 = 52$ parameters for the tree-shaped model.

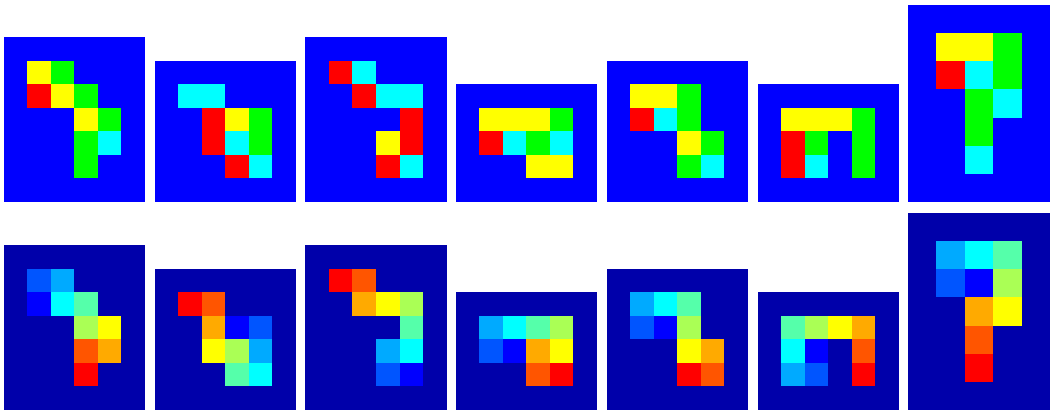

Figure 5.3: Visualization of the SNAKES dataset. The top row shows input patterns, the bottom row the corresponding labels. The colors in the image showing the labels correspond to dark blue for background, and encoding the length of the snake from head (red) to tail (blue).

5.1.4 2D Grid CRF (Snakes)

The SNAKES dataset is a synthetic dataset where samples are labeled 2D grids. It was introduced by Nowozin et al. [2011] to demonstrate the importance of learning conditional pairwise potentials. The dataset consists of "snakes" of length ten traversing the 2D grid. Each grid cell that was visited is marked as the snake heading out towards the top, bottom, left, or right, while unvisited cells are marked as background. The goal is to predict a labeling of the snake from "head" to "tail", that is, assigning numbers from zero to nine to the cells that are occupied by the snake. Figure 5.3 illustrates the principle. Local evidence for the target label is weak except around head and tail, making this a challenging task, requiring strong pairwise potentials. The dataset is noise-free in the sense that given the above description, a human could easily produce the desired labeling without making any mistake. The dataset is also interesting as the model proposed by Nowozin et al. [2011] produced notoriously hard-to-optimize energy functions.

Originally, the input is encoded into five RGB colors ("up", "down", "left", "right", "background"). To encode the input more suitably for our linear methods, we convert this representation to a one-hot encoding of the five states.

We use a grid CRF model for this task. Unary potentials for each node are given by the input of the 8-neighborhood of the node—using a 4-neighborhood would most likely yield better results, but we do not want to encode too much task-knowledge into our model. Using the one-hot encoding of the input, this

leads to $9 \cdot 5 = 45$ unary features. With 11 output classes, the unary potential has $11 \cdot 9 \cdot 5 = 495$ parameters. Features for the pairwise potentials are constructed by concatenating the features of the two neighboring nodes, taking the direction of the edge into account. The pairwise feature therefore has dimensionality $45 \cdot 4$, with the first 45 entries corresponding to the feature of the "top" node, the second 45 entries to the features "bottom" node, followed by the "left" and "right" nodes. As each edge is either horizontal *or* vertical, only two of these parts will be non-zero for any given edge. With $45 \cdot 4$ edge features, the pairwise potentials have $45 \cdot 4 \cdot 11^2 = 21780$ parameters.

5.1.5 Superpixel CRFs for Semantic Segmentation

Our main attention is devoted to the use of conditional random fields for semantic segmentation. We use the PASCAL VOC and MSRC datasets. The PASCAL VOC dataset has 964 training images, each divided into around 100 superpixels. There are 20 object categories, and an additional background class. The MSRC-21 dataset has 276 training images, also segmented into around 100 superpixels each. There are 21 semantic classes in the MSRC-21 dataset. Each superpixel is represented as an output variable, with the ground truth obtained by majority vote over the pixels belonging to the superpixel. We removed all superpixels in which the majority of pixels is labeled "void", leading to some samples having much fewer than 100 variables. Pairwise potentials are introduced for each pair of neighboring superpixels. We use the unary and pairwise potentials described in Section 4.3: 21 unary features for the PASCAL VOC dataset and 63 unary features for the MSRC-21 dataset. We use the same three pairwise features for both datasets: a constant feature, a color feature and a feature encoding relative vertical position. Overall, this results in $21 \cdot 21 + 3 \cdot 21 \cdot 21 = 1764$ parameters for the potentials on PASCAL VOC and $63 \cdot 21 + 3 \cdot 21 \cdot 21 = 2646$ parameters for the potentials on the MSRC-21 dataset.

Dataset	samples	variables	graph	dim θ	labels
MNIST	60000	1	none	7840	10
OCR-SMALL	704	3–14	chain	1444	26
OCR-LARGE	6173	3–14	chain	1444	26
SCENE-TREE	1211	6	tree	1244	2
SCENE-FULL	1211	6	loopy	1284	2
YEAST-TREE	1500	14	tree	1494	2
YEAST-FULL	1500	14	loopy	1806	2
SNAKES	200	84–168	loopy	22275	11
MSRC-21	276	7–113	loopy	1764	21
PASCAL VOC	964	8–112	loopy	2646	21

Table 5.1: Summary of datasets used in the evaluation.

5.2 Experiments

We compare the following algorithms on the above models (see Section 4.2 for details):

- Stochastic Subgradient Descent (SSGD) using the Pegasos schedule for step-sizes
- The 1-slack cutting plane algorithm without inference caching
- The 1-slack cutting plane algorithm with caching the last 50 inference results for each sample (see Chapter 6 for details)
- The n-slack cutting plane algorithm, where the QP is solved after each sample
- The n-slack cutting plane algorithm, where the QP is solved every 100 samples
- The BCFW algorithm with weighted averaging

All algorithms take a C parameter, which we adjusted on a fully trained model on a validation set (experiments not reported here) and held constant for all models. We found, however, that the algorithms and models are quite robust to the choice of C within one or two orders of magnitude. As stopping criterion, we used a duality gap of 0.1 when possible, and a pre-defined number of iterations for

the subgradient algorithms. It is worth noting that the quadratic programming based algorithms have additional hyper-parameters that we do not discuss here in detail, such as the threshold for removing a constraint as inactive, how often inactive constraints are removed, and parameters of the underlying QP solver. On the other hand, the BCFW algorithm has no hyper-parameters except for the stopping criterion.

Our ultimate evaluation criterion is how fast the learning algorithms converge, and how robust they are to approximate inference. To quantify our goals, we track primal suboptimality and training set error during learning. We report primal suboptimality as a function of runtime, and as a function of passes over the training set. Both are informative to practitioners, as they give important insight into the working of the algorithm. The actual runtime is arguably the most relevant factor, but also highly influenced by implementation details and properties of the dataset.

We are particularly interested in cases where inference is highly non-trivial, making it the dominating factor with respect to runtime. We define one pass over the dataset as calling prediction once for each sample. This way, we can get a clear picture of how learning times scale with complexity of the inference task. For the subgradient and BCFW algorithms, learning time is linear in the number of passes over the dataset, while caching and solving of a QP makes the learning time depend on the number of iterations in hard-to-predict ways. All algorithms were run on a single core i7 processor.

5.2.1 Experiments using Exact Inference

In this section, we present results on multi-class classification, sequence labeling and multi-label prediction. In these tasks, exact inference is possible, and we can compute the exact objective of the various algorithms easily. We use the dynamic programming algorithm implemented in OpenGM [Kappes et al., 2013] for experiments using the tree and chain models. For the full pairwise multi-label model, we use branch-and-bound together with the AD^3 [Martins et al., 2011] algorithm. We found that using the linear programming relaxation on the SCENE dataset is often tight, and there was no need to resort to approximate inference. Inference on the YEAST dataset was more complex, and on the verge of being non-practical. We use this dataset as an example for very costly inference.

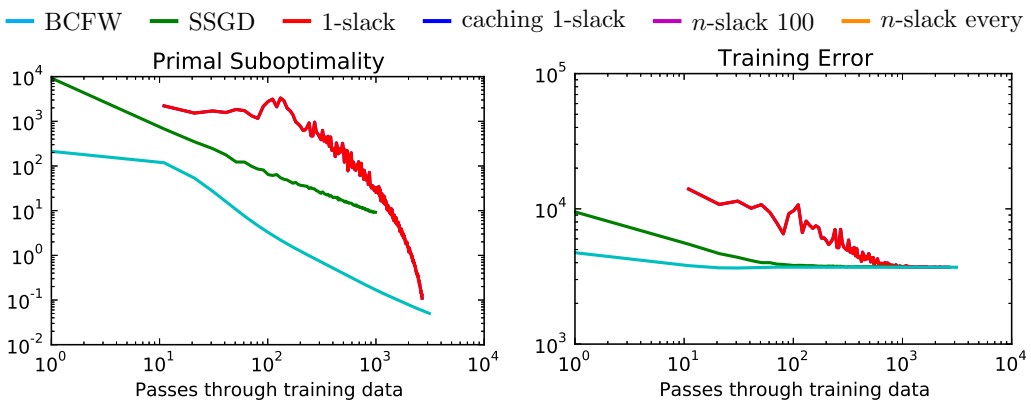

Figure 5.4: Convergence of the primal suboptimality and training set loss on MNIST.

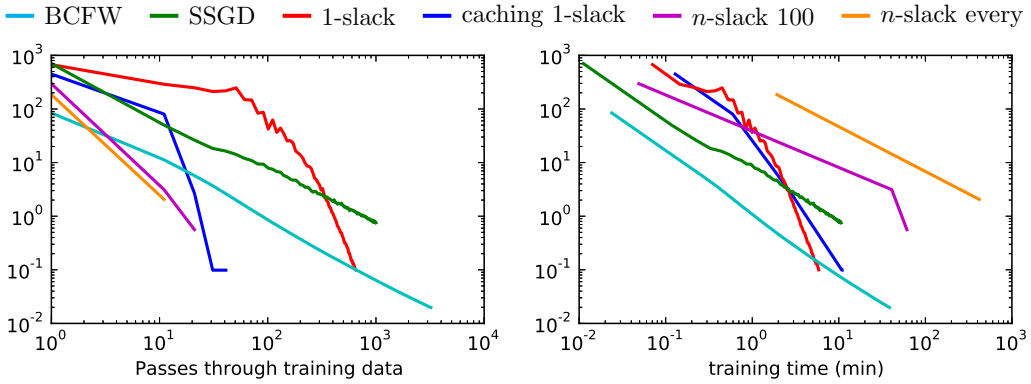

Figure 5.5: Convergence of the primal suboptimality on OCR-small.

We start our experiments with the MNIST dataset. The convergence of the primal suboptimality and training set loss in terms of passes over the training set is shown in Figure 5.4. While inference is trivial, this is by far the largest dataset, making the n-slack algorithms not feasible. We see that the stochastic algorithms are much faster than the cutting plane algorithm, in particular initially, even in terms of iterations, with the BCFW clearly leading the way. Interestingly, the 1-slack algorithm catches up near the desired suboptimality. Looking at the training set loss in Figure 5.4 suggests that the high precision we demanded was not necessary, and we could have terminated the stochastic algorithms much earlier.

We now consider datasets with non-trivial inference. The results for OCR-SMALL are shown in Figure 5.5. Considering the plot against passes over the dataset

on the left, several trends can be observed: the n-slack cutting plane algorithms converge fastest, with the version that recomputes the QP at every step leading the race. The algorithms are closely followed by the caching 1-slack cutting plane algorithm. They are followed by the significantly slower BCFW algorithm, and finally the non-caching 1-slack cutting plane algorithm and subgradient descent. These results are intuitive, as they reflect "how much work" each algorithm does for each loss-augmented prediction step. More work towards the objective leads to faster convergence. This "more work" is quantified on the right hand side of Figure 5.5, where the suboptimality is plotted against time. Clearly the n-slack algorithms do "too much" work, leading to very slow convergence. Also, caching does not speed up learning on this dataset.

We want to highlight an interesting phenomenon here: The cutting-plane algorithms stop immediately when reaching the desired suboptimality of 0.1. The BCFW algorithm, on the other hand, keeps on learning much longer—even though the primal of the 1-slack algorithms is always above the primal of the BCFW. This is caused by a looser bound given by the dual. Remember that the stopping criterion for both cutting-plane and BCFW are given by the duality gap. It seems that while the primal objective converges much faster in the BCFW algorithm than in the non-caching 1-slack algorithm, the dual does not. This means that in this practical experiment, the non-caching 1-slack algorithm was *faster* in guaranteeing the desired suboptimality, and therefore in terminating. Figure 5.6 illustrates this point by plotting primal and dual objectives for the 1-slack cutting plane and the BCFW algorithms. Taking a closer look, we find that the same behavior occurred for MNIST, which is also shown in Figure 5.6.

The results for the OCR-large dataset are shown in Figure 5.7. It was not feasible to run the n-slack algorithm when solving the QP at every step here. The trends are the same as for the smaller dataset: n-slack and caching 1-slack cutting plane are very fast in terms of passes through the dataset, but n-slack cutting plane is impractical slow with respect to runtime. The BCFW algorithm converges fast initially, but the 1-slack cutting plane is faster at high precision and faster in certifying the desired duality gap.

Next, we consider the multi-label task. The results for the YEAST and SCENE datasets are shown in Figure 5.8. The caching 1-slack algorithm is much faster than the non-caching one on both datasets and both graph structures. While the two variants of the n-slack algorithm are equally fast with respect to passes over

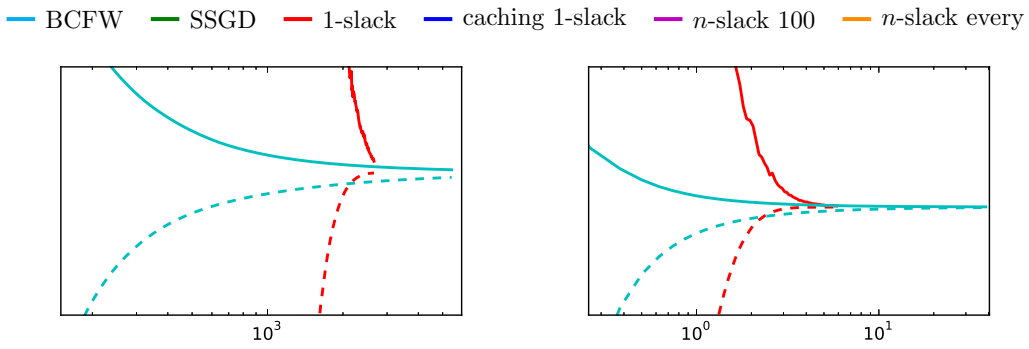

Figure 5.6: Primal objective and dual objective (dashed lines) for BCFW and 1-slack cutting plane on MNIST (left) and OCR-SMALL (right).

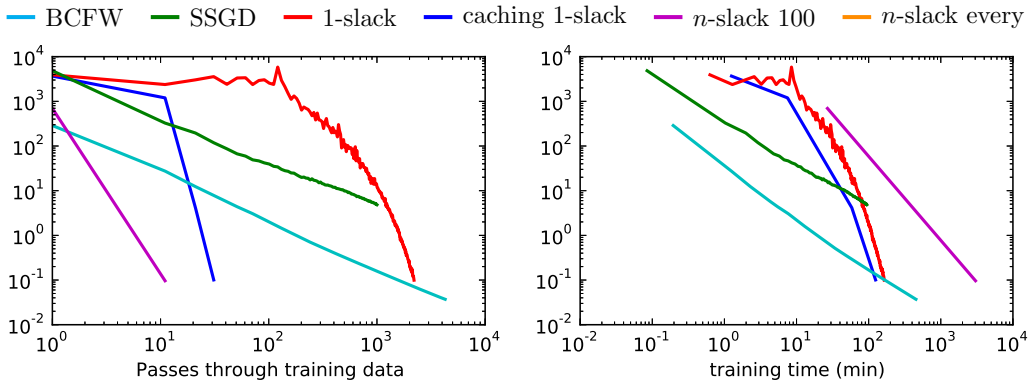

Figure 5.7: Convergence of the primal suboptimality on OCR-large.

the training data, solving the QP every 100 steps is among the fastest algorithms, while solving the QP at every step is among the slowest. We applied this method only to the smaller SCENE dataset, as applying it on the YEAST dataset was not practical.

The caching 1-slack algorithm is the fastest algorithm to achieve the desired primal suboptimality for tree-structured graphs, while it is out-performed by the n-slack cutting plane algorithm for the full graphs. This is intuitive, as the additional work the n-slack algorithm does becomes more valuable, the longer the inference takes.

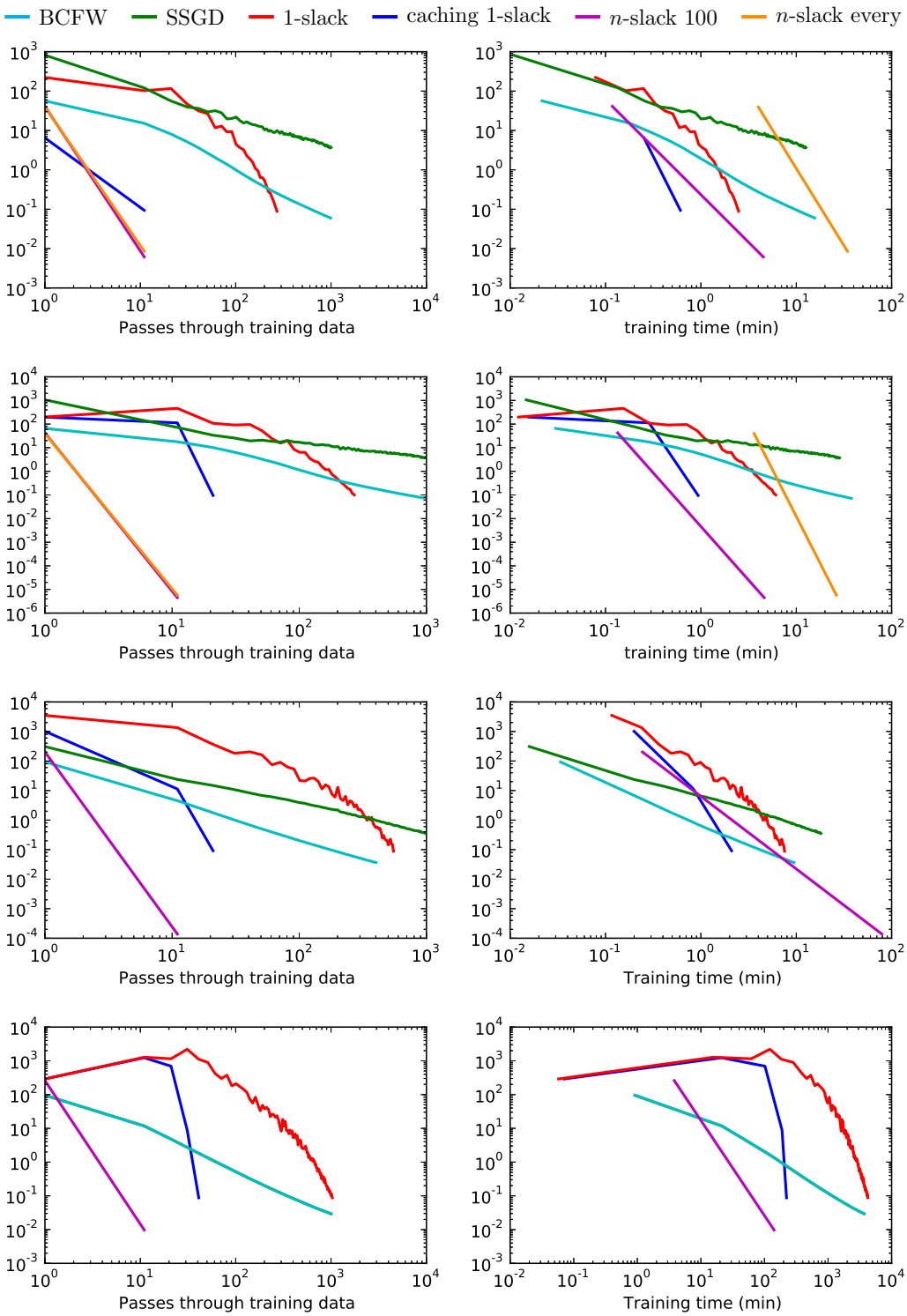

Figure 5.8: Convergence of the primal suboptimality on the multi-label datasets. From top to bottom: the SCENE dataset using a tree model, the SCENE dataset using a full model, the YEAST dataset using a tree model, and the YEAST dataset using a full model.

5.2.2 Experiments using Approximate Inference

For the remaining tasks, the SNAKES dataset and image segmentation, exact inference is intractable for learning. We use two different approximate inference algorithms, fusion moves [Lempitsky et al., 2010], a fast local search procedure, and the linear programming relaxation provided by AD^3 [Martins et al., 2011] not using branch-and-bound, in contrast to the multi-label setup above.

Evaluation of algorithms where exact inference is not possible is much harder, as it is usually not possible to evaluate the exact objective. When using AD^3, we can still consider the relaxed task, which will be solved exactly in the majority of cases (AD^3 can fail to find the primal solution to the LP relaxation). When using fusion moves, there is no obvious interpretation of the approximate primal objective, other than that it provides a lower bound of the actual objective. Nevertheless, we find it informative to analyze the behavior of this lower bound. In contrast to the previous section, we show primal and dual objective values instead of the primal suboptimality when using approximate inference. The dual values do provide lower bounds to the exact objective, and are therefore somewhat more informative here, see Chapter 6 for a discussion.

For both inference algorithms, we also evaluate the predictive performance on the training set. We explicitly do not consider the relaxed problem here—instead we round possible fractional results. While the training error does not directly reflect the objective, it provides a measure of how effective the "prediction machine" given by the learned model together with the inference algorithm is.

Figure 5.9 shows the results on the SNAKES dataset. The top row shows the objective when learning with fusion moves, the next row the training set loss. First, we observe that using fusion moves, even the approximate objective does not reach the desired optimality gap for the 1-slack and BCFW algorithms. This is caused by the algorithms not finding any further constraints. The n-slack algorithm fares a bit better and achieves a higher dual value. Looking at the training set loss, it is clear that the stochastic algorithms have an advantage, possibly caused by not stopping too early. Still, all algorithms ultimately fail to solve the task.

The situation is very different when using the linear programming relaxation. Looking at the last two rows of Figure 5.9, we see that all algorithms are able to solve the task perfectly. Evaluating on the test-set, all algorithms achieve an accuracy of around 99.5%, confirming the models actually learned the task. Surprisingly, SSGD was very successful on this dataset, even faster then the non-caching n-slack algorithm. Caching provides nearly an order of magnitude speed-up on the dataset. Using branch-and-bound to obtain exact results was not feasible on this dataset.

The last set of experiments is on the segmentation dataset MSRC-21 and PASCAL VOC. Because of the large number of labels and samples, it is only feasible to use the fusion move inference algorithm. The results are shown in Figure 5.10, with the plots in the two upper rows showing results on MSRC-21, and the tow rows below showing results on PASCAL VOC. First, we notice that all algorithms achieve low (approximate) duality gaps, and none of the algorithms terminates prematurely. Also, all algorithms achieve similar dual objectives and similar training errors after convergence. In both datasets, the 1-slack algorithm is somewhat slow in minimizing the training set error, and the n-slack and stochastic algorithms are much faster. Solving the QP only every 100 steps in the n-slack algorithm significantly slows down learning on both datasets. This is in contrast to the YEAST datasets, where the converse was true, even though YEAST has a similar number of training examples to PASCAL VOC.

Given the outcome described above, we suspect that learning was not affected by approximate inference too much. We leave a more detailed analysis for Chapter 6.

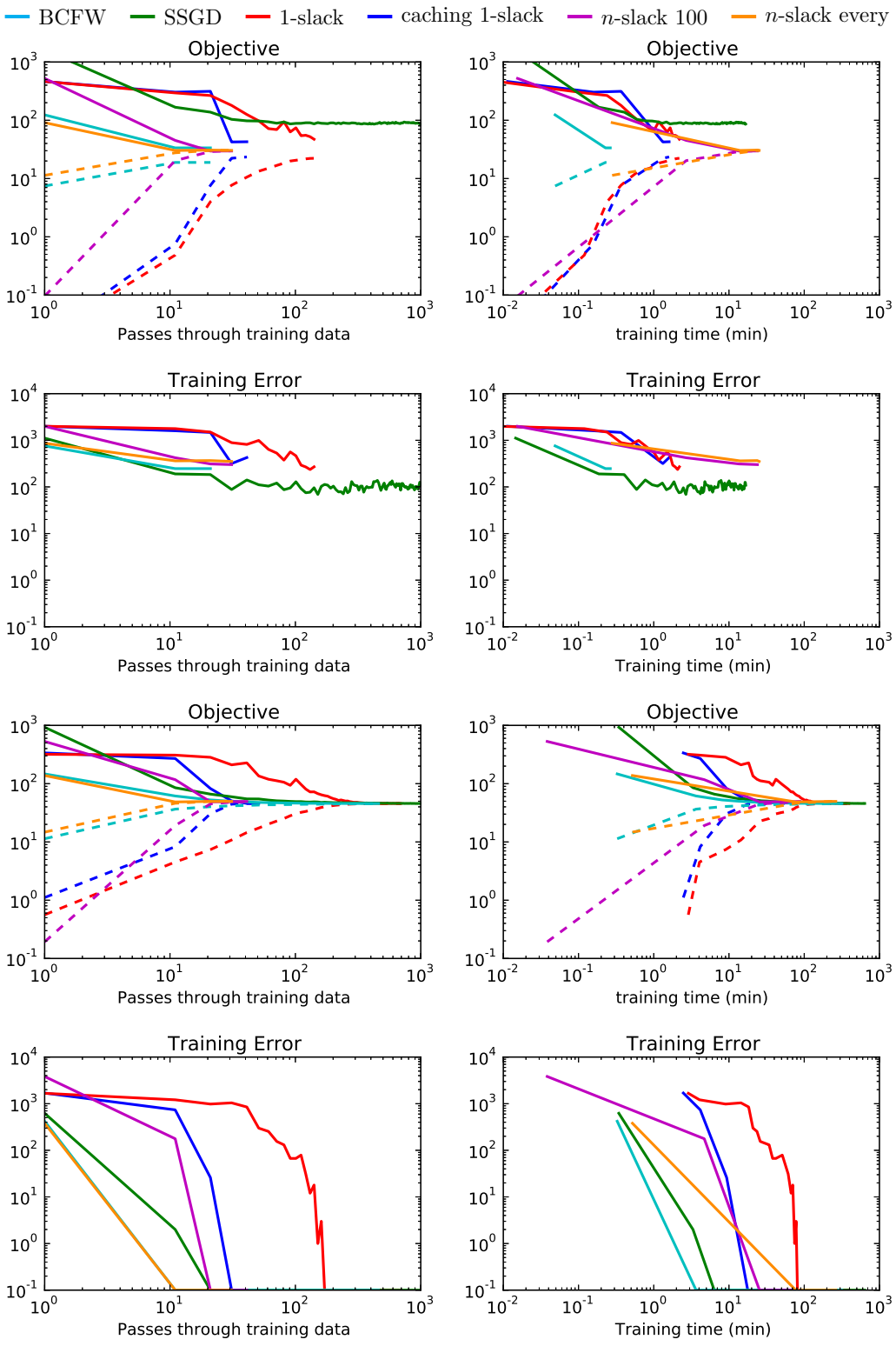

Figure 5.9: Results on the SNAKES dataset. The first two rows show results using fusion move inference, the two rows below using AD^3. Dashed lines indicate the dual objective.

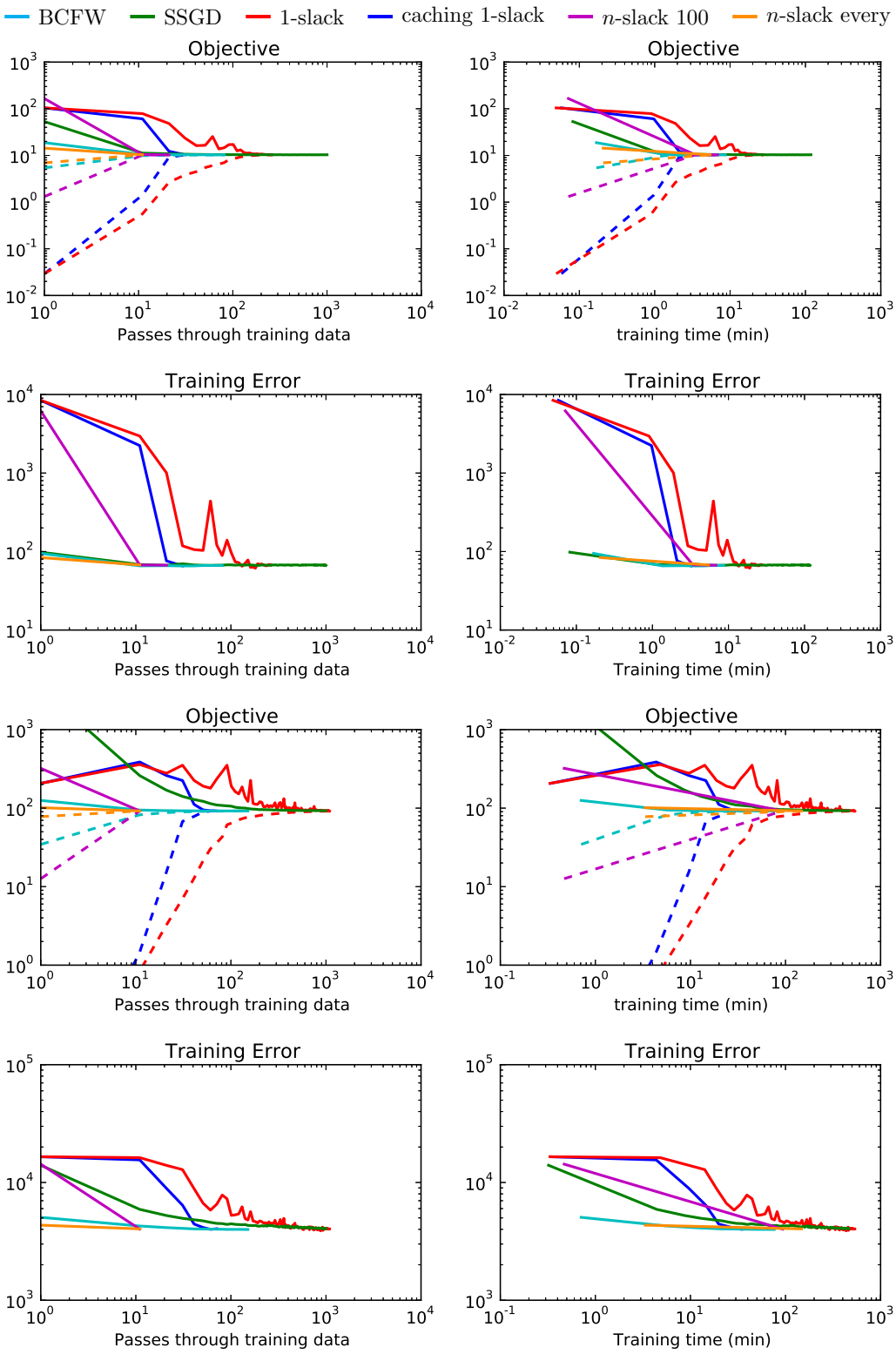

Figure 5.10: Objective and traning set loss on the segmentation task. The first two rows show results for the MSRC-21 dataset, the rows below for the Pascal VOC dataset.

5.3 Summary

From the experiments above, it is clear there is currently *no single best algorithm*. However, there are some clear trends:

- BCFW always converges faster than SSGD, with each iteration having the same time complexity. This indicates that BCFW should be preferred over SSGD, apart from settings with strong memory constraints.
- In nearly all experiments, caching significantly improved performance of the 1-slack cutting plane algorithm.
- Initially BCFW and even SSGD converge faster than the 1-slack cutting plane algorithm, while the 1-slack cutting plane algorithm was faster in achieving high precision solutions and guaranteeing low duality gap.
- The n-slack cutting plane algorithm was often fastest to converge in terms of passes over the training set, but slow in terms of runtime. For problems with very slow inference, the n-slack algorithm might be a better choice than the others—if the interval between re-solving the QP is chosen appropriately.

There are however some caveats to our analysis, in particular with respect to the cutting plane solvers. The performance of the cutting plane algorithms depends not only on the performance of the QP solver used (in our case quadratic cone programming), but also on heuristics for pruning variables from the QP, caching constraints (for the 1-slack algorithm) and deciding how often to solve the QP (for the n-slack algorithm). Better heuristics could have a beneficial effect on these algorithms. On the other hand, it is remarkable how competitive the BCFW algorithm is, without the need for any such implementation tricks. This makes the BCFW algorithm very attractive for the practitioner who does not want to spend much time tuning the implementation.

We do not present results for SSGD with averaging [Lacoste-Julien et al., 2012] here, but found it to produce results similar to plain SSGD in tentative experiments. In particular, it never outperformed BCFW.

With respect to approximate learning, we found that the inference algorithm has a stronger impact on the performance than the learning algorithm. While the

SSGD and BCFW algorithms have somewhat lower training error on the SNAKES dataset than the cutting-plane algorithm when using fusion-move inference, they, too, fail at solving the task. Using the linear programming relaxation, on the other hand, all algorithms are able to solve the task nearly perfectly. For the segmentation datasets, it is harder to judge the outcome, though there seems to be little difference between the learning algorithms in terms of the final result. In the next chapter, we will demonstrate how we can learn exactly, even with the complex models used for semantic segmentation.

6 Learning Loopy CRF Exactly

6.1 Introduction

As discussed in section 4.3, many computer vision algorithms employ conditional random field models on pixel or superpixel graphs. These graphs, by nature, contain loops, making exact prediction and loss-augmented prediction in general intractable. As loss-augmented prediction is a central step in the learning algorithms discussed in Chapter 4, approximate inference leads to complications in learning.

In this chapter, we investigate the necessity and consequences of these approximations in the 1-slack cutting-plane learning algorithm in the context of semantic image segmentation. We show that despite inference being deemed intractable in loopy superpixel models, we are able to learn a pairwise conditional random field model using a structured support vector machine exactly for this task. We evaluate our approach on the popular MSRC-21 and PASCAL VOC datasets. Our approach improves upon the state-of-the-art on the MSRC-21 dataset, and is competitive with comparable approaches on the PASCAL VOC dataset. The contribution of this chapter are:

- We analyze the simultaneous use of multiple approximate inference methods for learning SSVMs using the cutting plane method, relating approximate learning to the exact optimum.
- We introduce an efficient caching scheme to accelerate cutting plane training.
- We demonstrate that using a combination of under-generating and exact inference methods, we can learn an SSVM exactly in a practical application, even in the presence of loopy graphs.
- We show that even using a strong approximate inference procedure can improve upon the state-of-the-art on the MSRC-21 dataset.

While empirically exact learning yields results comparable to those using approximate inference alone, certification of optimality allows treating learning as a black-box, enabling the researcher to focus attention on designing the model for the application at hand. It also makes research more reproducible, as the particular optimization methods that are used become less relevant to the result.

6.2 Related Work

Recently, there has been an increase in research in learning structured prediction models where standard exact inference techniques are not applicable, in particular in the computer vision community. The influence of approximate inference on structural support vector machine learning was first analyzed by Finley and Joachims [2008]. Finley and Joachims [2008] showed convergence results for under-generating and over-generating inference procedures, meaning methods that find suboptimal, but feasible solutions, and optimal solutions from a larger (infeasible) set, respectively. Finley and Joachims [2008] demonstrated that over-generating approaches—in particular linear programming (LP)—perform best on the considered model. They also show that learning parameters with the LP relaxation minimizes a bound on the empirical risk when extending the target domain to the relaxed solutions. We argue that extending the target domain in this way is unnatural for many applications, and aim at optimizing the non-relaxed objective directly, minimizing the original empirical risk. This is an important difference, as relaxed solutions are usually not acceptable in practice.

As using LP relaxations was considered too costly for typical computer vision approaches, later work employed graph-cuts [Szummer et al., 2008] or Loopy Belief Propagation (LBP) [Lucchi et al., 2011]. These works use a single inference algorithm for the whole learning process, and can not provide any bounds on the true objective or the empirical risk. In contrast, in this chapter we show how to combine different inference methods that are more appropriate for different stages of learning.

Recently, Meshi et al. [2010], Hazan and Urtasun [2010] and Komodakis [2011] introduced formulations for joint inference and learning using duality. In particular, Hazan and Urtasun [2010] demonstrated the performance of their model on an image denoising task, where it is possible to learn a large number of

parameters efficiently. While these approaches show great promise, in particular for pixel-level or large-scale problems, they perform approximate inference and learning, and do not relate their results back to the original SSVM objective they approximate. It is unclear how they compare to standard structured prediction approaches in real-world applications.

6.3 Learning SSVMs with Approximate Inference

In this chapter, we use the 1-slack cutting plane algorithm, as we found this algorithm to be most suitable for out approach. Recall the optimization problem that is solved by the 1-slack cutting plane algorithm:

$$\begin{aligned} &\min_{\theta,\xi} \frac{1}{2}||\theta||^2 + C\xi \\ &\text{s.t. } \forall \hat{\mathbf{y}} = (\hat{y}^1, \ldots, \hat{y}^n) \in \mathcal{Y}^n : \\ &\qquad \theta^T \sum_{i=1}^{n} [\Phi(x^i, y^i) - \Phi(x^i, \hat{y}^i)] \geq \sum_{i=1}^{n} \Delta(y^i, \hat{y}^i) - \xi \end{aligned} \tag{6.1}$$

For reference, the algorithm described in Section 4.2.3, Algorithm 3 is reproduced as Algorithm 5, with the additional input of an inference algorithm $\hat{I}$, which is called in line 5. We investigate algorithms $\hat{I}$ (often called separation oracles in the context) that do not yield the exact maximum here. There are two groups of inference procedures, as identified by Finley and Joachims [2008]: under-generating and over-generating approaches. An under-generating approach satisfies $\hat{I}(x^i, y^i, \theta) \in \mathcal{Y}$, but does not guarantee maximality in line 5 of Algorithm 5. An over-generating approach on the other hand, does solve the loss-augmented prediction in line 5 exactly, but for a larger set $\hat{\mathcal{Y}} \supset \mathcal{Y}$, meaning that possibly $\hat{I}(x^i, y^i, \theta) \notin \mathcal{Y}$.

6.3.1 Bounding the Objective

Even using approximate inference procedures, several statements about the original exact objective (Equation 6.1) can be obtained.

Algorithm 5 1-Slack Cutting Plane Training of Structural SVMs

Input: training samples $\{(x^i, y^i), \ldots, (x^i, y^i)\}$, regularization parameter C, stopping tolerance ϵ, inference oracle $\hat{I}$.
Output: parameters θ, slack ξ

1: $\mathcal{W} \leftarrow \emptyset$
2: **repeat**
3:
$$(\theta, \xi) \leftarrow \underset{\theta,\xi}{\operatorname{argmin}} \frac{||\theta||^2}{2} + C\xi$$
$$\text{s.t. } \forall \hat{\mathbf{y}} = (\hat{y}^1, \ldots, \hat{y}^k) \in \mathcal{W} :$$
$$\theta^T \sum_{i=1}^{k} [\Phi(x^i, y^i) - \Phi(x^i, \hat{y}^i)] \geq \sum_{i=1}^{k} \Delta(y^i, \hat{y}^i) - \xi$$
4: **for** i=1, . . . , k **do**
5: $$\hat{y}^i \leftarrow \hat{I}(x^i, y^i, \theta) \approx \underset{\hat{y} \in \mathcal{Y}}{\operatorname{argmax}} \sum_{i=1}^{k} \Delta(y^i, \hat{y}) - \theta^T \sum_{i=1}^{k} [\Phi(x^i, y^i) - \Phi(x^i, \hat{y})]$$
6: $\mathcal{W} \leftarrow \mathcal{W} \cup \{(\hat{y}^i, \ldots, \hat{y}^i)\}$
7: $$\xi' \leftarrow \sum_{i=1}^{k} \Delta(y^i, \hat{y}^i) - \theta^T \sum_{i=1}^{k} [\Phi(x^i, y^i) - \Phi(x^i, \hat{y}^i)]$$
8: **until** $\xi' - \xi < \epsilon$

Let $o_{\mathcal{W}}(\theta)$ denote the objective in Equation 6.1 with given parameters θ restricted to a working set $\mathcal{W}$, as computed in line 3 of Algorithm 5 and let

$$o^{\hat{I}}(\theta) = C\xi' + \frac{||\theta||^2}{2}$$

when using inference algorithm $\hat{I}$, that is $o^{\hat{I}}(\theta)$ is the approximation of the primal objective given by $\hat{I}$. To simplify exposition, we drop the dependency on θ.

Depending on the properties of the inference procedure $\hat{I}$ used, it is easy to see:

1. If all constraints $\hat{y}$ in $\mathcal{W}$ are feasible, that is generated by an under-generating or exact inference mechanism, then $o_{\mathcal{W}}$ is an lower bound on the true optimum $o(\theta^*)$.

2. If $\hat{I}$ is an over-generating or exact algorithm, $o^{\hat{I}}$ is an upper bound on $o(\theta^*)$.

We can also use these observations to judge the suboptimality of a given parameter θ, that is see how far the current objective is from the true optimum.

Learning with any under-generating approach, we can use 1. to maintain a lower bound on the objective. At any point during learning, in particular if no more constraints can be found, we can then use 2., to also find an upper bound. This way, we can empirically bound the estimation error, using only approximate inference. We now describe how we can further use 1. to both speed up and improve learning.

6.4 Efficient Exact Cutting Plane Training of SSVMs

6.4.1 Combining Inference Procedures

The cutting plane method described in Section 4.2.3 relies only on some separation oracle $\hat{I}$ that produces violated constraints when performing loss-augmented prediction.

Using any under-generating oracle $\hat{I}$, learning can proceed as long as a constraint is found that is violated by more than the stopping tolerance ϵ. Which constraint is used next has an impact on the speed of convergence, but not on correctness. Therefore, as long as an under-generating method does generate constraints, optimization makes progress on the objective.

Instead of choosing a single oracle, we propose to use a succession of algorithms, moving from fast methods to more exact methods as training proceeds. This strategy not only accelerates training, it even makes it possible to train with exact inference methods, which is infeasible otherwise.

In particular, we employ three strategies for producing constraints, moving from one to the next if no more constraints can be found:

1. Produce a constraint using previous, cached inference results.
2. Use a fast under-generating algorithm.
3. Use a strong but slow algorithm that can certify optimality.

While using more different oracles is certainly possible, we found that using just these three methods performed very well in practice. This combination allows us to make fast progress initially and guarantee optimality in the end. Our strategy is visualized in Figure 6.1. Notably, it is not necessary for an algorithm used as

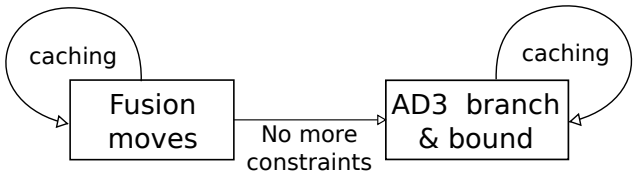

Figure 6.1: Illustration of the choice of inference algorithm. During the beginning of learning, fusion move inference together with caching is used. If no more constraint can be found, the fusion move algorithm is replaced by AD^3 with branch and bound.

the third strategy to always produce exact results. For guaranteeing optimality of the model, it is sufficient that we obtain a certificate of optimality when learning stops.

6.4.2 Dynamic Constraint Selection

Combining inference algorithm as described in Section 6.4.1 accelerates calls to the separation oracle by using faster, less accurate methods. On the down-side, this can lead to the inclusion of many constraints that make little progress in the overall optimization, resulting in much more iterations of the cutting plane algorithm. We found this particularly problematic with constraints produced by the cached oracle.

We can overcome this problem by defining a more elaborate schedule to switch between oracles, instead of switching only if no violated constraint can be found any more. Our proposed schedule is based on the intuition that we only trust a separation oracle as long as the current primal objective did not move far from the primal objective as computed with the stronger inference procedure.

In the following, we use the notation of Section 6.3.1 and indicate the choices of oracle $\hat{I}$ with Q for a chosen inference algorithm and C for using cached constraints.

To determine whether to produce inference results from the cache or to run the inference algorithm, we solve the QP once with a constraint from the cache. If the resulting o^C verifies

$$o^C - o^Q < \frac{1}{2}(o^Q - o_{\mathcal{W}}) \tag{6.2}$$

we continue using the caching oracle. Otherwise we run the inference algorithm

	Average	Global
Unary terms only	77.7	83.2
Pairwise model (move making)	79.6	84.6
Pairwise model (exact)	79.0	84.3
Ladicky et al. [2009]	75.8	85.0
Gonfaus et al. [2010]	77	75
Lucchi et al. [2013]	78.9	83.7

Table 6.1: Accuracies on the MSRC-21 Dataset. We compare a baseline model, our exact and approximately learned models and state-of-the-art approaches.

again. For testing Equation 6.2, the last known value of o^Q is used, as recomputing it would defy the purpose of the cache. It is easy to see that our heuristic runs inference only $O(\log(o^Q - o_{\mathcal{W}}))$ times more often than the strategy of Joachims et al. [2009] in the worst case.

6.5 Experiments

6.5.1 Inference Algorithms

As a fast under-generating inference algorithm, we used fusion moves [Rother et al., 2007, Lempitsky et al., 2010]. Fusion moves are a local search procedure, using moves that are generated using an auxiliar binary problem, which is solved using QPBO [Rother et al., 2007].

For inference with optimality certificate, we use the recently developed Alternating Direction Dual Decomposition (AD^3) method of Martins et al. [2011]. AD^3 produces a solution to the linear programming relaxation, which we use as the basis for branch-and-bound.

6.5.2 Semantic Image Segmentation

Our main application is of course semantic segmentation and object class segmentation. We evaluate the proposed learning approach on PASCAL VOC 2013 and MSRC-21, with model and features discussed in Section 4.3. We use the same model and pairwise features for the two datasets. Each image is represented

as a neighborhood graph of approximately 100 superpixels, extracted using the SLIC [Achanta et al., 2012] algorithm. Pairwise potentials are founded on two image-based features: color contrast between superpixels, and relative location (coded as angle), in addition to a bias term.

We set the stopping criterion $\epsilon = 10^{-4}$, though using only the under-generating method, training always stopped prematurely as no violated constraints could be found any more.

6.5.3 Caching

First, we compare our caching scheme, as described in Section 6.4.1, with the scheme of Joachims et al. [2009], which produces constrains from the cache as long as possible, and with not using caching of constraints at all. For this experiment, we only use the under-generating move-making inference on the MSRC-21 dataset. Times until convergence are 397s for our heuristic, 1453s for the heuristic of Joachims et al. [2009], and 2661s for using no cache, with all strategies reaching essentially the same objective.

Figure 6.2 shows a visual comparison that highlights the differences between the methods. Note that neither o^Q nor o^C provide valid upper bounds on the objective, which is particularly visible for o^C using the method of Joachims et al. [2009]. Using no cache leads to a relatively smooth, but slow convergence, as inference is run often. Using the method of Joachims et al. [2009], each run of the separation oracle is followed by quick progress of the dual objective $o_{\mathcal{W}}$, which flattens out quickly. Much time is then spent adding constraints that do not improve the dual solution. Our heuristic instead probes the cache, and only proceeds using cached constraints if the resulting o^C is not too far from o^Q.

MSRC-21 Dataset

For the MSRC-21 Dataset, we use unary potentials based on bag-of-words of SIFT features and color features and TextonBoost as described in Section 4.3.2. We used TextonBoost only on one scale for thi experiment, leading to $42 = 2 \cdot 21$ unary features for each node. The resulting model has around 100 output variables per image, each taking one of 21 labels. The model is trained on 335 images from the standard training and validation split.

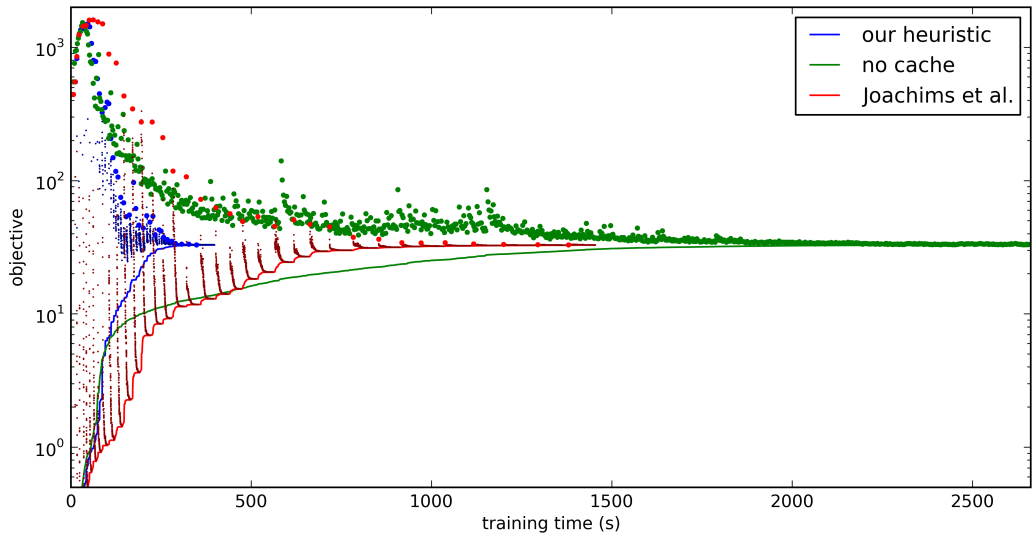

Figure 6.2: Training time comparison using different caching heuristics. Large dots correspond to o^Q, small dots correspond to o^C, and the line shows $o_{\mathcal{W}}$. See the text for details.

	Jaccard
Unary terms only	27.5
Pairwise model (move making)	30.2
Pairwise model (exact)	30.4
Dann et al. [2012]	27.4
Krähenbühl and Koltun [2012]	30.2
Krähenbühl and Koltun [2013]	30.8

Table 6.2: Accuracies on the PASCAL VOC Dataset. We compare our approach against approaches using the same unary potentials.

	Move-making	Exact
Dual Objective $o_{\mathcal{W}}$	65.10	67.66
Estimated Objective $o^{\hat{I}}$	67.62	67.66
True Primal Objective o^{E}	69.92	67.66

Table 6.3: Objective function values on the MSRC-21 Dataset

Pascal VOC 2010

For the Pascal VOC 2010 dataset, we follow the procedure of Krähenbühl and Koltun [2012] in using the official "validation" set as our evaluation set, and splitting the training set again. We use the unary potentials provided by the same work, and compare only against methods using the same setup and potentials, Krähenbühl and Koltun [2013] and Dann et al. [2012]. Note that state-of-the-art approaches, some not build on the CRF framework, obtain a Jaccard Index (also call VOC score) around 40% , notably Xia et al. [2012], who evaluate on the Pascal VOC 2010 "test" set.

Results

We compare classification results using different inference schemes with results from the literature. As a sanity check, we also provide results without pairwise interactions.

Results on the MSRC-21 dataset are shown in Table 6.1. We find that our model is improves upon state-of-the-art approaches. In particular, our results improve upon to those of Lucchi et al. [2013], who use a stochastic subgradient method with working sets. Their best model takes 583s for training, while training our model exactly takes 1814s. We find it remarkable that it is possible to guarantee optimality in time of the same order of magnitude that a stochastic subgradient procedure with approximate inference takes. Using exact learning and inference does not increase accuracy on this dataset. Learning the structured prediction model using move-making inference alone takes 4 minutes, while guaranteeing optimality up to $\epsilon = 10^{-4}$ takes only 18 minutes.

Results on the Pascal VOC dataset are shown in Table 6.2. We compare against several approaches using the same unary potentials. For completeness, we also

	Move-making	Exact
Dual Objective $o_{\mathcal{W}}$	92.06	92.24
Estimated Objective $o^{\hat{I}}$	92.07	92.24
True Primal Objective o^{E}	92.35	92.24

Table 6.4: Objective function values on the PASCAL VOC Dataset.

list state-of-the-art approaches not based on CRF models. Notably, out model matches or exceeds the performance of the much more involved approaches of Krähenbühl and Koltun [2012] and Dann et al. [2012] which use the same unary potentials. Using exact learning and inference slightly increased performance on this dataset. Learning took 25 minutes using move-making alone and 100 minutes to guarantee optimality up to $\epsilon = 10^{-4}$. A visual comparison of selected cases is shown in Figure 6.3.

The objective function values using only the under-generating move-making and the exact inference are detailed in Table 6.3 and Table 6.4. We see that a significant gap between the cutting plane objective and the primal objective remains when using only under-generating inference. Additionally, the estimated primal objective $o^{\hat{I}}$ using under-generating inference is too optimistic, as can be expected. This underlines the fact that under-generating approaches can not be used to upper-bound the primal objective or compute meaningful duality gaps.

6.5.4 Implementation Details

We implemented the described procedure in PYSTRUCT. We used the SLIC implementation provided by Achanta et al. [2012] to extract superpixels and the SIFT implementation in the VLFEAT package [Vedaldi and Fulkerson, 2008]. For clustering visual words, piecewise training of unary potentials and the approximation to the χ^2-kernel, we made use of the SCIKIT-LEARN machine learning package [Pedregosa et al., 2011]. The we implement fusion moves with the help of the QPBO-I method provided by Rother et al. [2007]. We use the excellent implementation of AD^3 provided by Martins et al. [2011].

Thanks to using these high-quality implementations, running the whole pipeline for the pairwise model takes less than an hour on a 12 core CPU. Solving the QP is done in a single thread, while inference is parallelized over all cores.

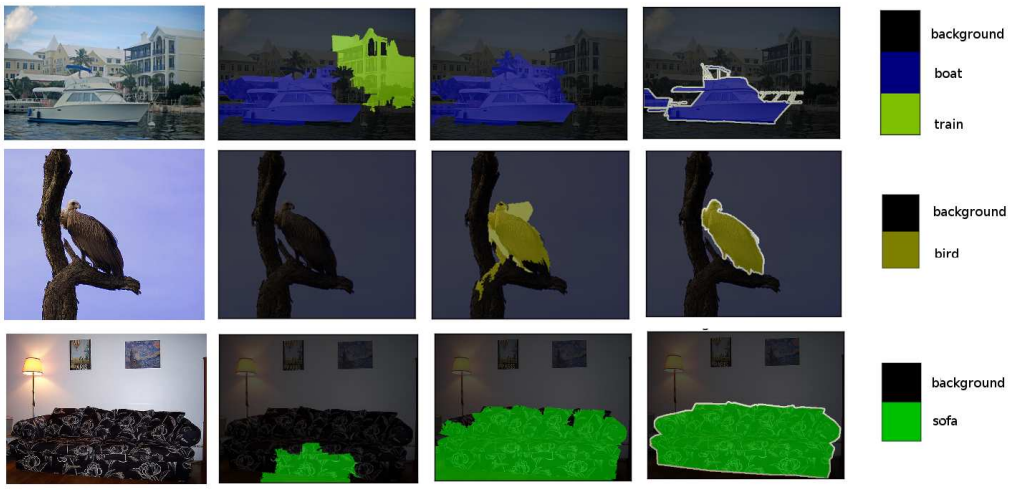

Figure 6.3: Visual comparison of the result of exact and approximate learning on selected images from the test set. From left to right: the input image, prediction using approximate learning, prediction using exact learning, and ground truth.

6.6 Summary

In this chapter we demonstrated that it is possible to learn state-of-the-art conditional random field models exactly using structural support vector machines, despite the model containing many loops. The key to efficient learning is the combination of different inference mechanisms and a novel caching scheme for the 1-slack cutting plane method, in combination with state-of-the-art inference methods.

We show that guaranteeing exact results is feasible in a practical setting, and hope that this result provides a new perspective onto learning loopy models for computer vision applications. Even though exact learning does not necessarily lead to a large improvement in accuracy, it frees the practitioner from worrying about optimization and approximation issues, leaving more room for improving the model, instead of the optimization. We do not expect learning of pixel-level models, which typically have tens or hundreds of thousands of variables, to be efficient using exact inference. However we believe our results will carry over to other super-pixel based approaches. Using other over-generating techniques, such as duality-based message passing algorithms, it might be possible to obtain meaningful bounds on the true objective, even in the pixel-level domain.

7 Learning Depth-Sensitive Conditional Random Fields

For robots to perform varied tasks in unstructured environments, understanding their surroundings is essential. In this chapter, we look at the semantic annotation of maps as a dense labeling of RGB-D images into semantic classes. We formulate the problem as learning a CRF over a superpixel segmentation of the RGB-D image, producing a labeling that takes 3D layout into account. Dense labeling of objects and structure classes allows for a detailed reasoning about the scene.

We thereby extend the success of learned CRF models for semantic segmentation in RGB images as considered in Chapter 6 to the domain of 3D scenes. Our emphasis lies on exploiting the additional depth and 3D information in all processing steps, while relying on *learning* to create a model that is adjusted to the properties of the sensor input and environment.

Our approach starts with a random forest, providing a noisy local estimate of semantic classes based on color and depth information. These estimates are grouped together using a superpixel approach, for which we extend previous superpixel algorithms from the RGB to the RGB-D domain. We then build a geometric model of the scene, based on the neighborhood graph of superpixels. We use this graph not only to capture spatial relations in the 2D plane of the image, but also to model object distances and surface angles in 3D, using a point cloud generated from the RGB-D image. The process is depicted in Figure 7.1.

We assess the accuracy of our model on the challenging NYU Segmentation dataset V2 [Silberman et al., 2012], where our model outperforms previous approaches. Our analysis shows that while our random forest model already has competitive performance, the superpixel-based grouping and in particular the loss-based learning are integral ingredients of the success of our method.

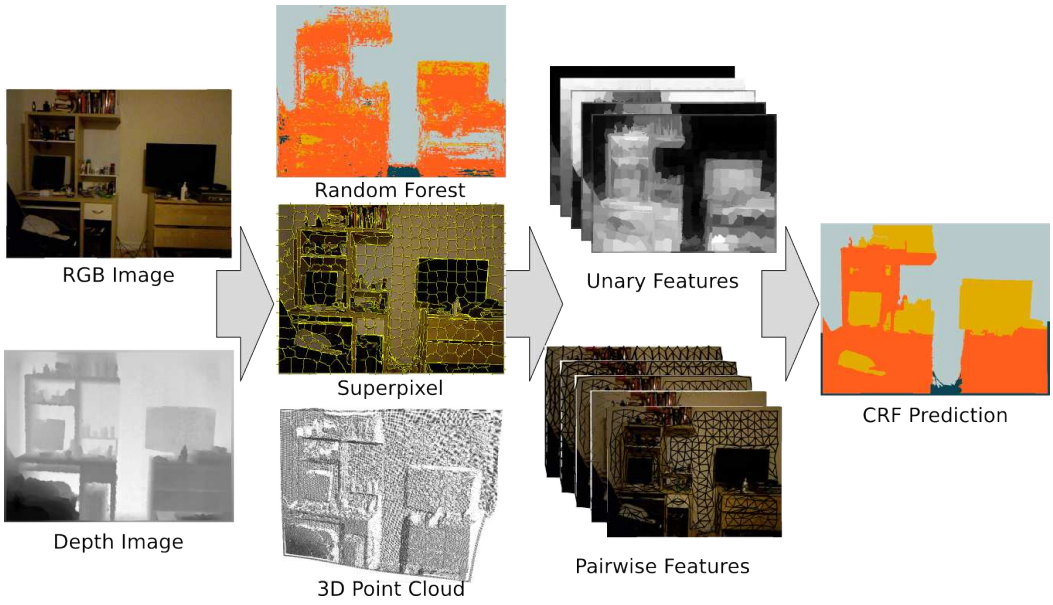

Figure 7.1: Overview of the proposed semantic segmentation method.

7.1 Related Work

The task of dense semantic annotation of 3D maps has seen an increased interest in recent years. Early work includes Nüchter and Hertzberg [2008], who combined 6D SLAM, surface annotation, and object recognition to build semantically annotated maps. They demonstrate their approach on a mobile robot in an indoor environment. More recently Sengupta et al. [2012] introduced a dataset of semantically annotated street-scenes on a closed track, captured as pairs of stereo images. They approach the task by jointly reasoning about 3D layout and semantics of the scenes and produce a dense labeling on image level. Sengupta et al. [2013] extended the approach to produce a volumetric reconstruction of the scene, together with a dense semantic labeling of the volumetric representation. Their approach to image segmentation builds on the hierarchical CRF approach of Ladicky et al. [2009], which is similar in spirit to our approach, but used Potts potentials together with cross-validation to adjust parameters.

Recent approaches for indoor semantic annotation of maps mostly focused on RGB-D images, which are now easy to obtain using structured light sensors. Stückler et al. [2012], for example, used a Random Forest model to obtain a dense semantic labeling of images and integrated predictions over multiple views in 3D.

They evaluated their approach on table-top and simple indoors scenes. Silberman and Fergus [2011] introduced the NYU Depth Dataset V1 indoor dataset, which consisted of a large variety of densely annotated indoor scenes, captured as RGB-D images. Their work also introduced a baseline method for semantic segmentation of RGB-D image, which is based on a CRF over superpixels, with unary potentials given by interest point descriptors. While pairwise potentials for the CRF were carefully designed for the dataset, potentials were either directly set by hand or estimated using empirical frequencies. This is in contrast to our approach which applies structured prediction techniques to learn potentials automatically to optimize predictive performance. Ren et al. [2012] evaluated the design of features for semantic labeling of RGB-D data, and use a hierarchical segmentation to provide context. While they also define a CRF on superpixels, their model is again not learned, but a weighted Potts model, using only a probability of boundary map, and not taking spatial layout into account at all. Silberman et al. [2012] extended the NYU Depth Dataset V1 to the NYU Depth Dataset V2 that we are also using here. Their focus is on inferring support relations in indoor scenes, such as objects resting on tables or shelfs, which in turn rest on the floor. Their approach is based on robust estimation of 3D plane hypotheses, which are then jointly optimized with support relations and structure classes. Silberman et al. [2012] used a complex pipeline, employing significant domain knowledge. In our approach, on the other hand, we try to learn all relevant domain specific features directly from the data, which allows us to out-perform the work of Silberman et al. [2012] with respect to structure class segmentation.

Couprie et al. [2013] approached the task of semantic segmentation of structure classes in RGB-D using the paradigm of convolutional neural networks, extending previous work of Farabet et al. [2013] and Schulz and Behnke [2012]. Similar to our approach, Couprie et al. [2013] combined the output of a pixel-based, low-level learning algorithm with an independent unsupervised segmentation step. In contrast to their work, we improve our results by not only averaging predictions within superpixels, but also explicitly learning interactions between neighboring superpixels, favoring a consistent interpretation of the whole image.

Stückler et al. [2013] extended the approach of Stückler et al. [2012] to a real-time system for online learning and prediction of semantic classes. Their method use a GPU implementation of random forests, and integrate 3D scene information in an online fashion. They evaluated their approach on the dataset

of Silberman et al. [2012] with promising results. We use the implementation of random forests provided by Stückler et al. [2013], but instead of integrating predictions over time, we focus on exploiting the structure within a single frame.

While many of the works mentioned in this chapter make use of a CRF approach, we are not aware of any prior work on semantic annotation of 3D maps that fully learns their potentials.

7.2 Learning Depth-Sensitive Conditional Random Fields

We start with the CRF approach described in section 4.3, where nodes represent a labeling of superpixels. Recall the general form of the energy

$$g(x, y) = \sum_{v \in V} \psi_v(x, y_v) + \sum_{(v,w) \in E} \psi_{v,w}(x, y_v, y_w). \tag{7.1}$$

Here V enumerates the superpixels, and $E \subset V \times V$ is a set of edges, encoding adjacence between superpixels.

In contrast to competing approaches, the learning the parameters of the CRF using an SSVM allows for a principle, maximum-margin based, loss-sensitive training of CRFs. Learning the potentials yields much more complex interactions than the simple Potts potentials that are often used in the literature.

The features used to learn the potentials are described in detail below. We use the 1-slack formulation of the structural SVM [Joachims et al., 2009] as implemented in PyStruct and described in Chapter 4. The combination of fusion moves and AD^3 described in Chapter 6 allows us to learn the SSVM to optimality *exactly*.

7.2.1 Low Level Segmentation

We take a super-pixel based approach to semantic segmentation. Our superpixel generation is based on the SLIC algorithm [Achanta et al., 2012] described in Section 4.3. We extend the standard SLIC algorithm, which works on the Lab space, to also include depth information. The resulting algorithm is a localized k-means in Lab-D-XY space. Our implementation is publicly available through

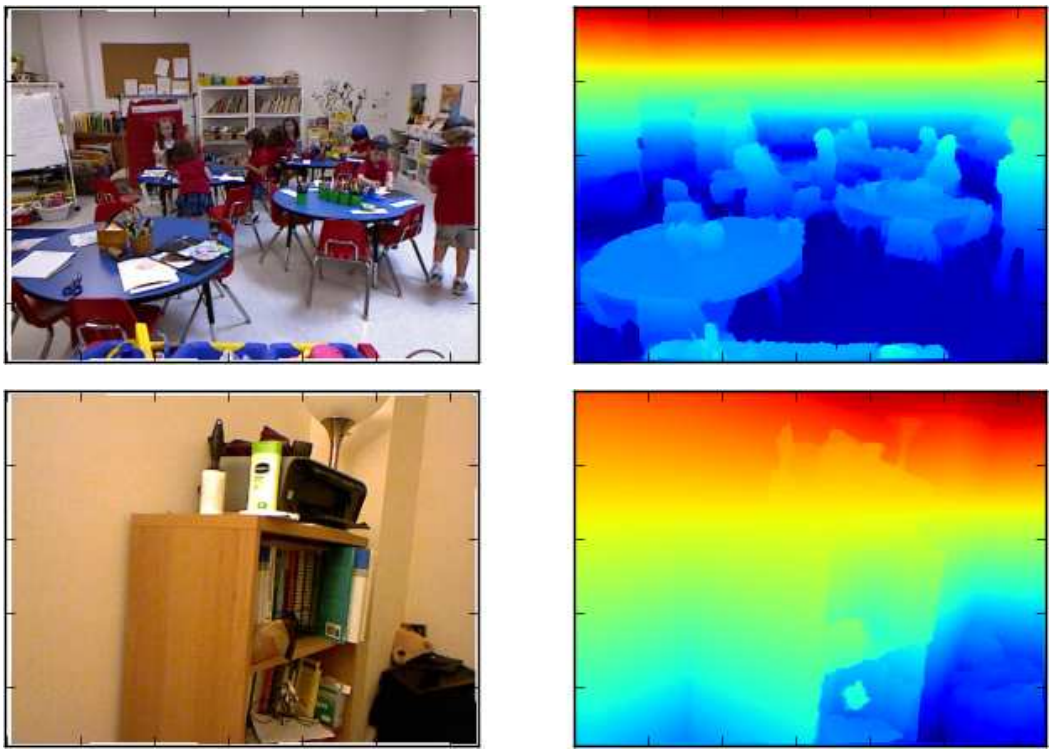

Figure 7.2: Visualization of the height computed using the method described in Section 7.2.2. Input images are shown on the left (depth not shown), the computed height is depicted on the right. The top row exemplifies a typical scene, while the bottom row shows a scene without horizontal surfaces, where our method fails.

the scikit-image library*. Similar to Silberman et al. [2012], we found little visual improvement over the RGB segmentation when using additional depth information. On the other hand, estimation of per-superpixel features based on the 3D point cloud was more robust when including depth information into the superpixel procedure. The resulting superpixels are compact in the 2D image. As the density of the corresponding point cloud is dependent on the depth, we did not succeed in creating superpixels that are compact in 3D while maintaining a meaningful minimum size.

*http://scikit-image.org

7.2.2 Unary Image Features

Our method builds on the probability output of a random forest, trained for pixel-wise classification of the structure classes. We use the GPU implementation provided by Stückler et al. [2013][†]. The input for training are the full RGB-D images, transformed to Lab color space. Each tree in the forest uses training pixels only from a subset of training images. For each training image, an equal number of pixels for each occurring class is sampled. Split features are given by difference of regions on color or depth channels. Region size and offsets are normalized using the depth at the target pixel. We accumulate the probabilistic output for all pixels within a superpixel, and use the resulting distribution as a feature for the unary node potentials in our CRF model. We augment these prediction with another feature, based on the height of a superpixel in 3D. This is a very informative feature, in particular to determine the floor. To compute the height of a (super) pixel, we first find the "up" direction. We use a very simple approach that we found effective: we cluster normal directions of all pixels into 10 clusters using k-means, and use the one that is most parallel to the Y direction, which roughly corresponds to height in the dataset. We then project the 3D point cloud given by the depth along this direction, and normalize the result between 0 and 1. This procedure works robustly if there is some horizontal surface in the image, such as the ground or a table. A few scenes contain only walls and furniture, and the approach fails for these. Figure 7.2 illustrates a typical case and one of the much rarer failure cases. While we could use a more elaborate scheme, such as the one from Silberman et al. [2012], we suspect that the feature is of little use in scenes without horizontal surfaces.

7.2.3 Pairwise Depth-Sensitive Features

There are five different features used to build pairwise potentials in our model:

Constant A constant feature allows to model general neighborhood relations.

[†] `https://github.com/deeplearningais/curfil`

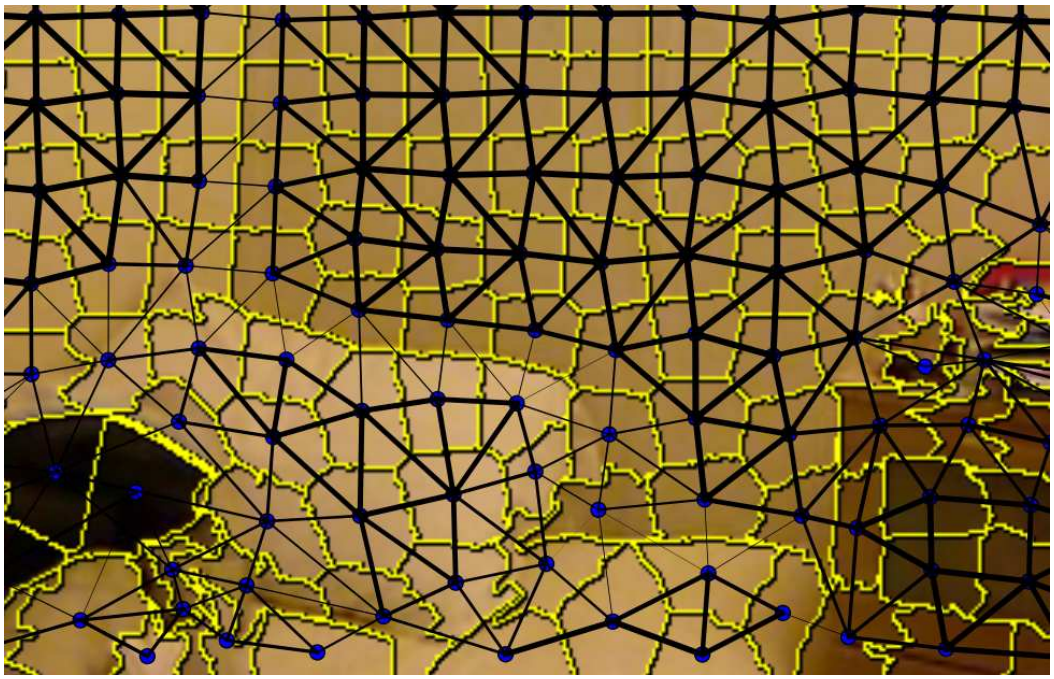

Figure 7.3: Visualization of one of the pairwise features, the similarity between superpixel normals. The image shows the zoom-in of a bedroom scene, together with the superpixel over-segmentation. Lines connect adjacent superpixels, and line-strength gives the magnitude of the orientation similarity.

Color Contrast We employ a non-linear color contrast, as is common in the computer vision literature, between the superpixel mean colors c_i and c_j:

$$\exp\left(-\gamma\|c_i - c_j\|^2\right).$$

Vertical Alignment We model the directed angle between superpixel centers in the 2D image plane. This allows the model to learn that "structure" is above "floor", but not the other way around.

Depth Difference We include the signed depth difference between superpixels, which allows the model to detect depth discontinuities that are not represented in the 2D neighborhood graph of the superpixels.

Normal Orientations Differences in normal vector orientation are a strong clue on whether two superpixels belong to the same surface, and therefore

	ground	structure	furniture	prop	class avg.	pixel avg.
RF	90.8	81.6	67.9	19.9	65.0	68.3
RF + SP	92.5	83.3	**73.8**	13.9	65.7	70.1
RF + SP + SVM	94.4	79.1	64.2	**44.0**	70.4	70.3
RF + SP + CRF	**94.9**	78.9	71.1	42.7	**71.9**	**72.3**
Silberman et al. [2012]	68	59	70	42	59.6	58.6
Couprie et al. [2013]	87.3	**86.1**	45.3	35.5	63.5	64.5
Stückler et al. [2013]†	**95.6**	83.0	**75.1**	14.2	67.0	70.9

Table 7.1: Quantitative comparison of the proposed method with the literature. The best value in each column is printed in bold†. The upper part of the table shows contributions by different parts of our pipeline. RF stands for random forest prediction, RF + SP for aggregated random forests prediction within superpixels, RF + SP + SVM for an SVM trained on the unary potentials, and RF + SP + CRF is our proposed pipeline. We optimized our approach for class average accuracy.
† Note that the work of Stückler et al. [2013] is not directly comparable, as they integrated information over multiple frames, and did not measure accuracy for pixels without valid depth measurement.

the same structural class. We compute the 3D orientation of normals using the method of Holz et al. [2011], as implemented in the point cloud library (pcl)‡. All normals within a superpixel are then averaged, to get a single orientation for each superpixel. The feature is computed as the difference of $\frac{\pi}{4}$ and the (undirected) angle between the normals belonging to two adjacent superpixels. The feature is illustrated in Figure 7.3. The change in normal orientation highlights that pillow and wall are distinct objects, even though there is no strong distinction in color or depth.

7.3 Experiments

We evaluate our approach on the public NYU segmentation dataset V2 of indoor scenes. The dataset comes with a detailed annotation of 1449 indoor RGB-D images belonging to a wide variety of indoor scenes, categorized into 26 scene classes. The annotation contains four semantic structural classes: structure, floor, furniture and prop. As in the MSRC-21 and Pascal VOC datasets, there is an additional "void" class, which is used for object boundaries and hard-to-annotate

‡http://pointclouds.org

regions. We follow the literature in excluding these pixels completely from the evaluation. We optimize our model for *average class accuracy* (the mean of the diagonal of the confusion matrix), which puts more emphasis on the harder classes of prop and furniture, which have smaller area than structure and floor.

Our approach is implemented using our PyStruct library introduced in Section 4.4. All hyper-parameters were adjusted using 5-fold cross-validation. The hyper parameters of the random forests were found using the hyperopt framework of Bergstra et al. [2011]. For the CRF model, the only hyper-parameters are related to the superpixel segmentation, and the single hyper-parameter C of the structural SVM formulation. These were adjusted using grid-search. We found 500 superpixels per image to work best, which allow for a maximum possible performance of 95% average class accuracy on the validation set.

We observed that the linear programming relaxation often found an integer solution, without the need for a branch-and-bound procedure. We also found that using fusion moves alone produced inferior results.

Table 7.1 compares different components of our approach with the literature. Note that we *first* designed our final model, using only the validation data. We now report accuracies of simpler models for reference, however these results were not used for model selection. To separate the influence of loss-based training and the spatial reasoning of the CRF, we also train a usual support vector machine (SVM) on the unary potentials for comparison.

The random forest prediction, as reported in Stückler et al. [2013] is already quite competitive. Grouping into superpixels slightly improves performance, by removing high-frequency noise and snapping to object boundaries. Somewhat surprisingly, using a standard unstructured SVM with rescaled loss already advances the mean accuracy decidedly above the previous state-of-the-art. We attribute this mostly to the ability of the SVM to exploit correlation between classes and uncertainty within the superpixels. Additionally, the SVM has access to the "height" feature, that was not included in the random forest. This performance is still improved upon, both in class average and pixel average performance by the learned CRF approach, yielding the best published result so far for both measures. The increase over the standard SVM is 1.5% for class average accuracy and 2.0% for pixel average accuracy.

A visualization of the impact of each processing step can be found in Figure 7.5, which shows prediction results on the test set. The four prediction methods

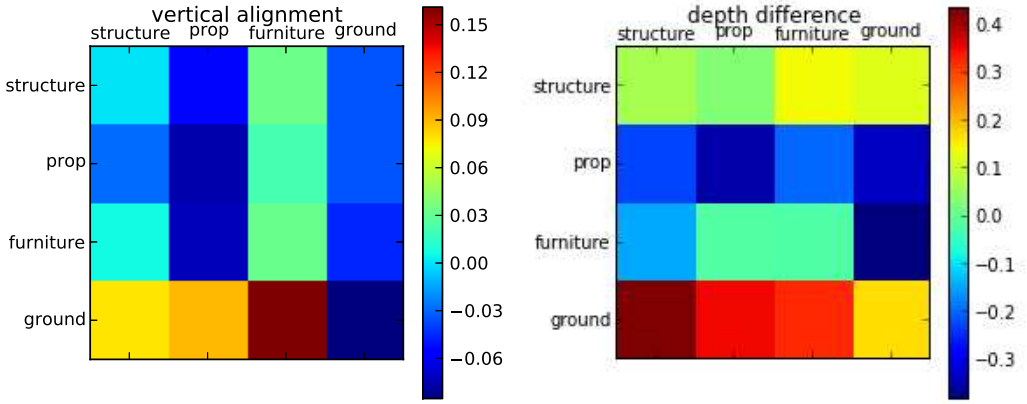

Figure 7.4: Visualization of some of the learned potentials. The right potential is applied to relative depth between superpixels, the second on the feature encoding whether one superpixel is above the other in the image. See section 7.3 for details.

correspond to the rows of Table 7.1. The difference between the SVM and CRF approaches are clearly visible, with the CRF producing results that are very close to the ground truth in several complex scenes.

We found that our approach improves results most for scenes with a clear geometric structure, which is not surprising. We see that evidence from the random forest is often very noisy, and biased away from the "prop" class. While the unstructured SVM can correct somewhat for the class imbalance, it has no way to make larger areas consistent, which the CRF can. On the other hand, performance of the CRF deteriorates slightly on very crowded scenes with a mixture of small furniture and prop objects, as can be seen in the two right-most images. In these scenes, depth information is often noisy, and it is hard to make geometric statements on the superpixel level. As the input from the random forest is also often of low quality for crowded scenes, the CRF has little chance to recover. Figure 7.4 visualizes some learned potential functions. Higher values correspond to favored configurations. One can see that the vertical alignment potential expresses that the floor is much more likely to be below other classes. It also encodes the fact that prop rest on furniture, but not the other way around. The potential of the depth feature encodes, for example, that the ground is usually behind the other classes, while furniture is in front of structures, such as the wall. Interestingly, the potential functions are not anti-symmetric, and forcing them to

be so degrades the results. This suggests that the direction of connecting edges, going from the top left to the bottom right in the image, is also exploited by the potentials.

7.4 Summary and Discussion

We introduce a CRF formulation for semantic segmentation of structure classes in RGB-D images. We base our model on the output of an efficient GPU implementation of random forest, and model spatial neighborhood using a superpixel-based approach. We combine color, depth and 3D orientation features into an energy function that is learned using the SSVM approach. By explicitly modeling 3D relations in a fully learned framework, we improve the state-of-the-art on the NYU dataset V2 for semantic annotation of structure classes.

While our approach allows modeling of spatial relations, these are limited to local interactions. In future work, these interactions could be extended to larger areas using latent variable models or higher order potentials [Ladicky et al., 2009]. Another possible line of future investigation is to combine our approach with a more task-specific one, directly including support plane assumptions into the model, as done by Silberman et al. [2012]. Finally, we could also combine our single-frame approach with the approach of Stückler et al. [2013], which fuses individual views in 3D to exploit temporal coherence.

We did not explicitly address real time application; the random forest implementation that we build upon allows for real-time processing [Stückler et al., 2013]. The SLIC superpixel algorithm can also be implemented on GPU in real-time, as was demonstrated by [Ren and Reid, 2011], and similarly the normal features we use also have real-time capabilities [Holz et al., 2011]. Finally, fusion move inference for our model is very efficient for our model, opening up the possibility to implement our approach entirely in real time.

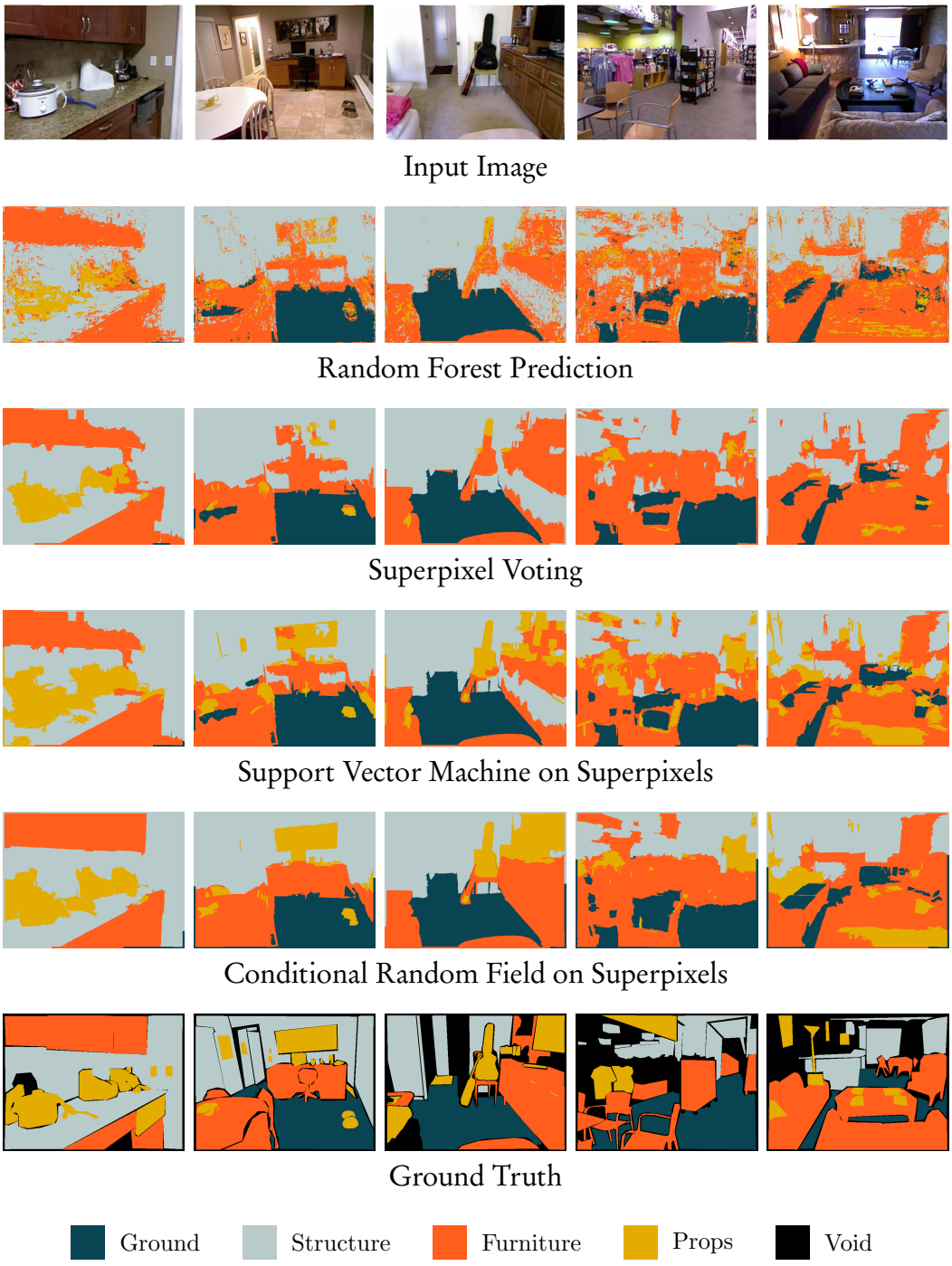

Figure 7.5: Qualitative evaluation of the CRF. The first three images illustrate errors in the original prediction that can be corrected, while the second two images illustrate failure modes. Pixels marked as void are excluded from the evaluation. See Section 7.3 for details.

8 Conclusion

In this thesis, we explored the use of structured prediction methods for semantic segmentation and object class segmentation of natural images, an important step towards general scene understanding. We use the paradigm of structured prediction, which allows for a principled integration of context and object relations. We focused on *learning* of structural models and the interaction of inference and learning in the neighborhood models typically employed for semantic segmentation. We presented an open source software implementation of a variety of popular learning algorithms for structural support vector machines, together with a thorough evaluation of their properties, in particular when using approximate inference. Our software provides a foundation for future research into learning, inference and models for computer vision by providing extensive examples and benchmarks.

We showed that effective use of available inference mechanisms enables exact learning, even in the presence of loops in the underlying factor graph. Our methods achieve competitive performance with similar methods on the PASCAL VOC 2010 dataset, and improve upon state-of-the-art results on the MSRC-21 dataset. We demonstrated the power of conditional interactions by learning spatial interactions in an RGB-D setting. Here, our approach improves upon the state-of-the art on the NYU V2 benchmark for annotation of semantic structure classes.

We also presented a novel approach for clustering based on information theoretic principles. Our algorithm improves upon methods from the literature in finding pre-defined classes on a wide range of datasets. This indicates that in the task of extracting superpixels, we can also hope to achieve better results than the k-means based SLIC algorithm that we used.

As manual annotation of images for learning semantic segmentation and object class recognition is laborious and error-prone, we suggested a method to learn object class segmentation for complex object classes from image-level annotations

alone. Our approach is formulated using multiple instance learning over a set of candidate segments. We demonstrated the feasibility and effectiveness of our approach on the challenging GRAZ-02 dataset of street scenes.

8.1 Future Directions

There are several directions for future research that we think would be interesting to pursue as an extension of the presented results:

Large-Scale Weakly Supervised Object Class Segmentation We demonstrated a new method for object class segmentation using only weak supervision. One of the main advantages of such a method is that it is potentially able to exploit the large amount of weakly labeled data that is available on the internet. Using additional, weakly labeled training data, and evaluating on the given, manually annotated data, is therefore a promising path for improving the presented results.

Cached Inference for BCFW We saw in Chapter 5 that the 1-slack cutting plane algorithm benefits immensely from caching inference results during training. Therefore, investigating the influence of caching for BCFW (see Section 4.2.4) seems a promising topic for future research.

Theoretical Analysis of the n-Slack Algorithm As we have seen in Chapter 5, the n-slack algorithm often converges very fast in terms of passes over the training data This is in stark contrast to the known theoretical convergence guarantee, which is the slowest of all the algorithms we considered with $O(\frac{1}{\epsilon^2})^*$. It seem as if the approach of Lacoste-Julien et al. [2013] can yield a better convergence guarantee, but it is also worth investigating the direction pursued by Shalev-Shwartz and Zhang [2012].

Inference Machines Recently Stoyanov et al. [2011] started a new trend in structured prediction, which is sometimes called "inference machines". The basic

*This is in terms of calls to the QP. We are not aware of any analysis in terms of inference calls or passes over the training set.

principle is simple: the process of *prediction using a given inference procedure* is viewed as a feed-forward method for prediction, and parameters of this prediction process are optimized directly using empirical risk minimization. The work of Stoyanov et al. [2011] used loopy belief propagation as their inference algorithm and the optimization is carried out simply using gradient descent on the non-convex but differentiable loss function. Other recent work in this direction includes Krähenbühl and Koltun [2013], who used mean-field inference in a fully connected conditional random field and Jancsary et al. [2013], who used closed form inference in a Gaussian CRF. While these algorithms show great promise, their relation to the traditional approach of structured prediction used in this work is mostly unclear. In particular, if exact traditional learning is possible in a model, it is uncertain how much in accuracy and efficiency can be gained by direct empirical risk minimization. Only limited empirical comparison is available, and we are not aware of theoretical work in this direction, leaving much room for future investigation.

Non-Linear Models In this work, we only considered models that are linear in the input features—though features are highly non-linear in the original input pixels. Allowing non-linear interactions increases the representational power of a CRF, possibly leading to more accurate prediction results. Kernelization of structural support vector machines is straight-forward in theory, but had only limited success in the context of CRFs for image segmentation [Lucchi et al., 2012]. Two major alternatives for non-linear CRFs were proposed in the literature, conditional neural fields [Peng et al., 2009] based on neural networks, and decision tree fields (DTFs) [Nowozin et al., 2011], based on decision trees. Conditional neural fields have only been applied to sequence classification so far, and extending them to our setting of semantic image segmentation would be very interesting. DTFs on the other hand have been applied to loopy graphs for image processing, but not for higher-level tasks such as semantic segmentation. If it is possible to include context in a meaningful way, it might be possible to address even object-centric tasks such as object class segmentation with DTFs.

Higher Order Potentials and Latent Variable Models While non-linear potentials would allow for more complex interactions between inputs and labelings, introducing higher order potentials [Kohli et al., 2009, Ladicky et al., 2009]

or latent variables [Dann et al., 2012] allows the model to express more complex interactions within the output variables. Possible examples are consistency of larger regions, learning parts or learning of scene classes and co-occurrences. In principle, higher order potentials and latent variable models are equivalent, in that each energy function expressed in either form can be transformed into an energy function of the other kind. In practice, learning of higher order potentials for semantic segmentation has received little attention, while approaches using latent variables are often limited by the non-convexity of learning. It would be interesting to compare current methods using latent variable and higher order approaches, and see how these interact with different inference and learning schemes.

Feature Design This work mostly focused on learning methods, and less on the input—with the exception of Chapter 7, which explores the use of 3D features for semantic segmentation of indoor scenes. It is clear, however, that the input features play an important role in the performance of any system. Using our approach for exact learning of loopy graphs, it seems to be worthwhile to revisit the works of Nowozin et al. [2010] and Lucchi et al. [2011], that evaluate the impact of input features and piecewise training, and of the importance of global constraints versus global features, respectively. In particular the importance of features for pairwise potentials has been somewhat overlooked in the computer vision literature, often being reduced to a single constant or contrast sensitive feature.

9 Bibliography

Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *Pattern Analysis and Machine Intelligence*, 2012.

Feliv V Agakov and David Barber. Kernelized infomax clustering. In *Neural Information Processing Systems*, 2006.

Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning. 2003.

Arindam Banerjee, Srujana Merugu, Inderjit S Dhillon, and Joydeep Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 6, 2005.

James Bergstra, Rémi Bardenet, Yoshua Bengio, Balázs Kégl, et al. Algorithms for hyper-parameter optimization. In *Neural Information Processing Systems*, 2011.

Andrew Blake, Pushmeet Kohli, and Carsten Rother. *Markov random fields for vision and image processing*. MIT Press, 2011.

Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence*, 23(11), 2001.

Steve Branson, Oscar Beijbom, and Serge Belongie. Efficient large-scale structured learning. In *Computer Vision and Pattern Recognition*, 2013.

Joao Carreira and Cristian Sminchisescu. Constrained parametric min-cuts for automatic object segmentation. In *Computer Vision and Pattern Recognition*, 2010.

Bryan Catanzaro, Bor-Yiing Su, Narayanan Sundaram, Yunsup Lee, Mark Murphy, and Kurt Keutzer. Efficient, high-quality image contour detection. In *International Converence on Computer Vision*, 2009.

Yixin Chen, Jinbo Bi, and James Z Wang. MILES: Multiple-instance learning via embedded instance selection. *Pattern Analysis and Machine Intelligence*, 2006.

C Chow and C Liu. Approximating discrete probability distributions with dependence trees. *Information Theory*, 14(3), 1968.

Luis Pedro Coelho. Mahotas: Open source software for scriptable computer vision. *Journal of Open Research Software*, 1, 2013.

Camille Couprie, Clement Farabet, Laurent Najman, and Yann LeCun. Indoor semantic segmentation using depth information. In *International Conference on Learning Representations*, 2013.

Joachin Dahl and Lieven Vandenberghe. Cvxopt: A python package for convex optimization. In *European Converence on Computer Vision*, 2006.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition*, volume 1, 2005.

Christoph Dann, Peter Gehler, Stefan Roth, and Sebastian Nowozin. Pottics–the potts topic model for semantic image segmentation. In *German Conference on Pattern Recognition (DAGM)*, 2012.

Inderjit S Dhillon, Subramanyam Mallela, and Rahul Kumar. A divisive information theoretic feature clustering algorithm for text classification. *Journal of Machine Learning Research*, 3, 2003.

Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2), 1997.

Ian Endres and Derek Hoiem. Category independent object proposals. In *European Converence on Computer Vision*, 2010.

Mark Everingham, Luc Van Gool, Christopher K I Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88, 2010.

Lev Faivishevsky and Jacob Goldberger. A nonparametric information theoretic clustering algorithm. In *International Conference on Machine Learning*, 2010.

Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence*, 2013.

Thomas Finley and Thorsten Joachims. Training structural SVMs when exact inference is intractable. In *International Conference on Machine Learning*, 2008.

Brian Fulkerson, Andrea Vedaldi, and Stefano Soatto. Class segmentation and object localization with superpixel neighborhoods. In *International Converence on Computer Vision*, 2009.

Thomas Gärtner, Peter A Flach, Adam Kowalczyk, and Alexander J Smola. Multi-instance kernels. In *International Conference on Machine Learning*, 2002.

Erhan Gokcay and Jose C Principe. Information theoretic clustering. *Pattern Analysis and Machine Intelligence*, 24, 2002.

Ryan Gomes, Andreas Krause, and Pietro Perona. Discriminative clustering by regularized information maximization. In *Neural Information Processing Systems*, 2010.

Josep M Gonfaus, Xavier Boix, Joost van de Weijer, Andrew D Bagdanov, Joan Serrat, and Jordi Gonzalez. Harmony potentials for joint classification and segmentation. In *Computer Vision and Pattern Recognition*, 2010.

John C Gower and GJS Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, 1969.

Oleksandr Grygorash, Yan Zhou, and Zach Jorgensen. Minimum spanning tree based clustering algorithms. In *International Converence on Tools with Artificial Intelligence*, 2006.

Tamir Hazan and Raquel Urtasun. A primal-dual message-passing algorithm for approximated large scale structured prediction. In *Neural Information Processing Systems*, 2010.

Xuming He, Richard S Zemel, and Miguel A Carreira-Perpinán. Multiscale conditional random fields for image labeling. In *Computer Vision and Pattern Recognition*, volume 2, 2004.

Alfred O Hero III and Olivier J J Michel. Asymptotic theory of greedy approximations to minimal k-point random graphs. *Information Theory*, 45, 1999.

Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-Time Plane Segmentation using RGB-D Cameras. In *RoboCup International Symposium*, 2011.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2, 1985.

Jeremy Jancsary, Sebastian Nowozin, and Carsten Rother. Learning convex qp relaxations for structured prediction. In *International Conference on Machine Learning*, 2013.

Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1), 2009.

Sepandar D Kamvar, Dan Klein, and D Manning, Christopher. Spectral learning. In *International Joint Conference of Artificial Intelligence*, 2003.

Jörg H Kappes, Bjoern Andres, Fred A Hamprecht, Christoph Schnörr, Sebastian Nowozin, Dhruv Batra, Sungwoong Kim, Bernhard X Kausler, Jan Lellmann, Nikos Komodakis, et al. A comparative study of modern inference techniques for discrete energy minimization problems. In *Computer Vision and Pattern Recognition*, 2013.

Pushmeet Kohli, Philip HS Torr, et al. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision*, 82(3), 2009.

Nikos Komodakis. Efficient training for pairwise or higher order crfs via dual decomposition. In *Computer Vision and Pattern Recognition*, 2011.

Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. 2012.

Philipp Krähenbühl and Vladlen Koltun. Parameter learning and convergent inference for dense random fields. In *International Conference on Machine Learning*, 2013.

Simon Lacoste-Julien, Mark Schmidt, and Francis Bach. A simpler approach to obtaining an o (1/t) convergence rate for projected stochastic subgradient descent. *arXiv preprint arXiv:1212.2002*, 2012.

Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate frank-wolfe optimization for structural svms. In *International Conference on Machine Learning*, 2013.

L'ubor Ladicky, Chris Russell, Pushmeet Kohli, and Philip HS Torr. Associative hierarchical CRFs for object class image segmentation. In *International Converence on Computer Vision*, 2009.

Christian Leistner, Amir Saffari, and Horst Bischof. MIForests: Multiple-instance learning with randomized trees. *European Converence on Computer Vision*, 2010.

Victor Lempitsky, Carsten Rother, Stefan Roth, and Andrew Blake. Fusion moves for markov random field optimization. *Pattern Analysis and Machine Intelligence*, 32(8), 2010.

Fuxin Li and Cristian Sminchisescu. Convex multiple-instance learning by estimating likelihood ratio. In *Neural Information Processing Systems*, 2010.

Fuxin Li, Joao Carreira, and Cristian Sminchisescu. Object recognition as ranking holistic figure-ground hypotheses. In *Computer Vision and Pattern Recognition*, 2010.

Fuxin Li, Joao Carreira, Guy Lebanon, and Cristian Sminchisescu. Composite statistical inference for semantic segmentation. In *Computer Vision and Pattern Recognition*, 2013.

Yu-Feng Li, James R Kwok, Ivor W Tsang, and Zhi-Hua Zhou. A convex method for locating regions of interest with multi-instance learning. *Machine Learning and Knowledge Discovery in Databases*, 2009.

Stuard P Lloyd. Least squares quantization in PCM. *Information Theory*, 28, 1982.

David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2), 2004.

Aurélien Lucchi, Yunpeng Li, Xavier Boix, Kevin Smith, and Pascal Fua. Are spatial and global constraints really necessary for segmentation? In *International Converence on Computer Vision*, 2011.

Aurélien Lucchi, Yunpeng Li, Kevin Smith, and Pascal Fua. Structured image segmentation using kernelized features. In *European Converence on Computer Vision*, 2012.

Aurélien Lucchi, Yunpeng Li, and Pascal Fua. Learning for structured prediction using approximate subgradient descent with working sets. In *Computer Vision and Pattern Recognition*, 2013.

James B MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

Michael Maire, Pablo Arbeláez, Charless Fowlkes, and Jitendra Malik. Using contours to detect and localize junctions in natural images. In *Computer Vision and Pattern Recognition*, 2008.

Olvi L Mangasarian and Edward W Wild. Multiple instance classification via successive linear programming. *Journal of Optimization Theory and Applications*, 137(3), 2008.

March, William B, Ram, Parikshit, and Gray, Alexander G. Fast Euclidean minimum spanning tree: algorithm, analysis, applications. In *International Conference on Knowledge Discovery and Data Mining*, 2010.

Marcin Marszatek and Cordelia Schmid. Accurate object localization with shape masks. In *Computer Vision and Pattern Recognition*, 2007.

André FT Martins, Mário AT Figueiredo, Pedro MQ Aguiar, Noah A Smith, and Eric P Xing. An augmented lagrangian approach to constrained map inference. In *International Conference on Machine Learning*, 2011.

Amir Massoud Farahmand, Csaba Szepesvári, and Jean-Yves Audibert. Manifold-adaptive dimension estimation. In *International Conference on Machine Learning*, 2007.

Ofer Meshi, David Sontag, Tommi Jaakkola, and Amir Globerson. Learning efficiently with approximate inference via dual losses. In *International Conference on Machine Learning*, 2010.

Joris M Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11, 2010. URL `http://www.jmlr.org/papers/volume11/mooij10a/mooij10a.pdf`.

Roozbeh Mottaghi, Sanja Fidler, Jian Yao, Raquel Urtasun, and Devi Parikh. Analyzing semantic segmentation using hybrid human-machine crfs. In *Computer Vision and Pattern Recognition*, 2013.

Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: analysis and an algorithm. In *Neural Information Processing Systems*, 2002.

Nam Nguyen. A New SVM Approach to Multi-instance Multi-label Learning. In *International Conference on Data Mining*, 2010.

Sebastian Nowozin and Christoph H Lampert. *Structured learning and prediction in computer vision*. Now publishers Inc, 2011.

Sebastian Nowozin, Peter V Gehler, and Christoph H Lampert. On parameter learning in CRF-based approaches to object class image segmentation. In *European Converence on Computer Vision*. 2010.

Sebastian Nowozin, Carsten Rother, Shai Bagon, Toby Sharp, Bangpeng Yao, and Pushmeet Kohli. Decision tree fields. In *International Converence on Computer Vision*, 2011.

Andreas Nüchter and Joachim Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 2008.

Stephen Malvern Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12, 2011.

Jian Peng, Liefeng Bo, and Jinbo Xu. Conditional neural fields. In *Neural Information Processing Systems*, 2009.

William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 1971.

Nathan Ratliff, J. Andrew (Drew) Bagnell, and Martin Zinkevich. (online) subgradient methods for structured prediction. In *Artificial Intelligence and Statistics*, March 2007.

Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010.

Carl Yuheng Ren and Ian Reid. gSLIC: a real-time implementation of SLIC superpixel segmentation. *University of Oxford, Department of Engineering, Technical Report*, 2011.

Xiaofeng Ren, Liefeng Bo, and Dieter Fox. RGB-(D) Scene Labeling: Features and Algorithms. In *Computer Vision and Pattern Recognition*, 2012.

Carsten Rother, Vladimir Kolmogorov, Victor Lempitsky, and Martin Szummer. Optimizing binary MRFs via extended roof duality. In *Computer Vision and Pattern Recognition*, 2007.

Hannes Schulz and Sven Behnke. Object-class segmentation using deep convolutional neural networks. In Barbara Hammer and Thomas Villmann, editors, *Proceedings of the DAGM Workshop on New Challenges in Neural Computation 2011*, volume 5 of *Machine Learning Reports*, 2011.

Hannes Schulz and Sven Behnke. Learning object-class segmentation with convolutional neural networks. In *European Symposium on Artificial Neural Networks (ESANN)*, volume 3, 2012.

Sunando Sengupta, Paul Sturgess, Philip HS Torr, et al. Automatic dense visual semantic mapping from street-level imagery. In *Intelligent Robots and Systems*, 2012.

Sunando Sengupta, Eric Greveson, Ali Shahrokni, and Philip HS Torr. Urban 3d semantic modelling using stereo vision. In *International Conference on Robotics and Automation*, 2013.

Shai Shalev-Shwartz and Tong Zhang. Proximal stochastic dual coordinate ascent. *arXiv preprint arXiv:1211.2717*, 2012.

Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming*, 127, 2011.

Ohad Shamir and Tong Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. *arXiv preprint arXiv:1212.1824*, 2012.

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence*, 22, 2000.

Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *European Converence on Computer Vision*, 2006.

Nathan Silberman and Rob Fergus. Indoor scene segmentation using a structured light sensor. In *Computer Vision Workshops (ICCV Workshops)*, 2011.

Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *European Converence on Computer Vision*, 2012.

Noam Slonim and Naftali Tishby. Agglomerative information bottleneck. *Neural Information Processing Systems*, 1999.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Artificial Intelligence and Statistics*, 2011.

Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3, 2003.

Jörg Stückler, Nenad Biresev, and Sven Behnke. Semantic mapping using object-class segmentation of RGB-D images. In *Intelligent Robots and Systems*, 2012.

Jörg Stückler, Benedikt Waldvogel, Hannes Schulz, and Sven Behnke. Dense Real-Time Mapping of Object-Class Semantics from RGB-D Video. *Submitted*, 2013. URL `http://www.ais.uni-bonn.de/papers/JRTIP_2013_Stueckler_RF_Fusion.pdf`.

Martin Szummer, Pushmeet Kohli, and Derek Hoiem. Learning CRFs using graph cuts. In *European Converence on Computer Vision*, 2008.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. *Neural Information Processing Systems*, 2003.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, Yasemin Altun, and Yoram Singer. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2), 2006.

Koen EA van de Sande, Theo Gevers, and Cees GM Snoek. Evaluating color descriptors for object and scene recognition. *Pattern Analysis and Machine Intelligence*, 32(9), 2010.

Andrea Vedaldi and Brian Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. In *Computer Vision and Pattern Recognition*, 2010.

Jakob Verbeek and Bill Triggs. Region classification with Markov field aspect models. In *Computer Vision and Pattern Recognition*, 2007.

Alexander Vezhnevets and Joachim M Buhmann. Towards weakly supervised semantic segmentation by means of multiple instance and multitask learning. In *Computer Vision and Pattern Recognition*, 2010.

Alexander Vezhnevets, Vittorio Ferrari, and Joachim M Buhmann. Weakly supervised semantic segmentation with a multi-image model. In *International Converence on Computer Vision*, 2011.

Song Wang and Jeffrey M Siskind. Image segmentation with Ratio Cut. *Pattern Analysis and Machine Intelligence*, 2003.

Michael Wick, Khashayar Rohanimanesh, Kedar Bellare, Aron Culotta, and Andrew McCallum. Samplerank: Training factor graphs with atomic gradients. In *International Conference on Machine Learning*, 2011.

Wei Xia, Zheng Song, Jiashi Feng, Loong-Fah Cheong, and Shuicheng Yan. Segmentation over detection by coupled global and local sparse representations. In *European Converence on Computer Vision*, 2012.

Jian Yao, Sanja Fidler, and Raquel Urtasun. Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation. In *Computer Vision and Pattern Recognition*, 2012.

Charles T Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, 100, 1971.

Zheng-Jung Zha, Xian-Sheng Hua, Tao Mei, Jingdong Wang, Guo-Jun Qi, and Zengfu Wang. Joint multi-label multi-instance learning for image classification. In *Computer Vision and Pattern Recognition*, 2008.

Qi Zhang and Sally A Goldman. EM-DD: An improved multiple-instance learning technique. *Neural Information Processing Systems*, 2, 2002.

Zhi-Hua Zhou and Min-Ling Zhang. Multi-instance multi-label learning with application to scene classification. In *Neural Information Processing Systems*, 2006.

Zhi-Hua Zhou, Yu-Yin Sun, and Yu-Feng Li. Multi-instance learning by treating instances as non-iid samples. In *International Conference on Machine Learning*, 2009.