Reasoning about Object Instances, Relations and Extents in RGBD Scenes

by

Nathan Silberman

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January 2015

Rob Fergus

David Sontag

Dedication

To my family

Acknowledgements

My work over the last few years would not have been possible without the efforts of my PhD advisors. I would like to thank Rob Fergus and David Sontag for the many hours they spent providing insight, advice and direction. Thanks to Rob for taking me on as a student, helping me forge a path through 3D scene understanding and teaching me how to approach problems in research. Thanks to David for his patience and his unrelenting ability to see problems from different angles and propose creative solutions. I would also like to thank Lakshmi Subramanian who opened the door to my academic career and made the choice of starting a PhD an easier one.

Many thanks to my collaborators. In particular, thanks to Derek Hoiem for giving me a better intuition for thinking about scene understanding and helping build the NYU Depth V2 dataset. Thanks to Pushmeet Kohli for teaching me about graphical models and providing much insightful feedback on my work.

Many thanks to the other members of NYU's learning and vision lab including Yann LeCun who provided a constant source of discussion and perspective and Dilip Krishnan, Koray Kavukcuoglu and Graham Taylor who's advice and guidence was especially helpful when I was just getting started. Special thanks to Matt Zeiler who made late-night hours in the lab enjoyable through both technical and laughable discussion.

Thanks to Eyal Ofek and Ran Gal for an excellent experience at MSR and special thanks to my MSR mentor Lior Shapira for our frequent discussions on efficacy versus elegance, philosophizing, and life outside the lab.

Thanks to the entire Google Innerspace team for a fantastic summer and to Scott Satkin for making it all possible.

Thanks to Raquel Urtasun, Sanja Fidler, Andreas Geiger, Antonio Torralba, Derek Hoiem and Jianxiong Xiao for helping to put together the RMRC workshops.

Finally, I'd like to thank my siblings, Debbie, Esther and Eyal for their constant encouragement and my parents who continue to be a source of inspiration, stability and support in my life.

Abstract

The vast majority of literature in scene parsing can be described as semantic pixel labeling or semantic segmentation: predicting the semantic class of the object represented by each pixel in the scene. Our familiar perception of the world, however, provides a far richer representation. Firstly, rather than just being able to predict the semantic class of a location in a scene, humans are able to reason about object instances. Discriminating between a region that might represent a single object versus ten objects is a crucial and basic faculty. Secondly, rather than reasoning about objects as merely occupying the space visible from a single vantage point, we are able to quickly and easily reason about an object's true extent in 3D. Thirdly, rather than viewing a scene as a collection of objects independently existing in space, humans exhibit a representation of scenes that is highly grounded through a intuitive model of physics. Such models allow us to reason about how objects relate physically: via physical support relationships.

Instance segmentation is the task of segmenting a scene into regions which correspond to individual object instances. We argue that this task is not only closer to our own perception of the world than semantic segmentation, but also directly allows for subsequent reasoning about a scenes constituent elements. We explore various strategies for instance segmentation in indoor RGBD scenes.

Firstly, we explore tree-based instance segmentation algorithms. The utility of trees for semantic segmentation has been thoroughly demonstrated and we adapt them to instance segmentation and analyze both greedy and global approaches to inference.

Next, we investigate exemplar-based instance segmentation algorithms, in which a set of representative exemplars are chosen from a large pool of regions and pixels are assigned to exemplars. Inference can either be performed in two stages, exemplar selection followed by pixel-to-exemplar assignment, or in a single joint reasoning stage. We consider the advantages and disadvantages of each approach.

We introduce the task of support-relation prediction in which we predict which objects are physically supporting other objects. We propose an algorithm and a new set of features for performing discriminative support prediction, we demonstrate the effectiveness of our method and compare training mechanisms.

Finally, we introduce an algorithm for inferring scene and object extent. We demonstrate how reasoning about 3D extent can be done by extending known 2D methods and highlight the strengths and limitations of this approach.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

While humans expend little effort to parse and interpret complex visual scenes, state-of-the-art computer algorithms still struggle to match such abilities. Most efforts to mimic human visual recognition capabilities constitute of one of three tasks. Firstly, image classification involves assigning an image to one of a number of discrete labels. Secondly, detection involves placing a rectangular bounding box around an object of a predefined class. Thirdly, semantic segmentation involves assigning each pixel in an image to one of a number of discrete labels. While algorithms that perform these tasks have advanced to the point that they are commonly and profitably used in both academia and industry, these tasks alone still represent a poor approximation of the visual experience.

Where classification produces a single label, an image might correctly be described by various means. Detection algorithms rely heavily on finding quantifyable objects and their geometry but many images contain materials, (such as liquids, grass and sand) that are amorphous and do not exhibit any typical shape. While semantic segmentation produces a map of pixels to semantic classes, it fails to represent information about object instances and cannot discriminate between an image that contains 1 chair and 5 chairs.

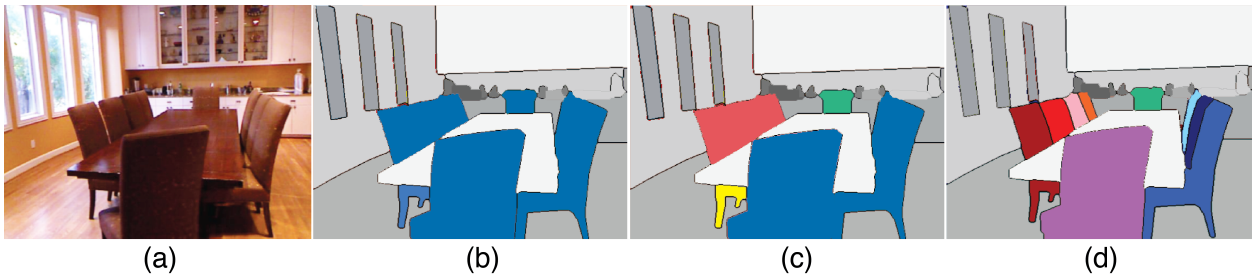


Figure 1.1: An illustration of the limits of semantic segmentation: (a) the input image. (b) a perfect semantic segmentation; note all of the chair pixels are labeled blue. (c) a naive instance segmentation in which all connected components of the same class are considered separate instances of the chair class. (d) a correct instance segmentation, which correctly reasons about instances within contiguous segments and across occlusions.

In constrast, instance segmentation, the task of grouping pixels together that belong to the same object instance and labeling each region with a semantic class, is able to deal with all of these issues. Instance segmentations can be used to describe a scene as it contains a list of the scene's objects. Furthermore, it's not limited to articulated objects and can be used to discriminate counts of objects.

Furthermore, while all of these tasks focus heavily on semantics, humans are able to reason accurately about a scene and its parts in ways that are non semantic. Without knowing an object's semantic class, we are able to reason about how that object is situated in a room and how it's physically supported. We understand the consequences of physical interaction and are able to determine by visual inspection alone whether an object perched on another will likely rest safely or fall. Finally, human's intuitive sense of geometry allows us to reason about objects' true extents in 3D space. Unfortunately, such tasks cannot be performed via a classification, detection or semantic segmentation. Reasoning about both physical support and object extent inherantly requires an instance segmentation of the scene.

Reasoning about instances, support and extent also have various practical benefits. Robotics applications in which a single object must be grasped or moved require a clean instance segmentation for estimating where on the object to grab or push. A robot that can only lift

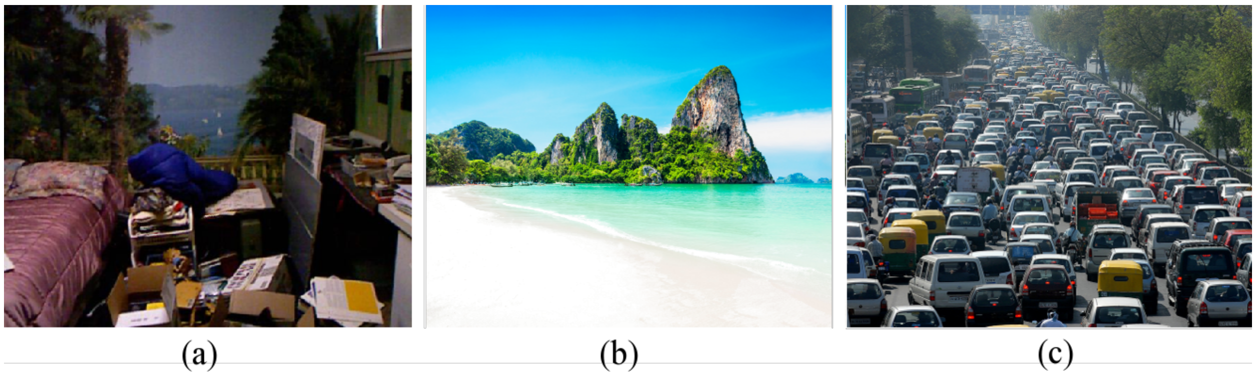


Figure 1.2: What is the right representation for a scene? (a) A single label such as "bedroom" doesn't capture the messiness and clutter of the scene while "messy bedroom" doesn't describe the objects present. (b) Detection is very useful for detecting articulated objects but is less useful for labeling the sand or water in the center image. (c) Semantic segmentation can indicate the presence of cars but cannot discriminate between a single car and a multitude of crowded cars In contrast instance segmentation provides a representation which captures all of these elements.

a certain number of objects needs to be able to count the number of objects present before acting. Search queries such as "three cups in a row" require understanding the difference between a single instance and three. Reasoning about support relations is also a critical faculty required of robotics algorithms. If a robot aims to obtain a book supporting a cup of hot coffee, it must first infer that the book is supporting the cup before it can decide to move the cup first before recovering the book. Finally, extent reasoning is important for augmented reality. Obtaining high fidelity scene geometry is important for many augmented reality applications such as gaming, virtual furniture placement and shopping experiences and augmented television experiences. However, collecting geometry information from a scene is tedious for the average user and more intelligent algorithms that can infer plausible scene geometry from a small sample will speed up the adoption and ease of use of augmented reality games and tools.

These alternative domains of visual reasoning have attracted increased attention concurrent with another trend in computer vision: combining RGB and depth (RGBD) inputs for visual processing. With the introduction of cheap depth cameras, one is able to circumvent the problem of reasoning about scene geometry from color alone. The depth image provided by a depth camera provides direct access to part of the scene's geometry and allows for the exploration of algorithms that are able to reason about the full room geometry, object's true physical extents, physical support relations and improved semantic and instance segmentation.

To help advance research in RGBD scene understanding, we introduce two new RGBD scene datasets. Where previously proposed datasets were either too small or contained artificial or simple objects and scenes, our datasets contain real world indoor scenes exhibiting large amounts of clutter, occlusion and lighting variation as well as a large number of different semantic classes. These datasets have become the de-facto standard for RGBD-based visual recognition tasks including semantic segmentation, depth from RGB, surface normals from RGB, 3D detection and instance segmentation.

Our first thread of investigation concerns semantic segmentation. We adapt several RGB based features to the RGBD domain, introduce a novel 3D location prior and demonstrate the effectiveness of the prior in improving semantic segmentation.

Next, we study the problem of tree-based instance segmentation. Segmentation trees are commonly used by state-of-the-art methods in semantic segmentation. We demonstrate how to adapt these trees to the task of instance segmentation and illustrate two methods of training these trees: the first greedy and the second global.

While the use of trees makes inference efficient, tree creation is imperfect and introduces errors from which our instance segmentation cannot recover. To address this limitation, we investigate exemplar-based instance segmentation algorithms. In these algorithms, exemplars are selected from a pool of regions and pixels are assigned to exemplars. This process can be done in two separate stages or jointly and we analyze the advantages and disadvantages of each.

We introduce the new task of support reasoning. In this task, we are given an instance segmentation and must reason about which objects in the scene are physically supported by which other objects. We demonstrate that a relatively simple set of hand-crafted features allows us to train a discriminative model that performs the task of support inference with a high degree of accuracy.

We also introduce a method for performing automatic extent inference of rooms and planar object surfaces. While a high fidently thorough 3D mapping of an interior space would be of great use for gaming and augmented reality, such a model is expensive and time consuming to obtain. We introduce an algorithm that is able to reason about scene and planar surface extents. Given a partial model obtained from a limited viewpoint, we automatically complete the boundaries of the room and extend the planar surfaces of the scene to provide a physically plausible 3D scene model.

Chapter 2

Background

2.1 RGBD Datasets

Early usage of RGBD data was heavily focused on outdoor environments. Saxena *et al.* [?] [?] introduced two datasets of outdoor scenes for the purposes of depth estimation from RGB images. Gould *et al.* [?] extended this work by adding semantic labels for several classes. Munoz *et al.* [?] created a dataset of sparse depth and dense RGB images from a laser scanner mounted on a driving car. In the indoor domain, Gould *et al.* [?] combined laser range finder data with RGB images to create a dataset of indoor images while Quigley *et al.* [?] used a laser-line scanner to create a dataset of indoor scenes with several classes to aid a robotic door-opening task. These early works introduced the task of depth estimation from RGB images and allowed for detection and segmentation algorithms that reasoned more explicitely in 3D than their RGB-only counterparts. While being novel and innovate, these datasets are either exclusively limited to outdoor scenes or capture a set of indoor scenes with a large degree of homogeneity. Furthermore, the semantic classes captured are limited to a very small set of objects.

Following the release of the Kinect, a number of RGBD datasets were released. Lai *et al.* [?] introduce a dataset of single objects presented in front of a simple background. The images contain isolated objects with uncluttered backgrounds rather than entire scenes making their dataset qualitatively similar to COIL [?].

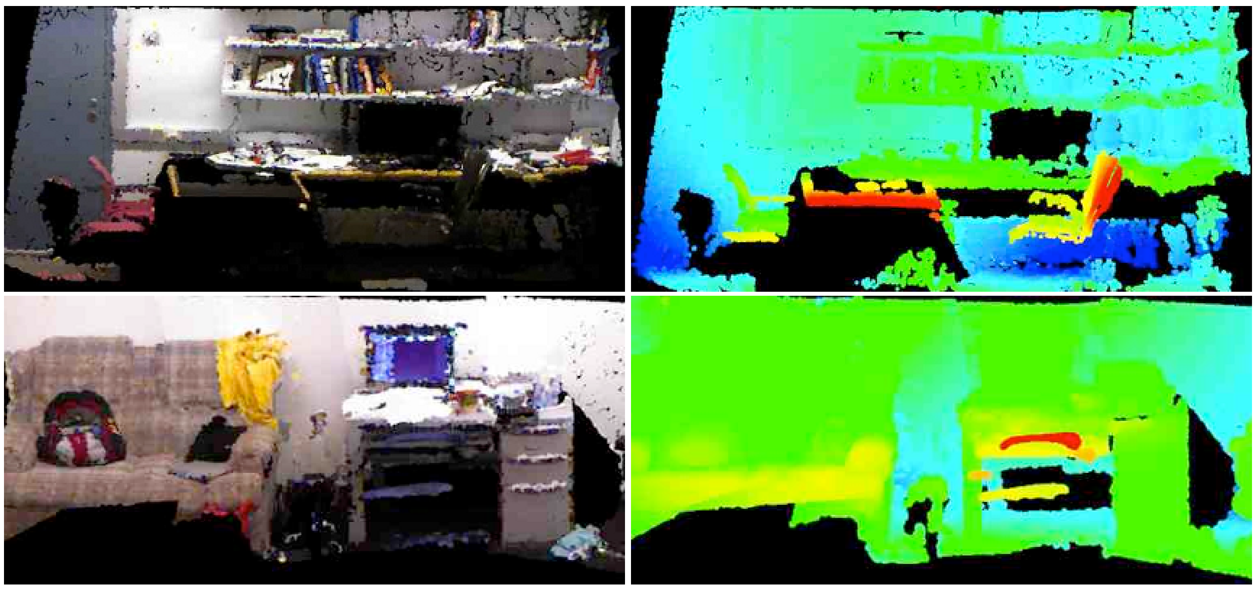


Figure 2.1: Example scenes from the Cornell RGBD dataset [?]. The dataset contains 24 labeled point clouds from office scenes and 28 labeled point clouds from home interiors.

Koppula *et al.* [?] introduce a dataset of indoor point clouds for a number of home and office scenes. Unlike our datasets, theirs is focused on point clouds and not single images (Figure 2.1). Karayev *et al.* [?] introduce a similar dataset of indoor scenes. It contains many more object labels than previous datasets but its labels are limited to bounding boxes (Figure 2.2 making it unsuitable to either semantic or instance segmentation. Finally Xiao *et al.* [?] introduce a dataset of indoor scenes qualitatively similar to our own with individually labeled frames that can be propagated to entire video sequences. However, theirs currently contains only a single scene and video sequence completely labeled.

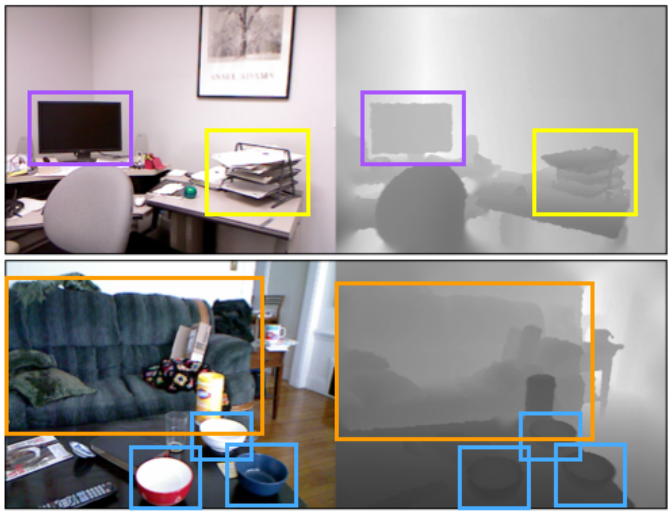


Figure 2.2: Example scenes from the Berkeley 3-D Object Dataset. [?]. The dataset contains 848 RGBD images but is limited to bounding box labels, rather than semantic or instance segmentation labels.

2.2 Semantic Segmentation

The task of semantic segmentation has attracted much attention. Inspired by early work including Kumar *et al.* [?], He *et al.* [?] and Shotten *et al.* [37], many semantic segmentation algorithms follow a similar approach. First, the likelihood of each pixel taking on a particular label is determined via a classifier using local appearance information. While such early approaches to semantic segmentation were characterized by pixel-based features and low order potentials, various region-based features and higher order potentials have been proposed leading to improved semantic segmentation results. Kohli et al [22] produce multiple superpixelations with various parameters and introduce a higher-order terms encouraging pixels that fall into the same superpixel regions to take on the same semantic label. Ladicky *et al.* [?] use the output of a series of object detectors to create higher order potentials that encourage the pixel-wise labeling to be consistent with the detections.

Nonparametric methods have also been proposed that transfer labels from a dataset to new images. Liu *et al.* [?] create a dataset of semantic segmentations of outdoor scenes. For a new image, they find a nearest neighbor match in their dataset and map the semantic labels onto the query image by computing a flow-vector field from the dataset image to the query image. Given a query image Isola and Liu [?] use a dictionary of scenes and regions to create a label collage from various similar and related images in their dataset.

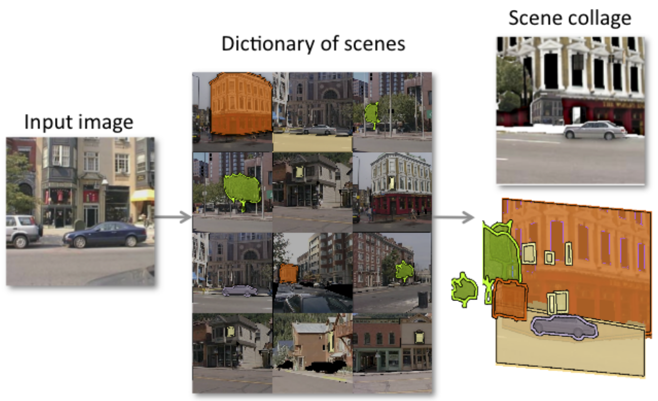


Figure 2.3: In the work of Isola and Liu [?] query images are assigned semantic labels by combining labeled regions from a non-parameteric dataset of scenes.

Several approaches have explored various feature descriptors and algorithms applied to the RGBD domain. Ren *et al.* [?] introduce a series of kernel descriptors which exhibit excellent performance in RGBD semantic segmentation. Couprie *et al.* [?] train a multiscale convolutional neural network directly on RGBD image patches.

Once common thread in all of these methods is that they produce a semantic labeling from which instance information cannot easily be recovered. A single cluster of contiguous pixels taking on the same semantic label might represent any number of objects.

Various previous works use location priors to improve semantic segmentation accuracy. The seminal work of Shotton *et al.* [37] use a 2D location prior that captures the likelihood of a semantic class appearing in various parts of the image plane. Gould *et al.* [?] use a 2D relative location prior to encourage segmentations whose classes appear at predictable relative positions from one another. Inspired by the work of [?], Choi *et al.* [?] use a relative location prior by assuming a simple projective geometry based on the height of the object in the image plane.

2.3 Instance Segmentation

2.3.1 Foreground-Background Segmentation

Instance segmentation is inherant to the task of foreground-background segmentation. In this task, the pixels of an image are split into two sets: a foreground set of pixels and a background set of pixels. Early work in this area [?] demonstrated that a bayesian interpretation of the inference problem resulted in a formulation that could be optimized via max-flow min-cut algorithms. Shi and Malik [?] introduced an alternative graph partitioning scheme that takes both inter-group dissimilarity and intra-group similarity into account.

Subsequent work in foreground-background segmentation made much larger strides by focusing on assuming a specific semantic class [?], jointly reasoning over semantic classes and segmentations [?] [?] or allowed the task to be performed iteratively [?] [?] [?].

Unfortunately these algorithms are not well suited for cluttered scenes as they rely on there being a multi-model color distribution, a single large object in the foreground, or a highly discriminative shape that may be matched by a deformable template. In contrast, our work is able to produce an object-instance segmentation for multiple objects simultaneously. We make no prior assumptions about how many objects there are, nor their prominance in the scene.

2.3.2 Agglomerative Segmentation

Hoiem et al [17] introduced an agglomerative clustering method for greedily merging regions that belong to the same object instance. Ren and Shakhnarovich [34] explore the effects of using a more conservative and deeper agglomerative segmentation tree. We explore the use of these types of trees in Chapter 5.

2.3.3 Segmentation Trees

Motivated by the observation that a single segmentation of an image is unlikely to produce a perfect result, numerous approaches [9] [30] [35] [33] make use of multiple segmentations of an image. These approaches differ in how they use the various segmentations and whether the regions proposed are strictly hierarchical or structureless. Starting with [9], various efforts [22] [24] have used multiple independent segmentations in a pixel labeling task.

Several works [6] [18] use a structureless bag of regions to perform segmentation in which inference is composed of a search for the best non-overlapping set of regions that respects object boundaries. However, neither work uses semantics for reasoning. Rather than use arbitrary or structureless regions as input, an increasing number of approaches [26] [31] [27] [11] have been introduced that utilize hierarchical segmentations to improve semantic segmentation. These models are trained to cut a segmentation tree such that the resulting cut produces high pixel-wise accuracy.

Higher order losses for segmentation have also been previously explored. Tarlow et al. [40] introduce the Pascal Loss which smoothly minimizes the overlap score [10] of a single foreground/background segmentation. The Pascal Loss is closely related to the Coverage loss (Section 5.6).

A crucial difference between the two is that in the case of Pascal Loss, the best overlapping region is specified a priori (there is only one region, the foreground), whereas in the Coverage Loss, the best overlapping region can only be computing by jointly reasoning over every proposed region.

2.3.4 Clustering and Exemplar Based Segmentation

Various clustering-based schemes have also been introduced that might be used to produce object-instance segmentations. Comaniciu and Meer [8] introduced the Mean shift algorithm to iteratively cluster pixels together based on intensity and spatial location. Givoni and Frey presented Affinity Propagation [?] in which a subset of data points or pixels are used as exemplars to which all other points are assigned. Kim et al [21] create a hierarchy of segments over which a hypergraph is constructed. Each hyperedge in the graph indicates whether or not the associated regions belong to the same object instance.

Tarlow et al [?] introduce a novel message passing scheme for incorporating priors on the number of clusters in an exemplar-based clustering scheme. The recent work of He and Gould [?] is similar to our own in they too cast the object-instance segmentation problem as a labeling problem where the labels represent exemplars. However, their method assumes a single known semantic class and cannot infer the identity of multiple classes.

2.4 Support Reasoning

The goal of inferring support relations is most closely related to Gupta et al. [?], who apply heuristics inspired by physical reasoning to infer volumetric shapes, occlusion, and support in outdoor scenes. Our 3D cues provide a much stronger basis for inference of support, and our dataset enables us to train and evaluate support predictors that can cope with scene clutter and invisible supporting regions. Russell and Torralba [?] show how a dataset of user-annotated scenes can be used to infer 3D structure and support; our approach, in contrast, is fully automatic.

2.5 Extent Inference

In recent years there have been a number of papers that reason about the scene shape or extent. Barinova et al [?] geometrically parse a single image to identify edges, parallel lines and vanishing point, and a level horizon. Hedau et al [?] estimate the layout of a room from a single image, by using a generalized box detector. This approach is not well suited to completing either highly occluded scenes or non-rectangular objects. Ruiqi and Hoiem [14] attempt to predict occluded surfaces using a learning-based approach. This method is limited by the number of classes which can be learned, and does not infer the extent of the hidden objects. A more recent paper by the same authors [?] is more relevant to our approach. They detect a set of supporting horizontal surfaces in a single-image depth frame, and their extent is deduced using the features at each point and surrounding predictions. However, they do not attempt to infer the full scene layout and limit themselves to support-related objects.

Our work in extent inference is also related to scene augmnetation. Kim *et al.* [?] augment and accelerate reconstruction of complex scenes by carefully scanning and modeling objects which repeat in the scene, in advance. These objects (e.g. office chairs, monitors) are then matched in the noisy and incomplete point cloud of the larger scene, and used to augment it. This approach is less suited to capturing a new environment, in which we cannot model the repeating objects independently (given that there are any). Kim *et al.* [?] jointly estimate the 3D reconstruction of a scene, and the semantic labels associated with each voxel, on a coarse 3D volume. Their method works best with simple occlusions and does not extend the volume to estimate the overall shape of the room. Zheng *et al.* [**?**] employ physical and geometrical reasoning to reconstruct a scene. Their method relies on successful object segmentation, and a Manhattan-world assumption. They fill occluded parts of objects by extrapolating along the major axes of the scene. However, none of these methods can handle previously unseen object and surfaces with non-linear boundaries.

Chapter 3

NYU Depth Datasets

3.1 Introduction

With the introduction of the Microsoft Kinect, computer vision has exhibited a renewed interest in RGBD based vision algorithms and 3D reasoning techniques. Where laser range finders are expensive and often difficult to calibrate, the Kinect is cheap and easy to use. Its adoption along with similar cameras has led to improved results in various tasks such as pose estimation, pedestrian detection, tracking, segmentation and SLAM.

These efforts have been made possible by the release of 3D datasets that have allowed researchers to train models requiring a significant amount of data and that permit reliable comparison of algorithmic performance. Previous datasets (Section 2.1), however, have been limited in several ways. Firstly, most of the images in previous datasets do not exhibit diversity in terms of lighting conditions, object types or backgrounds. Many are taken outdoors or are of simple indoor office scenes. Secondly, while a few of these datasets have been annotated with semantic class labels [?] [?], these semantic classes are limited to a very small set of classes and rarey contain pixel-wise labels. Thirdly, previous datasets failed

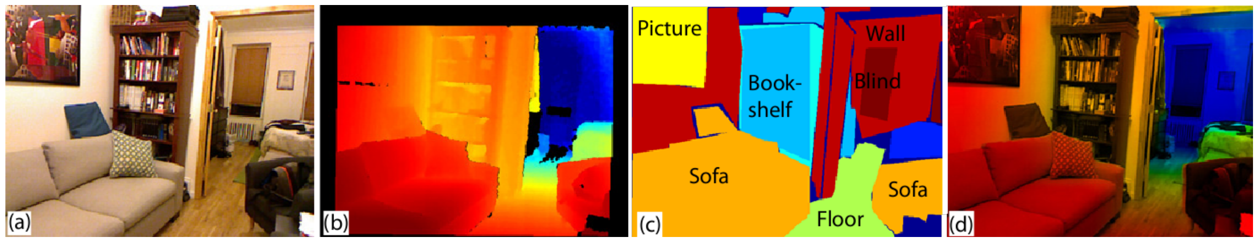


Figure 3.1: A typical indoor scene captured by the Microsoft Kinect. (a): Webcam image. (b) Raw depth map (red=close, blue=far). (c) Labels obtained via Amazon Mechanical Turk. (d) After a homography, followed by pre-processing to fill in missing regions, the depth map (hue channel) can be seen to closely aligned with the image (intensity channel).

to capture real-world indoor scenes relating to everyday life (bedrooms, bathrooms, living rooms, etc). Finally, no previous dataset recorded dense, pixel-wise semantic *and* instance labels.

To address these shortcomings, we introduce two new datasets: NYU Depth V1 and NYU Depth V2. These datasets capture a large number of varied, indoor, everyday scenes. The images exhibit large amounts of occlusion, lighting variation and clutter not commonly found in previous datasets of indoor scenes. In both datasets, around 1000 different semantic classes can be found.

Both datasets have made a large impact on the field of computer vision. They are both commmonly used for benchmarking various tasks including contour segmentation [?], [15], [?], [?] semantic segmentation [?], [?], [?], [?], [?] 3D detection [?], depth from RGB [?], [?], normals from RGB [?], [?], [?] and physics based reasoning [19].

3.2 Approach

Both datasets were collected using the Microsoft Kinect I sensor. This device uses structured light methods to give an accurate depth map of the scene, which can be aligned spatially and temporally with the device's webcam (see Fig. 3.1).

3.2.1 Capture Setup

The Kinect has two cameras: the first is a conventional VGA resolution webcam that records color video at 30Hz. The second is an infra-red (IR) camera that records a non-visible structured light pattern generated by the Kinect's IR projector. The IR camera's output is processed within the Kinect to provide a smoothed VGA resolution depth map, also at 30Hz, with an effective range of $\sim$0.7–6 meters. See Fig. 3.1(a) & (b) for typical output.

The Kinect requires a 12V input for the Peltier cooler on the IR depth camera, necessitating a mains adapter to power the device (USB sockets only provide 5V at limited currents). Since the mains adapter severely limits portability of the device, we remove it and connect a rechargeable 4200mAh 12V battery pack in its place. This is capable of powering the device for 12 hours of operation. The output from the Kinect was logged on a laptop carried in a backpack, using open-source Kinect drivers [?] to acquire time synchronized image, depth and accelerometer feeds. The overall system is shown in Fig. 3.2(a). To avoid camera shake and blur when capturing data, the Kinect was strapped to a motion-damping rig built from metal piping, shown in Fig. 3.2(b). The weights damp the motion and have a significant smoothing effect on the captured video.

Both the depth and image cameras on the Kinect were calibrated using a set of checkerboard images in conjunction with the calibration tool of Burrus [?]. This also provided the homography between the two cameras, allowing us to obtain precise spatial alignment between the depth and RGB images, as demonstrated in Fig. 3.1(d).

3.2.2 NYU Depth V1

We visited a range of indoor locations within a large US city, gathering video footage with our capture rig. These mainly consisted of residential apartments, having living rooms,

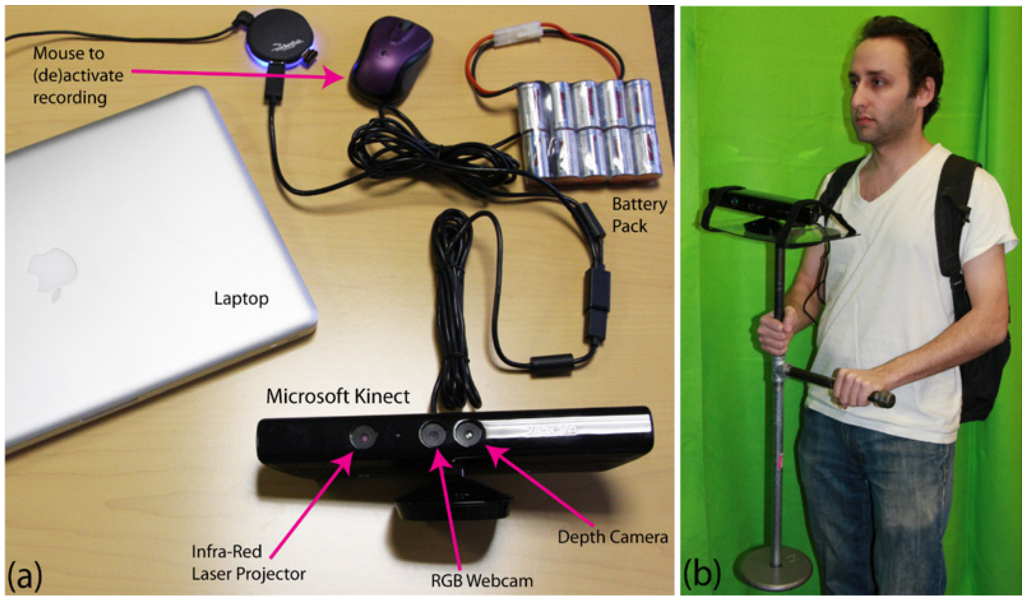


Figure 3.2: (a): Our capture system with a Kinect modified to run from a battery pack. (b) Our capture platform, with counterweights to damp camera movements.

bedrooms, bathrooms and kitchens. We also captured workplace and university campus settings. From the acquired video, we extracted frames every 2–3 seconds to give a dataset of 2347 unique frames, spread over 64 different indoor environments. The dataset is summarized in Table 3.1.

Scene class	Scenes	Frames	Labeled Frames
Bathroom	6	5588	76
Bedroom	17	22764	480
Bookstore	3	27173	784
Cafe	1	1933	48
Kitchen	10	12643	285
Living Room	13	19262	355
Office	14	19254	319
Total	64	108617	2347

Table 3.1: Statistics of captured sequences from NYU Depth V1

3.2.3 NYU Depth V2

The second version of the dataset was gathered from a wide range of commercial and residential buildings in three different US cities. To ensure diverse scene content and lack of similarity to other frames, each room was scanned from several locations including its corners and center. From each room filmed, we manually selected at most three frames carefuly chosen to ensure each image provided a distinct viewpoint of the room. A dense per-pixel labeling was obtained for each image using Amazon Mechanical Turk. The dataset contains 35,064 distinct objects, spanning 894 different classes. A breakdown of the type and frequency of scenes captures can be found in Table 3.2.

3.2.4 Pre-processing

Following alignment with the RGB webcam images, the depth maps still contain numerous artifacts. Most notable of these is a depth "shadow" on the left edges of objects. These regions are visible from the depth camera, but not reached by the infra-red laser projector pattern. Consequently their depth cannot be estimated, leaving a hole in the depth map. A similar issue arises with specular and low albedo surfaces. The internal depth estimation algorithm also produces numerous fleeting noise artifacts, particularly near edges.

Before extracting features for recognition, these artifacts must be removed. To do this, we filtered each image using the cross-bilateral filter of Paris [?]. Using the RGB image intensities, it guides the diffusion of the observed depth values into the missing shadow regions, respecting the edges in intensity. Examples of our in-painting method are shown in Figure 3.3.

3.2.5 Semantic and Instance Annotations

The selected frames from each dataset were uploaded to Amazon Mechanical Turk and manually annotated using the LabelMe interface [?]. The annotators were instructed to provide polygonal labels for every object instance in the scene such that no object was left unlabeled (see Fig. 3.1(c)). Furthermore, each polygon label was named with both a semantic class and an instance (e.g. cup1, cup2, cup3).

Scene class	Scenes	Frames	Labeled Frames
Basement	1	2,025	7
Bathroom	57	26,331	121
Bedroom	134	101,697	383
Bookstore	3	27,173	36
Cafe	1	1,933	5
Classroom	20	13,911	49
Computer Lab	2	1,904	6
Conference Room	2	1,426	5
Dinette	1	746	4
Dining Room	37	46,973	117
Excercise Room	1	904	3
Foyer	2	727	4
Furniture Store	2	11,395	27
Home Office	14	13,367	50
Home Storage	1	1,120	5
Indoor Balcony	1	330	2
Kitchen	53	66,635	225
Laundry Room	1	472	3
Living Room	77	94,693	221
Office	26	21,068	78
Office Kitchen	3	3,117	10
Playroom	7	6,725	31
Printer Room	1	582	3
Reception Room	4	4,556	17
Student Lounge	1	1,895	5
Study	8	3,935	25
Study Room	4	3,437	7
Total	464	459077	1449

Table 3.2: Statistics of captured sequences from NYU Depth V2.

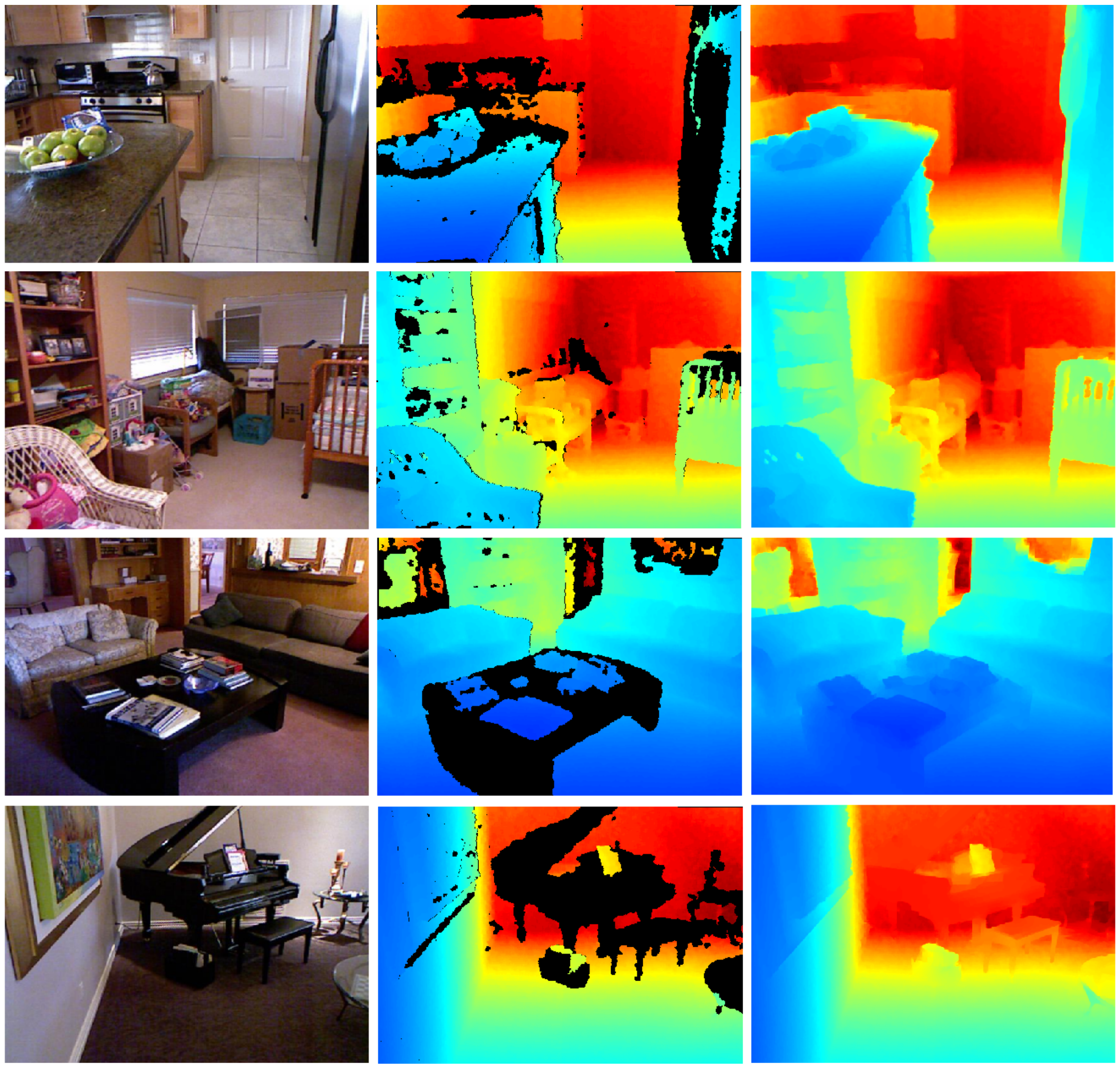


Figure 3.3: Examples of the results of the depth filling procedure. The first two rows demonstate scenes with small amounts of missing depth values which are filled plausibly. The coffee table's depth values are entirely missing in the third row but the filled depth still provides a plausible depth result. An example of poor depth filling is found in the fourth row where the piano and piano seat are completely missing depth values.

3.2.6 Support Annotations

Along with the semantic and instance annotations, we also provide support annotations for the NYU Depth V2 dataset. These annotations define a physical support relation type between two objects in the scene. Specifically, a support annotations consists of a set of 3-tuples: $[R_i, R_j, type]$ where R_i is the region ID of the supported object, R_j is the region ID of the supporting object and *type* indicates whether the support is from below (e.g. cup on a table) or from behind (e.g. picture on a wall).

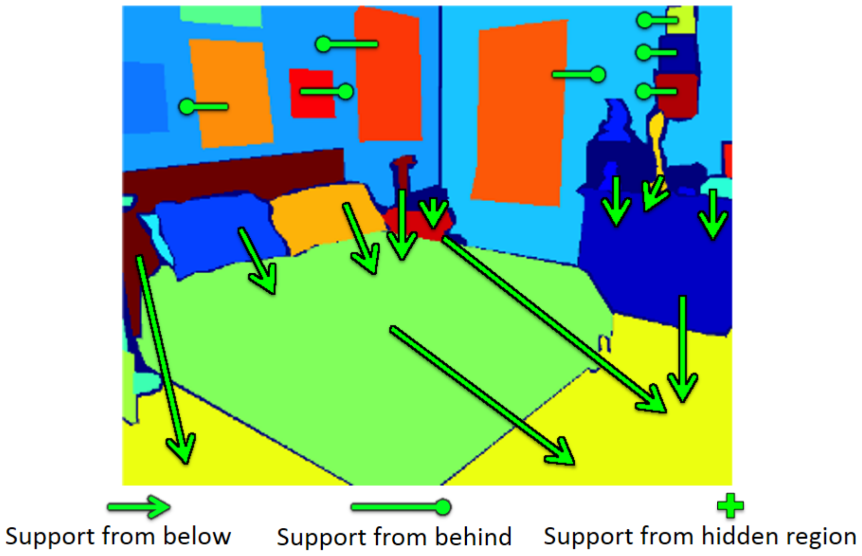


Figure 3.4: An example of the support labels. Each label indicates the supported region, the supporting region and the direction of support.

Chapter 4

Semantic Segmentation with 3D Priors

4.1 Introduction

Many efforts have been expended attempting to craft useful priors for semantic segmentation. While certain priors like co-occurance [?], [?], relative 2D locations [?] [?] or absolute 2D location [37] have been shown to improve semantic segmentation performance, they do not reason about the 3D locations of objects in the scene. We propose a new 3D location prior that captures the likelihood of objects appearing in different parts of the room relative to the room's boundaries. We show that such a prior improves semantic segmentation accuracy on a RGBD dataset and provides better performance than standard 2D priors.

4.2 3D Location Prior

Given a dataset of aligned RGB and depth frames, along with semantic labels for each pixel, our goal is to build a 3D location prior for each semantic class. By projecting each depth pixel out into 3D space, we can estimate the 3D position of each object in the scene. However, since each scene has a unique size and shape, we cannot trivially combine this position data across scenes. The design of our 3D prior is motivated by three empirical constraints (C1-C3):

C1: While the absolute depth of an individual object in a scene is arbitrary with respect to the location of the viewer, objects of different classes exhibit a high degree of regularity with respect to their *relative* depths in a room. Figure 4.1 highlights several examples. Walls are obviously at the farthest depths of rooms, televisions tend to placed just in front of them, and tables and beds are much more likely to occupy regions near the center of a scene.

C2: Many objects tend to be clustered near the edges of a room, such as walls, blinds, curtains, windows and pictures. Consequently, we want a non-linear scaling function that places increased emphasis on depths near the boundaries of a room.

C3: While objects show regularity in relative depth, any representation of an objects prior location must be somewhat invariant to the viewer moving around the room.

Our solution, therefore, is to normalize the depth of an object, using the depth of the room itself. We assume that in any given column[1] of the depth map, the point furthest from the camera is on the bounding hull of the room. Figure 4.2 demonstrates the reliability of the procedure in separating the boundaries of the room from objects of similar depth. We scale the depths of all points in a given column so that the furthest point has relative depth $\tilde{z} = 1$. This effectively maps each room to a lie within a cylinder of radius 1. This allows us to build

[1]This is assisted by the pitch and roll correction made in pre-processing.

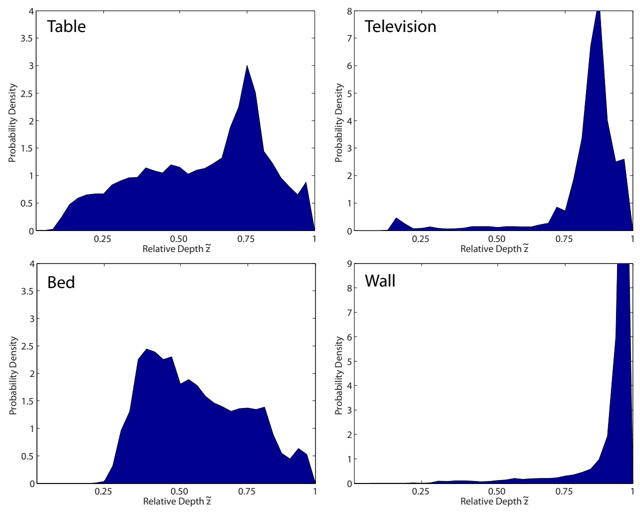


Figure 4.1: Relative depth histograms for table, television, bed and wall. As walls usually are on the boundary, they cluster near $\tilde{z} = 1$. Televisions lie just inside the room boundary, while tables and beds are found in the room interior.

the highly regular depth profiles for each class.

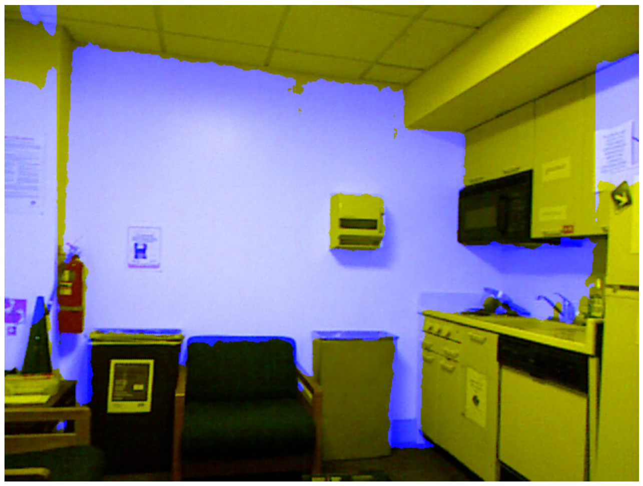


Figure 4.2: A demonstration of our scheme for finding the boundaries of the room. In this scene, the blue channel has been replaced by a binary mask, set to 1 if the depth of point is within 4% of the maximum depth within each column (and 0 otherwise). The walls of the room are cleanly identified, while segmenting objects of similar depth such as the fire extinguisher and towel dispenser. On the right, the cabinets and sink are correctly resolved as being in the room interior, rather on the boundary.

Within this normalized reference frame, we then build histograms from the 3D positions of objects in the training set. These 3D histograms are over $(h, \omega, \tilde{z})$ where h is the absolute scene height relative to the horizon (in meters); ω is angle about the vertical axis and $\tilde{z}$ is relative depth.

In addition, we use a non-linear binning for $\tilde{z}$. This produces very fine bins near the boundaries of the room, allowing us to discriminate between the many objects at the extremal edges of the room (satisfying C2), and coarse bins at the center of the room, giving us a degree of invariance to the camera's position (satisfying C3).

Similar to the 2D versions, the 3D histograms are normalized so that they sum to $1/C$ for each class (see Fig. 4.3 for examples). During testing, the extremal depth for each column in the depth map is found and the relative 3D coordinate of each point can be computed. Looking up these coordinates in the 3D histograms allows us to measure the probability of a class label given its relative 3D location: $P(y_i, i)$.

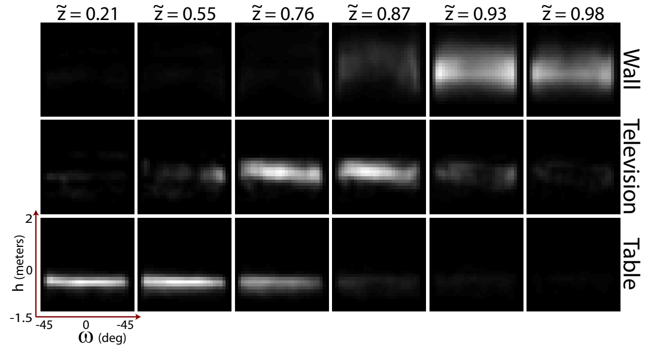


Figure 4.3: 3D location priors for wall, television and table. Each column shows a different relative depth $\tilde{z}$. For each subplot, the x-axis is orientation ω about the vertical and the y-axis is height h (relative to the horizon). The non-linear bin spacing in $\tilde{z}$ gives a more balanced distribution than linear spacing used in Fig. 4.1.

4.3 Model

We now describe the model used to measure baseline performance for the dataset. Like many other multi-class segmentation approaches [?, ?], we model the problem of semantic segmentation using a conditional random field (CRF). This formulation will allow us to explore a variety of different potential functions and permits efficient approximate inference [?].

Formally, given an image I with N pixels, we wish to find a labeling $\mathbf{y} = \{y_i \in C, i = 1 \dots N\}$ where C is the number of semantic classes. The energy of labeling is defined as:

$$E(\mathbf{y}) = \sum_{i \in N} \phi_i(x_i) + \sum_{i,j \in \mathcal{N}} \psi_{ij}(y_i, y_j) \tag{4.1}$$

where x_i are features for pixel i, $\mathcal{N}$ is the set of all neighboring pixels, ϕ_i is the unary cost function and ψ_{ij} is a pairwise cost function.

4.4 Unary Potentials

The unary potential function ϕ is the product of two components, a local appearance model and a location prior.

$$\phi(x_i, i|\theta) = -\log(\underbrace{P(y_i|x_i)}_{\text{Appearance}} \underbrace{P(y_i, i)}_{\text{Location}}) \tag{4.2}$$

4.4.1 Appearance Model

Our appearance model $P(y_i|x_i)$ is discriminatively trained using a range of different local descriptors. Rather than extract descriptors at every pixel in the image, we extract pixel-centric descriptors (f_i^P) from a sample of the pixels defined by a fixed lattice. Next, we compute a coarse superpixelation of the scene and the descriptors are aggregated over each superpixel to produce a superpixel descriptor f_s^S for each superpixel s in the scene. Because we seek a pixel-wise probability function, we define $P(y_i|x_i) = P(y_i|f_s^S)$ where pixel $i \subset s$.

Extracting local descriptors

Local pixel-centric descriptors are first extracted over a dense grid with stride 10. Each descriptor f_i^P is extracted from a patch centered at pixel i of size 40×40. We extract several types of local descriptors:

- **RGB-SIFT:** SIFT descriptors are extracted from the RGB image.
- **Depth-SIFT:** SIFT descriptors are extracted from the depth image. These capture both large magnitude gradients caused by depth discontinuities, as well as small gradients that reveal surface orientation. The depth image is normalized such that its smallest value is 0 and largest value is 1.
- **Depth-SPIN:** Spin image descriptors [?] are extracted from the depth map. To review, this is a descriptor designed for matching 3D point clouds and surfaces. Around each point in the depth image, a 2D histogram is built that counts nearby points as a function of radius and depth. The histogram is vectorized to form a descriptor.

We also propose several approach that combine information from the RGB and depth images:

- **RGBD-SIFT:** SIFT descriptors are extracted from both depth and RGB images. At each location, the 128D descriptors both images are concatenated to form a single 256D descriptor $x_i^{(s)}$ at each scale s.
- **RGB-SIFT/D-SPIN:** Spin image descriptors are extracted from the depth map, while SIFT is extracted from the RGB image.

Aggregating local descriptors

Given a collection of descriptors of a particular type, we learn a sparse coding dictionary [**?**] and represent each pixel-centric descriptor f_i^P by its sparse coded coefficients g_i^P. Next, the coefficents are aggregated over a course superpixelation (Section **??**) of the image to produce a superpixel descriptor:

$$f_s^S = \frac{\sum_{i \in s} f_i^P}{|| \sum_j \sum_{i \in s} f_{ij}^P ||} \tag{4.3}$$

where j indexes the dimensions of the descriptor. The denominator has the effect of ensuring that the sum of the dimensions of f_s^S is 1.

Inferring Superpixel Labels

Given a collection of superpixel descriptors f_s^S, we train a simple neural network with a single hidden layer of size $H(= 1000)$ and a soft-max output layer of dimension C, which is interpreted as $P(y_i|f_s^S)$. It has parameters θ (two weight matricies of sizes $(D+1) \times H$ and $(H+1) \times C$) which are learned using back-propagation and a cross-entropy loss function.

4.4.2 Location Prior

Our location prior $P(y_i, i)$ can take on two different forms. The first captures the 2D location of objects, similar to other context and segmentation approaches (e.g [**?**]). The second is a novel 3D location prior that leverages the depth information.

2D location priors: The 2D priors for each class are built by averaging over every training image's ground truth label map $\mathbf{y}^*$. To provide a degree of 2D spatial invariance, we then smooth the averaged map with an 11×11 Gaussian filter. To compute the actual prior distribution $P(y_i, i)$, we normalize each map so it sums to $1/C$, i.e. $\sum_i P(y_i, i) = 1/C$. Note that this assumes the prior class distribution to be uniform[2]. Figure Fig. 4.4 shows the resulting distributions for 4 classes.

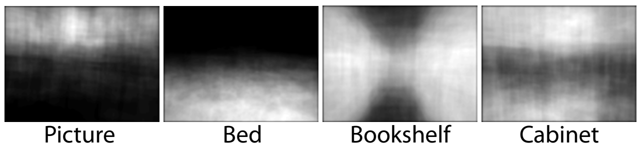


Figure 4.4: 2D location priors for select object classes.

4.5 Transition Potentials

The transition potentials term is a spatially varying Potts model [**?**]:

$$\psi_{ij}(y_i, y_j) = \eta(i, j)[y_i \neq y_j] \tag{4.4}$$

where η is a spatially varying transition term that incurs cost only when neighboring pixels differ in their label assignments. We explore several options using a potential of the form:

$$\eta(i, j) = \eta_0 \, e^{-\alpha \max(|I(i)-I(j)|-t, 0)} \tag{4.5}$$

where $|I(i) - I(j)|$ is gradient between adjacent pixels i, j in image I, t is a threshold and α and η_0 are scaling factors. We use $\eta_0 = 100$ for all the following methods:

- **None:** The baseline method is to keep $\eta(i, j) = 1$ for all i, j in the CRF. The smoothness of the labels $\mathbf{y}$ is then solely induced by the class transition potential ψ.

[2]In practice, if the true class frequencies are used, common classes would be overly dominant in the CRF output.

- **RGB Edges:** We use I_{RGB} in Eqn. 4.5, thus encouraging transitions at intensity edges in the RGB image. $\alpha = 40$ and $t = 0.04$.
- **Depth Edges:** We use I_{Depth} in Eqn. 4.5, with $\alpha = 30$ and $t = 0.1$. This encourages transitions at depth discontinuities.
- **RGB + Depth Edges:** We combine edges from both RGB and depth images, with $\eta(i, j) = \beta\eta_{\text{RGB}}(i, j) + (1 - \beta)\eta_{\text{Depth}}(i, j)$ and $\beta = 0.8$.
- **Super-Pixel Edges:** We only allow transitions on the boundaries defined by the super-pixels, so set $\eta(i, j) = 1$ on super-pixel boundaries and η_0 elsewhere.
- **Super-Pixel + RGB Edges:** As for **RGB-Edges** above, but now we multiply $|I(i) - I(j)|$ in Eqn. 4.5 by the binary super-pixel boundary mask.
- **Super-Pixel + Depth Edges:** As for **Depth-Edges** above, but now we apply the binary super-pixel boundary mask to $|I(i) - I(j)|$.

4.6 Experiments

We now evaluate our CRF-based model using the fully labeled set of 2347 frames. The annotations cover over 1000 classes, which we reduce using a Wordnet synonym/homonym structure to 12 common categories plus a generic background class (containing rare objects). We generated 10 different train/test splits, each of which divides the data into roughly 60% train and 40% test. The error metric used throughout is the mean diagonal of the confusion matrix, computed for per-pixel classification over the 13 classes on the test set.

Unary Appearance

We first use our dataset to compare the local appearance models listed in Section 4.4.1, with the results shown in Table 4.1. We show the performance of the unary potential (with no location prior) in isolation, as well as the full CRF (sans spatial transition potential). The first row in the table, which makes no use of depth information achieves 43.4% accuracy. We note that: (i) combining RGB and depth information gives a significant performance gain of ∼5%; (ii) the CRF model gives a gain of ∼2.5% and (iii) the SIFT-based descriptors outperform the SPIN-based ones.

Descriptor	Unary Only	CRF
RGB-SIFT (S_{RGB})	40.9 ± 3.0	43.4 ± 3.3
RGB-SIFT (S_{RGBD})	40.4 ± 2.8	43.3 ± 3.1
Depth-SIFT	39.3 ± 2.2	41.1 ± 2.5
Depth-SPIN	34.0 ± 2.8	35.8 ± 3.1
RGBD-SIFT	45.8 ± 2.6	**48.1 ± 2.9**
RGB-SIFT/D-SPIN	42.5 ± 1.5	45.0 ± 1.6

Table 4.1: A comparison of unary appearance terms using mean per-pixel classification accuracy. All methods in this table compute appearance using S_{RGBD} super-pixels apart from the 1st row.

Unary Location

We now investigate the effect of location priors in our model. Table 4.2 compares the effect of the 2D and 3D location priors detailed in Section 4.4.2. All methods used S_{RGBD} super-pixels and no spatial transition potentials in the CRF. The 2D priors give a modest boost of 2.8% when used in the CRF model. However, by contrast, our novel 3D priors give a gain of 10.3%. We also tried bulding a prior using absolute 3D locations (3D priors (abs) in Table 4.2), which did not use our depth normalization scheme. This performed very poorly, demonstrating the value of our novel prior using relative depth. The overall performance gains for each of the 13 classes, relative to the RGB-SIFT (S_{RGB}) model (1st row of Table 4.1, which makes no use of depth information), is shown in Fig. 4.5. Using RGBD-SIFT and 3D priors, gains of over to 29% are achieved for some classes.

In Fig. 4.6 we show six example images, each with label maps output by the RGB-SIFT+2D

Descriptor	Unary Only	CRF
RGB-SIFT	40.9 ± 3.0	43.4 ± 3.3
RGB-SIFT+2D Priors	45.7 ± 2.8	46.2 ± 2.8
RGBD-SIFT	45.8 ± 2.6	48.1 ± 2.9
RGBD-SIFT+2D Priors	49.2 ± 2.2	49.9 ± 2.3
RGBD-SIFT+3D Priors	53.0 ± 2.2	**53.7 ± 2.3**
RGBD-SIFT+3D Priors (abs)	38.7 ± 3.2	39.9 ± 3.5

Table 4.2: A comparison of location priors.

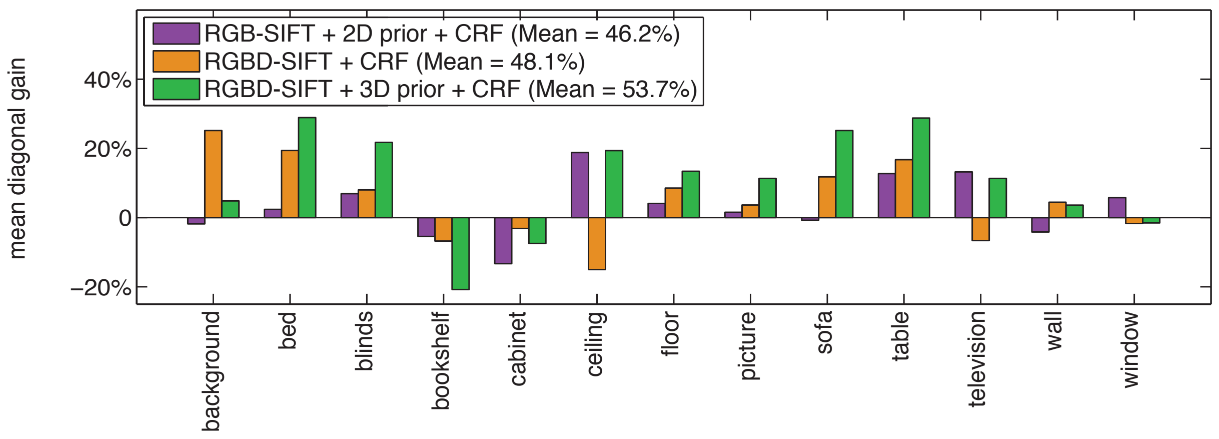


Figure 4.5: The per-class improvement over RGB-SIFT (S_{RGB}) + CRF (which uses no depth information), for models that add depth and 3D location priors. The large gains benefits of adding depth information to both the appearance and location potentials.

Priors and RGBD-SIFT+3D Priors models. The RGB model (2nd column) makes mistakes which are implausible based on the object's 3D location. The RGBD and 3D prior model gives a more powerful spatial context, with its label map (3rd column) being close to that of ground truth (4th column).

Spatial Transition Potentials

Table 4.3 explores different forms for the spatial transition potential. All methods use unary potentials based on S_{RGBD} super-pixels and RGBD-SIFT + 3D prior. The results show that using an RGB-based spatial transition gives a performance gain of 2.8%. Using the depth or super-pixel constraints does not give a significant gain however.

Type	CRF
None	53.7 ± 2.3
RGB Edges	56.6 ± 2.9
Depth Edges	53.9 ± 3.1
RGB + Depth Edges	56.5 ± 2.9
Super-Pixel Edges	54.7 ± 2.4
Super-Pixel + RGB Edges	56.4 ± 3.0
Super-Pixel + Depth Edges	53.0 ± 3.0

Table 4.3: A comparison of spatial transition potentials. Mean per-pixel classification accuracy (in %).

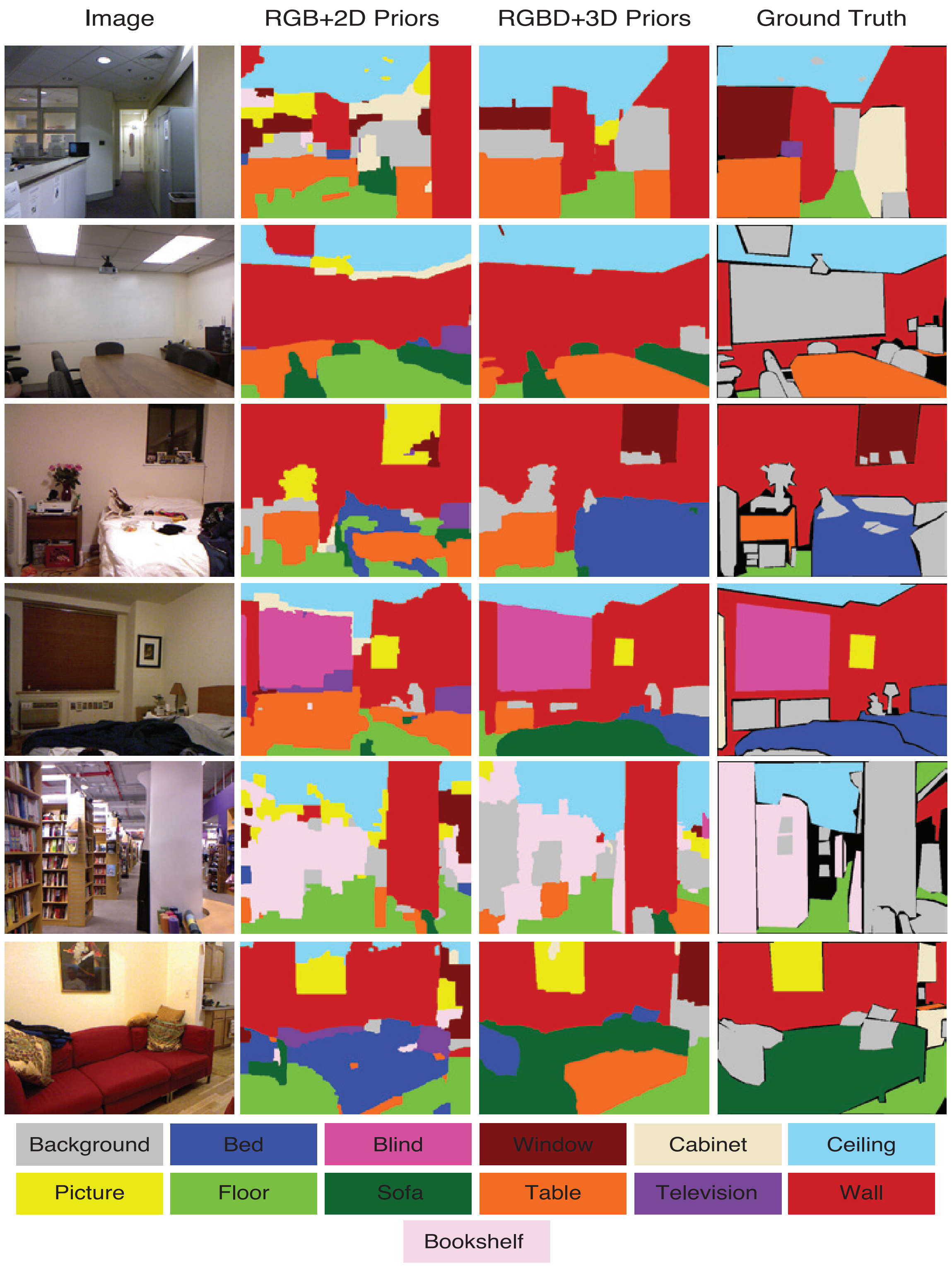


Figure 4.6: Six example scenes, along with outputs from 2 different models. See text for details. This figure is best viewed in color.

Chapter 5

Tree-Based Instance Segmentation

Semantic segmentation models have made great strides in the last few years. From early efforts to densely label scenes [37], numerous advances including reasoning with multiple segmentations [26], higher-order label constraints [22] and fast inference mechanisms [27] have advanced the state of the art considerably. One limitation in all of these methods, however, is their inability to differentiate between different *instances* of the same class. This limitation is illustrated in Figure 1.1.

The ability to differentiate between instances of the same class is important for a variety of tasks. In image search, one needs to understanding instance information to properly understand count-based searches: "three cars waiting at a light" should retrieve different results from "a single car waiting at the light". Robots that interact with real world environments need to be able to understand instance information as well. For example, when grasping a bottle from a collection of bottles, the pixels that represent the target bottle instance must be identified as being different from those around it. Furthermore, when picking up a box a robot needs to distinguish between a single box and a stack of them. Finally, being able to correctly infer object instances drastically improves performance on high level scene reasoning tasks such as support inference [32] or inferring object extent [14].

Unfortunately, searching over the space of all semantic and instance segmentations for a given image is computationally infeasible and we must limit the solution space to make the problem tractable. Like previous work in semantic segmentation [26] [31] [27] [11], we make use of hierarchical segmentations, referred to as segmentation trees, to limit the search space to a more manageable size.

We explore two different inference schemes for learning to cut segmentation trees to produce instance segmentations: a greedy model and a global model. The greedy model is trained to merge regions together that belong to the same instance. The final (coarsest) level of the segmentation tree represents the results of segmentation inference. While the greedy inference scheme is simple to train, it unfortunately limits our ability to (a) undo mistakes and (b) reason about both segments and semantics at the same time. Conversely, our global inference model is more difficult to train but it allows us to reason *across* levels of the segmentation tree, simultaneously predicting both the best instance segmentation and the appropriate semantic class for each region.

5.1 Segmentation Trees

A set of regions or segments $S = \{s_1, \dots, s_R\}$ forms a valid segmentation tree $\mathcal{T} = \{S, P\}$ for an image $\mathcal{I}$ if it satisfies the following constraints:

Completeness: Every pixel $\mathcal{I}_i$ is contained in at least one region of S.

Tree Structure: Each region s_i has at most one parent: $P(s_i) \in \{\emptyset, s_j\}, j \neq i$

Strict Nesting: If $P(s_i) = s_j$, then the pixels in s_i form a strict subset of s_j

A cut $\mathcal{T}(A)$ of the tree selects a subset $S_A \subset S$ of segments that form a *planar segmentation*, a map $M : \mathcal{I}_i \mapsto \mathbb{Z}$ from each pixel $\mathcal{I}_i$ to exactly one region. The goal of this work is to take as input a segmentation tree and cut it such that the resulting planar segmentation is composed of a set of regions, each of which corresponds to a single object instance in the input image.

We will make use of two types of segmentation trees: standard segmentation trees and biased segmentation trees.

5.1.1 Standard Segmentation Trees

We use the term *standard segmentation tree* to refer to a hierarchical segmentation created by iteratively merging similar regions together [34] [17] [3]. To summarize, given an initial over-segmentation, adjacent regions are iteratively merged together to produce ever coarser segmentations of the scene. The decision of whether or not to merge a pair of regions is typically based on a classifier or series of classifiers that predict whether a pixel or edge between regions represents an object boundary based on local appearance cues. The tree structure is consequently defined as follows. If two regions r_i and r_j were merged to produce a new region r_k, then r_i and r_j are said to be the children of r_k. If regions r_i and r_k were created during the same stage, they are said to have the same height or belong to the same level of the tree.

5.1.2 Biased Segmentation Trees

A biased segmentation tree is a tree that contains, as a possible cut a pre-specified planar segmentation. This can be produced by (a) ensuring that the initial segmentation respects the biasing segmentation and (b) never merging two regions if they fall into different regions

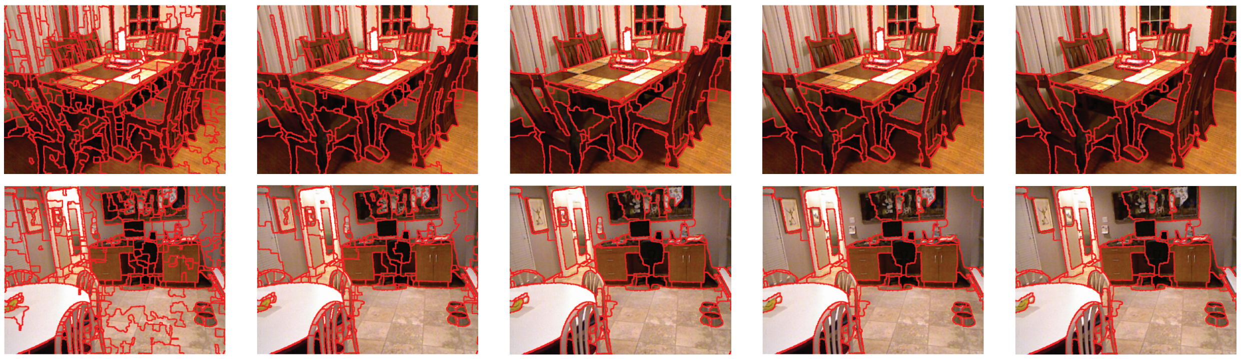


Figure 5.1: **Segmentation Examples.** We show two examples of hierarchical segmentation. Starting with roughly 1500 superpixels (not shown), our algorithm iteratively merges regions based on the likelihood of two regions belonging to the same object instance. For the final segmentation, no two regions have greater than 50% chance of being part of the same object.

in the biasing segmentation. More formally, if a region r_i is a subset of biasing region b_1 and region r_j is a subset of biasing region b_2, then r_i and r_j can never be merged. However, if r_i and r_j are both subsets of b_1 or both subsets of b_2, then they are merged according to the logic of the segmentation tree model.

5.2 Greedy Tree Training and Inference

For our greedy tree training scheme, we adapt the agglomerative segmentation model of Hoiem *et al.* [17] to RGBD images. We begin with an initial fine oversegmentation of the scene. Over a series of 5 stages, regions are iteratively merged to form coarser and coarser segmentations culminating in a final segmentation of the scene. In each stage, a separate classifier is trained to predict whether or not a pair of adjacent regions represent the same object instance and thus whether the pair should be merged. Following training, regions are merged in order of decreasing confidence until a minimum confidence threshold is reached.

5.2.1 Initializing Segmentation

To create an initial set of regions, we use the watershed algorithm applied to Pb boundaries, as first suggested by Arbeleaz [**?**]. After fitting a series of 3D planes (Section 8.2), we force this oversegmentation to be consistent with the detected planes. This primarily helps to avoid regions that span wall boundaries with faint intensity edges. We also experimented with incorporating edges from depth or surface orientation maps, but found them unhelpful, mostly because discontinuities in depth or surface orientation are usually manifest as intensity discontinuities. Our initial oversegmentation typically produces around 2000 regions, such that very few regions overlap more than one object instance.

5.2.2 Region Merging Classifier

At each stage, we train a classifier to predict whether or not neighboring regions belong to the same object instance. In particular, we train a boosted decision tree classifier as $\mathrm{P}(y_i \neq y_j | \mathbf{x}_{ij}^s)$, where y_i is the instance label of the i^{th} region and $\mathbf{x}_{ij}^s$ are paired region features. The classifier is trained using similar RGB and position features to Hoiem et al. [17], but the "geometric context" features are replaced with ones using more reliable depth-based cues. These proposed 3D features encode regions corresponding to different planes or having different surface orientations or depth differences are likely to belong to different objects. Both types of features are important: 3D features help differentiate between texture and objects edges, and standard 2D features are crucial for nearby or touching objects.

If more than two boundaries separate a pair of neighboring regions (due to earlier merging), the composite strength is computed as the maximum of the individual boundary strengths. As the regions grow, their features change and new features become useful. For this reason, we estimate new boundary costs at several thresholds. In our implementation, separate trained classifiers predict boundary strength for the initial superpixels and then at 5 iterative stages. Merging at each stage stops when no boundaries remain whose merge probability is less than some threshold (0.8 in our experiments). See Figure 5.2 for two examples of this procedure.

5.3 Global Tree Training and Inference

By inferring the best possible cut of a segmentation tree across all levels of the hierarchy, we have the ability to undo mistakes made in early stages of merging. Unfortunately, two problems complicate global training: defining the ground truth and selecting the appropriate loss function.

5.3.1 The Ground Truth Mapping Problem:

When using a reduced search space, such as one provided by a given hierarchical segmentation, it is extremely rare that the exact ground truth regions are among the set of bottom-up *proposed* regions, due to mistakes made at detecting object boundaries. Therefore, during training, we must be able to map the human-provided labels to a set of surrogate labels, defined on the set of proposed regions.

In semantic segmentation, these surrogate labels are typically produced by simply assigning each proposed region the semantic label of the ground truth region that it overlaps with the most. This heuristic is easy to compute, can be performed for each region independently, and achieves good empirical results on semantic segmentation.

Obtaining these surrogate labels is problematic for semantic-instance segmentation. The constraint that the regions must not non-overlap means the best possible subset of regions

Segmentation Feature Descriptions	**Dims**
Boundary	**8**
B1. Strength: average Pb value	1
B2. Length: perimeter of each region; (boundary length) / (smaller perimeter)	3
B3. Smoothness: length / (L1 endpoint distance)	1
B4. Continuity: minimum angle difference at each junction	2
B5. Long-range: number of chained boundaries	1
Region	**19**
R1. Color: diff in RGB histogram entropy for separate regions vs merged region	1
R2. Color: diff in RGB mean/std near region borders and within region interiors; for each channel separately and overall RMS	16
R3. Area: fraction of image occupied by each region	2
3D Surface	**16**
S1. Normals: difference in surface normal of planes fit to each region	1
S2. Normals: diff in hist of polar coord of normals within each region (4 inclinations, 4 azimuths)	8
S3. Planes: difference in plane labels	1
S4. Planar fit: mean/med/max diff of pt positions to other region's plane	3
S5. Planar fit: mean/med/max diff of pt normals to other region's plane	3
3D Position/Volume	**14**
P1. Volume: intersection / smaller volume; ratio of volumes of the regions	2
P2. Volume distance: min dist of 3D bounding boxes; 3D centroid dist	2
P3. Footprint overlap: intersection / smaller XY bbox area	1
P4. Footprint distance: min dist of XY bboxes; XY centroid dist	2
P5. Shape: ratio of height to footprint for each region	2
P6. Height: diff in min and max heights above ground	2
P7. Height: diff in min and max heights above ground	2
P8. Density: diff in point densities (num region pixels / volume)	1

Table 5.1: **Agglomerative Segmentation features.** Note: differences are all absolute differences. Volumes and footprints are computed based on axis-aligned 3D bounding box fits to 3D points of the region.

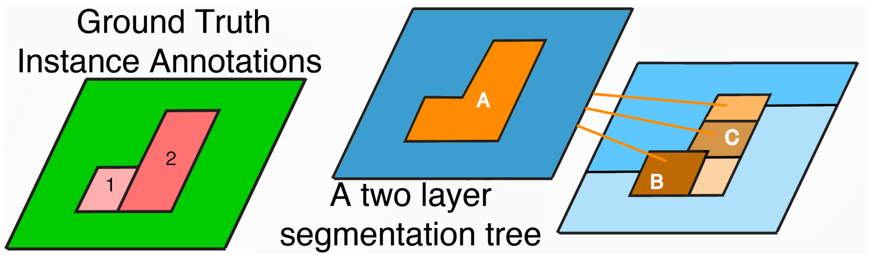


Figure 5.2: Computing the best possible set of instances that overlap with the ground truth cannot be computed independently per ground truth region. For example, ground truth region 2 best overlaps with proposed region A and ground truth region 1 best overlaps with proposed region B. But both proposed regions A and B cannot be selected at the same time because they overlap.

cannot be computed independently as the inclusion of one region may exclude the inclusion of another (see Figure 5.2). Therefore, computing the 'best possible' instance segmentation with respect to a segmentation tree is an optimization problem in its own right.

Identifying a Good Loss Function:

In semantic segmentation, it is easy to penalize mistakes: a region is either assigned the correct or incorrect label. In our setting, we require a more continous measure, since an inferred region might not exactly match the 'best' region, but it might be extremely close (e.g. differing by a single pixel). Although previous continuous higher order loss functions exist (e.g. [40]) , these loss functions cannot handle the multiple ground truth regions encountered in complex scenes.

5.4 Creating the Segmentation Trees

When training globally, we make use of both standard and biased segmentation trees. However, we use a different methods for creating the trees than those outlined in Section 5.2.

5.4.1 Standard Segmentation Trees

Given an image with N pixels, we begin by producing a graph with N nodes where each node corresponds to a pixel in the image. For each pair of neighboring pixels, an edge is added between the corresponding graph nodes. The edge weights will indicate the probability that each pair of neighboring pixels represents a boundary between object instances. As in [3], the edge weights are computed by first extracting gPb features [29] and calculating the Ultrametric Contour Map (UCM). To create a segmentation, edges lower than some threshold are removed and the induced regions are the connected components of the resulting graph. This process is repeated with various thresholds to create finer or coarser regions. The unique superset of regions produced by the various thresholded region maps form a tree $\mathcal{T}$ such that each region is represented as a node and any pair of region r_i, r_j are the children of region r_k if r_i and r_j were merged to produce r_k.

5.4.2 Biased Segmentation Trees

To create our biased segmentation trees, we first threshold the UCM to obtain a base set of regions. Next, we split these regions further by taking the intersection of every ground truth region with the segmented regions. To create the segmentation tree, we use the same algorithm as in Section 5.4.1, except that the nodes correspond to the regions and edge weights for neighboring regions are given by the average gPb values for pixels near their boundary. Any boundary aligned with a ground truth edge is given a weight of 0. Finally, a coarser set of regions is obtained by repeating this process starting from the ground truth regions.

5.5 Cutting Instance Segmentation Trees

Given an image and a segmentation tree, our goal is find the best cut of the tree such that each of the resulting regions corresponds to a single *instance* of an object and is assigned the appropriate semantic class.

Let a cut of the tree be represented by $\{\mathbf{A} : A_i \in \{0,1\}, i = 1..R\}$, a vector indicating whether or not each of the R regions in the tree are selected. Let $\{\mathbf{C} : C_i \in \{1..K\}, i = 1..R\}$ be a vector indicating the semantic class (out of K) of each region. Finally let $y = \{\mathbf{A}, \mathbf{C}\}$ be the combined output of semantic labels and region instances.

5.5.1 Model

We frame tree cutting as a structured prediction problem by optimizing over the space $\mathcal{Y}$ of region selections and semantic class assignments for a given segmentation tree. Formally, we predict using

$$y^* = \arg\max_{y \in \mathcal{Y}} w^T \phi(x, y) \tag{5.1}$$

where x represents the input image, y encapsulates both the region selection vector A and class assignments C, and w represents a trained weight vector. ϕ is a feature function on the joint space of x and y such that $w^T \phi(x, y)$ can be interpreted as measuring the compatibility of x and y. $w^T \phi(x, y)$ can be decomposed as follows:

$$w^T \phi(x, y) = w_{\text{reg}}^T \phi_{\text{reg}}(x, y) + w_{\text{sem}}^T \phi_{\text{sem}}(x, y) + w_{\text{pair}}^T \phi_{\text{pair}}(x, y) + \phi_{\text{tree}}(y) \tag{5.2}$$

The **generic region potentials** encode class-agnostic appearance features of selected regions:

$$\phi_{\text{reg}}(x,y) = \sum_{i=1}^{R} f_i^{\text{reg}}[A_i = 1] \tag{5.3}$$

where f_i^{reg} are region features extracted from region i. The **semantic compatibility** features capture class-specific features of each region:

$$\phi_{\text{sem}}(x,y) = \sum_{k=1}^{K} \sum_{i=1}^{R} f_i^{\text{sem}}[A_i = 1 \wedge C_i = k] \tag{5.4}$$

where k is the semantic class and f_i^{sem} are semantic features extracted from region i. The **pairwise** features $\phi_{\text{pair}}(x,y)$ are given by

$$\sum_{ij \in \mathcal{E}} f_{ij}^{\text{pair}}[A_i = 1 \wedge A_j = 1] \tag{5.5}$$

where $\mathcal{E}$ is the set of all adjacent regions and f_{ij}^{pair} are pairwise features extracted along the boundary between regions i and j. Finally, the **tree-consistency function** $\phi_{\text{tree}}(y)$ enforces the Completeness condition of segmentation trees (Section 5.1): every pixel must be explained by exactly one region. To satisfy this condition, the potential ensures that exactly one region along every path from the leaf nodes to the root node of the tree is selected:

$$\phi_{\text{tree}}(y) = \sum_{\gamma \in \Gamma} -\infty\big[1 \neq \sum_{i \in \gamma} 1[y.A_i = 1]\big] \tag{5.6}$$

where γ is a single path in the tree: a set of regions that includes a leaf and all of its ancestors. Γ is the set of all paths in the tree from the leaves to the root.

5.5.2 Inference

We first show how the inference task, $\arg\max_{y\in\mathcal{Y}} w^T\phi(x,y)$, can be formulated as an integer linear program. To do so, we introduce binary variable matrices to encode the different states of $\mathbf{A}$ and $\mathbf{C}$ and auxiliary vector variables to encode the pairwise states. Let $a \in \mathbb{B}^{R\times 2}$ encode the states of $\mathbf{A}$ such that $a_{i,0} = 0$ indicates that region i is not selected and $a_{i,1} = 1$ indicates that region i is selected. Let $c_{i,k}$ encode the states of $\mathbf{C}$ such that $c_{i,k} = 1$ if $\mathbf{C}_i = k$ and 0 otherwise. Finally, let $p \in \mathbb{B}^{E\times 1}$ be a vector which encodes whether any neighboring pair of regions are both selected where E is the number of neighboring regions.

Our integer linear program can be stated as:

$$\arg\max_{a,c,p} \quad \sum_{i=1}^{R}\theta^{\text{r}}_i a_{i,1} + \sum_{i=1}^{R}\sum_{k=1}^{K}\theta^{\text{s}}_{i,k}c_{i,k} + \sum_{ij\in\mathcal{E}}\theta^{\text{p}}_{ij}p_{ij} \tag{5.7}$$

$$\text{s.t.} \quad a_{i,0} + a_{i,1} = 1 \qquad \forall\, i \in R \tag{5.8}$$

$$a_{i,0} = c_{i,0} \qquad \forall\, i \in R \tag{5.9}$$

$$\sum_{k=0}^{K} c_{i,k} = 1 \qquad \forall\, i \in R \tag{5.10}$$

$$\sum_{i\in\gamma} a_{i,1} = 1 \qquad \forall\, \gamma \in \Gamma \tag{5.11}$$

$$p_{ij} \le a_{i,1}, \quad p_{ij} \le a_{j,1}, \quad a_{i,1} + a_{j,1} - p_{ij} \le 1 \qquad \forall\, i,j \in \mathcal{E} \tag{5.12}$$

where $\theta^{\text{r}}_i = w^T_{\text{reg}} f^{\text{reg}}_i$ is the cost of selecting region i, $\theta^{\text{s}}_{i,j} = w^T_{\text{sem:k}} f^{\text{sem}}_i$ is the cost of assigning region i to semantic class k and $\theta^{\text{p}}_{ij} = w^T_{\text{pair}} f^{\text{pair}}_{ij}$ is the cost of selecting neighboring regions i and j. Equation 5.8 ensures that each region is either active or inactive. Equation 5.9 ensures that if a region is inactive, then it cannot be assigned a semantic class. Equation 5.10 constrains each region to be assigned to at most a single semantic class. Equation 5.11 ensures that exactly one region in each of the paths of the tree (from each leaf to most coarse node) is active. Equation 5.12 ensures that the auxiliary pairwise variable p_{ij} is on if and only if both regions i and j are selected.

5.5.3 Learning

Let $D = \{(x^{(1)}, y^{(1)}), ...(x^{(N)}, y^{(N)})\}$ be a dataset of pairs of images and labels where $y^{(i)} = \{\mathbf{A}, \mathbf{C}\}$ comprises the best assignment of segments from the pool and semantic class labels for image i. We use a Structured SVM [42] formulation with margin re-scaling to learn weight vector w:

$$\begin{aligned} &\min_{w,\, \xi \geq 0} \quad \frac{1}{2} w^T w + \frac{\lambda}{N} \sum_{i=1}^{N} \xi_i \\ &\text{s.t.} \quad w \cdot [\phi(x^{(n)}, y^{(n)}) - \phi(x^{(n)}, y)] \geq \Delta(y, y^{(n)}) - \xi_n \quad \forall n,\ y \in \mathcal{Y} \end{aligned} \tag{5.13}$$

where ξ_i are slack variables for each of the training samples $1..N$, and λ is a regularization parameter. We define the loss function used in the following section.

5.6 Coverage Loss

Let a ground truth set of regions for an image be represented as $G = \{r_1^G, ...r_{|G|}^G\}$ and let $S = \{r_1^S, ...r_{|S|}^S\}$ be a proposed set of regions, respectively. We measure the overlap of two regions using the intersection over union score for a pair of regions r_i and r_j:

$$\text{Overlap}(r_i, r_j) = \frac{r_i \cap r_j}{r_i \cup r_j} \tag{5.14}$$

The weighted coverage score [17] measures the similarity between two segmentations:

$$\text{Coverage}_{\text{weighted}}(G,S) = \frac{1}{|\mathcal{I}|}\sum_{g=1}^{|G|} |r_g^G| \max_{s=1..|S|} \text{Overlap}(r_g^G, r_s^S). \tag{5.15}$$

where $|G|$ is the number of regions in G and $|S|$ are the number of regions in S, $|\mathcal{I}|$ is the number of pixels in the entire image and $|r_g^G|$ is the number of pixels in ground truth region r_g^G. We define the Coverage Loss function to be the amount of coverage score unattained by a particular segmentation:

$$\Delta_{\text{W}_1}(y^{(i)}, \bar{y}) = \frac{1}{|\mathcal{I}_i|}\sum_{g=1}^{|G_i|} |r_g^{(i)}| \big[1 - \max_{s:\bar{A}_s=1} \text{Overlap}(r_g^{(i)}, \bar{r}_s)\big] \tag{5.16}$$

where $y^{(i)}$ is the ground truth label for image i and $\bar{y}$ is a predicted cut and semantic assignment for the image. The sum is computed over each of the $|G_i|$ ground truth regions in $y^{(i)}$ and the max is over each of the predicted selected regions in $\bar{A}$. $|\mathcal{I}_i|$ is the total number of pixels in the image and $|r_g^{(i)}|$ is the number of pixels in the g'th ground truth region of image i.

5.6.1 Integer Program Formulation with Loss Augmentation

During training, most structured learning algorithms [20] [25] need to solve the *loss augmented inference* problem, in which we seek to obtain a high energy prediction that also has high loss:

$$y^* = \arg\max_{\bar{y}\in\mathcal{Y}} \Delta(y^{(i)}, \bar{y}) + w^T\phi(x,\bar{y}) \tag{5.17}$$

To solve the loss augmented inference problem, we introduce an additional auxiliary matrix $o \in \mathbb{B}^{G\times R}$ where G represents the number of ground truth regions and R represents the number of regions in the tree. The variable o_{gj} will be 1 if region r_j is the argmax in (5.16) for the ground truth region g, and 0 otherwise. To ensure this, we add an additional set of constraints:

$$o_{gi} \leq a_{i,1} \qquad \forall\, g \in G,\ \forall i \in R \tag{5.18}$$

$$\sum_{i=1}^{R} o_{gi} = 1 \qquad \forall\, g \in G \tag{5.19}$$

$$o_{gi} + a_{j,1} \leq 1 \qquad \forall g \in G, i, j \in R \text{ s.t. } \text{Overlap}(s_g, s_j) > \text{Overlap}(s_g, s_i) \tag{5.20}$$

Equation (5.18) ensures that a prediction region i can only be considered the maximally overlapping region with ground truth region g if it is a selected region. Equation (5.19) ensures that every ground truth region is assigned exactly 1 overlap region. Finally, Equation (5.20) ensures that prediction region i can only be assigned the maximal region of g if and only if no other region j that has greater overlap with g is active.

The ILP objective is then altered to take into account the coverage loss:

$$\arg\max_{a,c,p,o} \quad \sum_{i=1}^{R} \theta_i^{\mathrm{r}} a_{i,1} + \sum_{i=1}^{R} \sum_{k=1}^{K} \theta_{i,k}^{\mathrm{s}} c_{i,k} + \sum_{ij \in \mathcal{E}} \theta_{ij}^{p} p_{ij} + \sum_{g=1}^{G} \sum_{i=1}^{R} \theta_{gi}^{\mathrm{o}} o_{gi} \tag{5.21}$$

where θ_{gi}^{o} encodes the loss incurred if region i is selected and region i is the region with the highest overlap with ground truth region g among all other ground truth regions.

5.7 Solving the Ground Truth Mapping Problem

Because the ground truth semantic and instance annotations are defined as a set of regions which are not among our segmentation tree-produced regions, we must map the ground truth annotations onto our segmented regions in order to learn. To do so, we build upon the ILP formulation described in the previous section.

Formally, given a ground truth set of instance regions $G = \{r^g_1, ...r^g_{R_G}\}$ and a proposed tree of regions $S = \{r_1, ...r_{R_P}\}$, the cut of the tree that maximizes the weighted coverage score is given by the following ILP:

$$\arg\min_{a,o} \quad \sum_{g=1}^{G}\sum_{i=1}^{R} \theta^{\mathrm{o}}_{gi} o_{gi}$$

subject to $a_{i,0} + a_{i,1} = 1 \; \forall i \in R$, $\sum_{i \in \gamma} a_{i,1} = 1 \; \forall \gamma \in \Gamma$, and Equations (5.18),(5.19), and (5.20).

5.7.1 Learning with Surrogate Labels

When the segmentation trees contain the ground truth regions as a possible cut, the minimal value of the Coverage Loss (Equation 5.16) is 0. However, in practice, the segmentation trees that we use only sample a small number of regions and it is rare that the ground truth regions are among them. Consequently, we must learn to predict a set of surrogate labels $\{z^{(1)}, ..., z^{(N)}\}$ instead.

We modify the loss used in training to ensure that the magnitude of the surrogate loss of a prediction $\bar{y}_i$ is defined relative to the best possible cut $z^{(i)}$:

$$\Delta_{\mathrm{W}_2}(z^{(i)}, \bar{y}) = \Delta_{\mathrm{W}_1}(y^{(i)}, \bar{y}) - \Delta_{\mathrm{W}_1}(y^{(i)}, z^{(i)}) \tag{5.22}$$

Note that the first term in the loss can be pre-computed and simply serves the purpose of modifying the scale so that getting the notation of margin is equivalent for all data points and is with respect to the best possible cut in a given segment tree. It should be clear than when $y^{(i)} = z^{(i)}$, that $\Delta_{\mathrm{W}_2} = \Delta_{\mathrm{W}_1}$.

5.8 Convolutional Network Features for Dense Segmentation

While Convolutional Neural Networks (CNNs) have show impressive performance in Classification [23] and Detection [36] tasks, it has not yet been demonstrated that CNN features improves dense segmentation performance. Recently [13] showed how to use a pretrained convolutional network to improve the ranking of foreground/background segmentations on the PASCAL VOC dataset. Because most of the segmented objects in the PASCAL VOC dataset occupy such a large part of the scene, it is not obvious that a similar scheme will succeed on densely labeled scenes where many objects are so small that they require context, and not local shape information, to identify them. Furthermore, because CNNs are generally trained on RGB data (no RGBD data exists with enough labeled examples to properly train these deep models) it is unclear whether CNN features provide an additionl performance boost when combined with depth features. To investigate these unknowns, we evaluted RGB+D features, CNN features and the combination of the two. To generate our CNN features, we use the model from [23] extract the CNN features on sub-windows that tightly surround a given arbitrarily shaped region. Because a particular subwindow of an image may contain multiple objects, there is an inherent ambiguity in the use of subwindows for computing CNN features. To deal with this ambiguity, we experimented with three types of masking operations performed on each subwindow before the CNN features were computed. Firstly, we perform no masking (Normal Windows). Secondly, we blur the subwindow (Mask Blurred Windows) with a blur kernel whose radius increases with respect to the euclidean distance from the region mask. This produces a subwindow that appears *focused* on the object itself. Finally, we use the masking operation from [13] (Masked Windows) in which any pixels falling outside the mask are set to the image means so that the background regions have zero value after mean-subtraction.

While we experimented with using the final pooled convolutional layer and the two fully connected layers, we did not observe a major difference in performance when using any of these three. To reduce the capacity of the model, we used the first fully connected hidden layer, which has fewer dimensions that the pooled layer, as the CNN feature vector for a given region.

We additionally experiment with a superset of features from [15] and [32] as well as compare the convolutional network features to Sparse Coded SIFT features from [38]. The pairwise region features are likewise a superset of pairwise region and boundary features from [15] [32].

5.9 Experiments

To evaluate our instance-segmentation scheme, we use the NYU Depth V2 [32] dataset. While datasets like Pascal VOC [10], Berkeley [5] and MSRC [37] are frequently used to evaluate segmentation tasks, MSRC does not provide instance labels and the Berkeley dataset provides neither semantic nor instance labels. Pascal only contains a few segmented objects per scene, mostly at the same scale. Conversely the NYU Depth dataset has densely labeled scenes with instance masks for objects of highly varying size and shape.

While the original NYU V2 dataset has over 800 semantic classes, we mapped these each of these classes to one of 20 classes: cabinet, bed, table, seating, curtain, picture, window, pillow, books, television person, sink, shelves, cloth, furniture, wall, ceiling, floor, prop and structure. We use the same classes for evaluating both the CNN features and the semantic instance segmentation.

5.9.1 Evaluating Greedy Inference

To evaluate our greedy inference scheme, we analyze several properties of the inferred trees. Firstly, we compare the results of using RGB only, Depth only and combined RGBD features. As demonstrated in Figure 5.3, both RGB-only and Depth-only features perform similarly

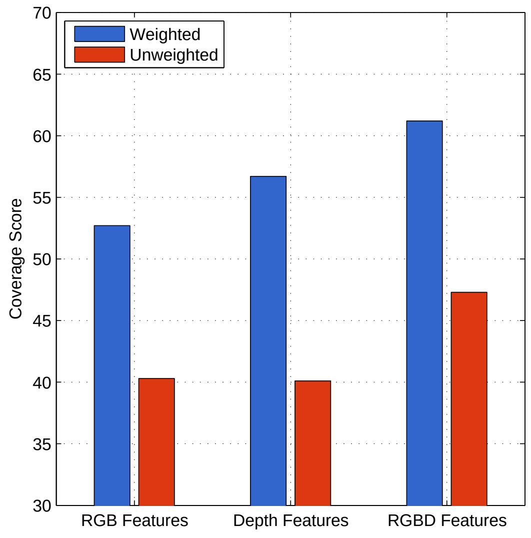


Figure 5.3: Comparing the performance on greedy tree inference using RGB only, Depth only and combined RGBD features.

with regard to unweighted performance, with its higher focus on small objects, but the depth features do provide a boost in terms of weighted performance, with its higher focus on larger objects like walls, floors, and furniture. Clearly, the combined set of RGBD features are superior resulting in a weighted coverage score of 61.2 (9% higher than RGB) and an unweighted coverage score of 47.3 (7% higher than RGB).

To understand the effects of merging regions over time, we plot the Coverage Upper Bound (CUB) as a function of the merging stage. As Figure 5.9.1 indicates, the first two stages result in the most substantial increases in coverage score performance but also result in the largest decreases in CUB scores. This is due to mistakes made in the first stages of region merging from which the algorithm cannot recover. Note that the upper bound scores are computed by allowing merges across occlusion boundaries.

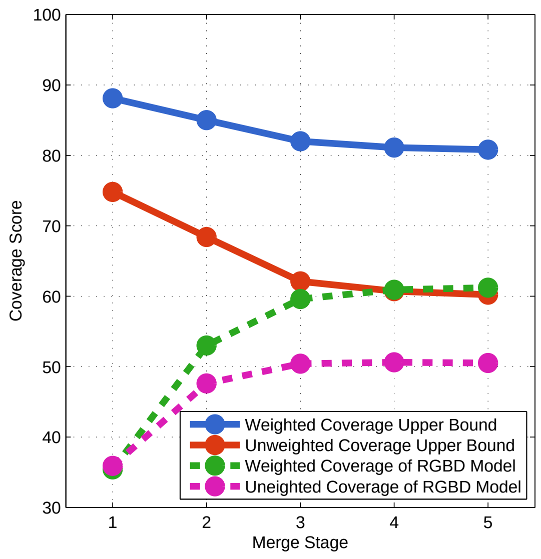


Figure 5.4: Visualizing the effects of merging at each stage on the upper bound scores as computed with an oracle.

5.9.2 Evaluating Tree Proposal Methods

Our loss formulation (Section 5.6) allows us to directly measure the Coverage upper bound (CUB) scores achievable by a particular hierarchical segmentation. Consequently, we can evaluate not only which region proposal algorithms obtain the highest CUB scores, but also consider the trade-off between additional regions and CUB. More regions may raise the CUB score at the expense of computational demands and any application seeking to use segmentation trees in practice will have to evaluate the point of diminishing returns.

We evaluate several region proposal schemes in Table 5.9.2 by computing for each the surrogate labels that maximize the weighted coverage score. Unsurprisingly the depth signal raises the CUB score. Gupta *et al.* [15] outperforms our greedy tree inference scheme both in

Input	Algorithm	Weighted CUB	Average Number of Regions
RGB	Hoiem *et al.* [17]	50.7	117.7 ± 36.7
RGB	Zhile and Shakhnarovich [34]	50.7	102.4 ± 56.4
RGB	Isola *et al.* [?]	48.8	368.4 ± 173.6
RGB+D	Tree from Greedy Tree Inference	64.1	210.0 ± 106.0
RGB+D	Gupta et al [15]	70.2	195.8 ± 88.5

Table 5.2: Segmentation results on the testing set.

terms of CUB score and requires far fewer regions. Zhile *et al.* [34] and Hoiem *et al.* [17] both achieve the same weighted CUB scores but Zhile *et al.* [34] is a bit more efficient requiring fewer regions.

5.9.3 Evaluating CNN Features

To evaluate the CNN features, we used the ground truth instance annotations from the NYU Depth V2 dataset. By evaluating on the ground truth regions, we can isolate errors inherent in evaluating poor regions from the abilities of the descriptor as well as avoid the ground truth mapping problem for assigning semantic labels. To prepare the inputs to the CNN we perform the following operations: For each instance mask in the dataset (a binary mask for a single object instance), we compute a tight bounding box around the object plus a small margin (10% of the height and width of the mask). If the bounding box is smaller than 140×140, we use a 140×140 bounding box and upsample to 244×244. Otherwise, we rescale the image to 244×244. During training, we additionally include all horizontal reflection of each window as well as several scales $(1.1, 1.3, 1.5, 1.7)$. Finally, we ignore regions whose original size is smaller than 20×20. We computed a random train/val/test split using the original 1449 images in the dataset of equal sizes. After performing each of the aforementioned masking operations and computing the CNN features from each subwindow, we normalize each output feature from the CNN by subtracting the mean and diving by the variance, across the training set. We then train a linear classifier with a softmax output and

Features	Accuracy	Conf Matrix Mean Diagonal
Normal Windows	48.8	23.5
Mask-Blurred Windows	56.6	36.8
Masked Windows [13]	60.8	42.3
RGBD Features [32]	59.9	32.1
Sparse Coded Sift + RGBD Features [38]	60.3	34.4
Unblurred Windows + RGBD Features	60.3	38.0
Mask-Blurred Windows + RGBD Features	63.1	46.1
Masked Windows + RGBD Features	64.9	46.9

Table 5.3: A comparison on region-feature descriptors on ground truth regions.

L2 regularization to predict the correct semantic labels of each instance. The regularization parameters were chosen to maximize accuracy on the validation set and are found in the supplementary material.

As shown in Table 5.3, the CNN features perform surprisingly well, with Masked Windows computed on RGB only beating both RGBD Features and the combination of sparse coded SIFT and RGBD Features. Our Mask-Blurring operation does not do as well as Masking, with the combination of RGBD Features and CNN Features extracted from Masked regions performing the best.

5.9.4 Segmentation

To evaluate our semantic/instance segmentation results, we use the semantic and instance labels from [32]. We train and evaluate on the same train/test split as [32] and [19]. We computed the surrogate labels using the weighted coverage loss and report all of the results using the weighted coverage score. To train our model, we used the Block Coordinate Frank Wolf algorithm [25] for minimizing the loss function, and the Gurobi [?] ILP solver for performing inference. Loss augmented inference takes several seconds per image whereas inference at test time takes half a second on average. Note that at test time, inference can be performed instead using the graph cuts based model of [27]. If we do not use the pairwise terms, inference can be performed extremely quickly using the algorithm for tree-cutting in Section A.3.

Evaluating Semantic-Instance Segmentation Results

We evaluate several different types of Semantic-Instance Segmentation models. The model **SEG Trees, SIFT Features** uses the standard segmentation trees with Sparse Coded SIFT features, **SEG Trees, CNN Features** uses standard segmentation trees and CNN Features using the Masking strategy, **SEG + GT Trees, CNN Features** uses the CNN features as well but also trains with a set of height-1 trees created from the ground truth instance maps. Since these height-1 trees are by definition already segmented, during training we use the Coverage Loss for the SEG Trees and Hamming Loss on the semantic predictions for the GT Trees. The last model we evaluated, **GT-SEG Trees, CNN Features**, is a model trained on segmentation trees biased by the ground truth.

As shown in Table 5.4 our model achieves state of the art performance in segmenting the dense scenes from the NYU Depth Dataset. While the use of SIFT features makes a negligible improvement with regard to previous work, using convolutional features provides almost a 1% improvement. **SEG + GT Trees, CNN Features**, which was trained to minimize coverage loss on the SEG trees and semantic loss on the ground truth performs slightly better. The addition of the GT trees to the training data acts as a regularizer for the semantic weights on the high dimensional CNN features by requiring that the weights are both useful for finding instances of imperfectly segmented objects and correctly labeling objects if a perfect region is made available. Qualitative results for this model are shown in Fig. 5.9.4 along with a comparison to [19]. As these figures illustrate, the model performs

Algorithm	Weighted Coverage
Greedy Tree Inference	61.2
Jia et al [19]	61.7
Global Tree Inference - SEG Trees, SIFT Features	61.8
Global Tree Inference - SEG Trees, CNN Features	62.5
Global Tree Inference - SEG + GT Trees, CNN Features	62.8
Global Tree Inference - GT-SEG Trees, CNN Features	87.4

Table 5.4: Segmentation results on the NYU Depth V2 Dataset

Loss Function	Weighted Coverage
Hamming Loss	61.4
Weighted Coverage Loss	62.5

Table 5.5: Evaluating the use of the Coverage Loss

better on larger objects in the scene such as the couch in row 1, and the bed in row 4. Like [19] however, it struggles with smaller objects such as the clutter on the desk in row 5.

Finally **GT-SEG Trees, CNN Features** is a model trained on segmentation trees biased by the ground truth. This model achieves a coverage score of 87.4% which indicates that while our model shows improvement over previous methods at instance segmentation, it still does not achieve a perfect coverage score even when the ground truth is available as a possible cut.

Evaluating the Loss function

To evaluate the effectiveness of using the Coverage Loss for instance segmentation, we use the same model but vary the loss function used to minimize the structural SVM. We compare against using the hamming loss. We use the use regularization parameter [1] for both experiments.

[1] $\lambda = .001$

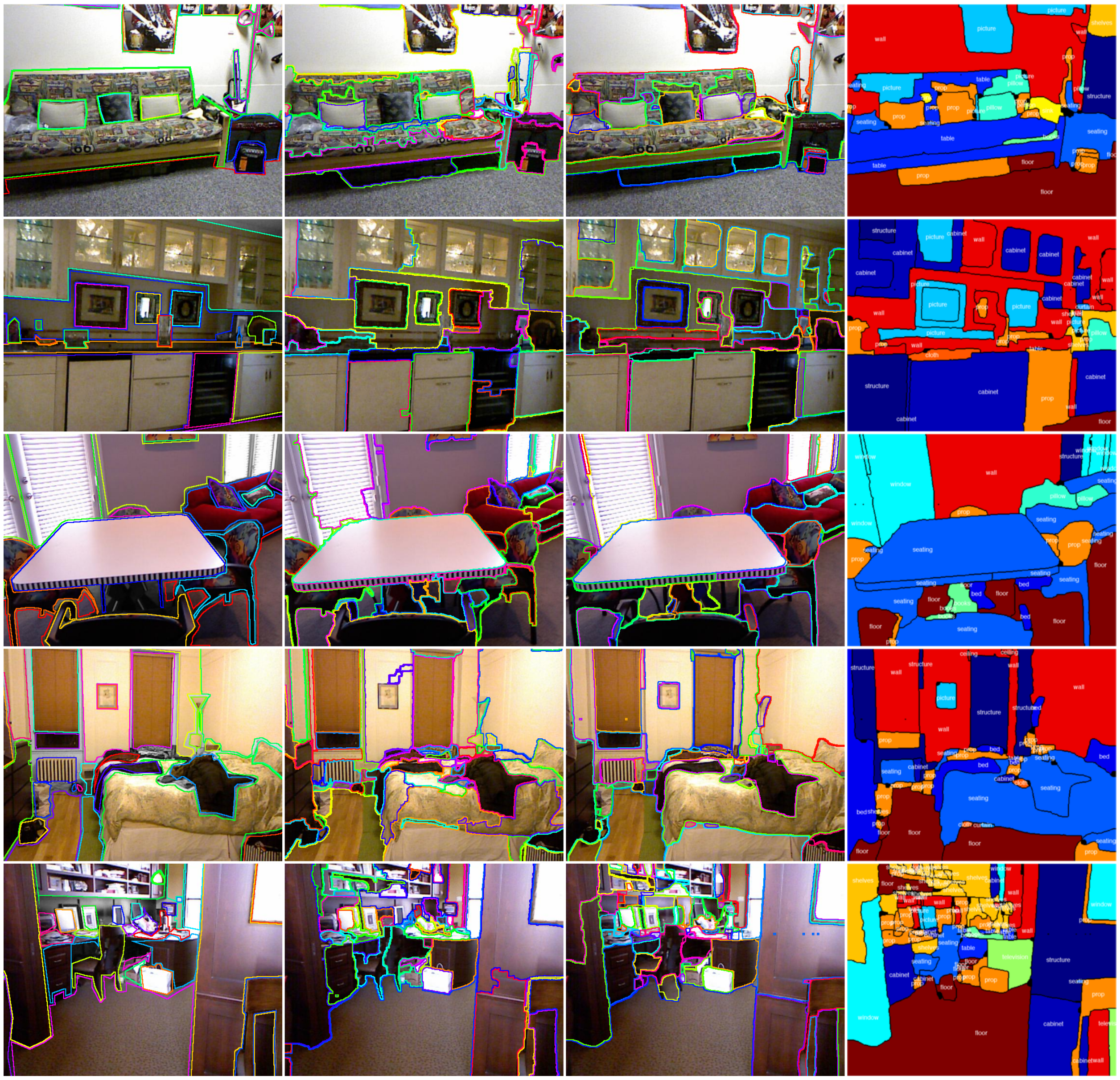


Figure 5.5: Random test images from the NYU Depth V2 dataset, overlaid with segmentations. 1st column: ground truth. 2nd column: segmentations from Jia et al [19]. 3rd colmun: our segmentations. 4th column: semantic labels, produced as a by-product of our segmentation procedure.

Chapter 6

Exemplar-based Instance Segmentation

While our greedy and global tree-based models show good performance, they are still hindered by the heuristic they both employ. By reasoning over a limited set of regions to make inference computationally feasible, they eliminate many superior solutions. Greedy tree merging is imperfect and makes mistakes from which it can never recover. Global tree inference does allow for reasoning across levels of the tree but it cannot undo mistakes made in tree creation. Furthermore, tree based methods traditionally only consider merging adjacent reasons for computational reasons. This prohibits the algorithm from reasoning across inter-object occlusions.

To address these concerns, we explore two exemplar-based instance segmentation algorithms. In both methods, we select a series of exemplars from a pool of regions to which each pixel in the scene is assigned. This approach has several advantages over tree-based methods. Firstly, by keeping region (exemplar) selection separate from pixel assignment, we are no longer limited to the small and static pool of regions first extracted. This is due to the fact that a pixel outside an exemplar can later be assigned to it and a pixel inside the exemplar can later be excluded from it. Secondly, by allowing any pixel to be assigned to any exemplar, we can easily reason across occlusion boundaries.

Our first exemplar-based algorithm performs instance segmentation in two separate stages. In the first stage, we select a series of exemplars and assign each one a semantic class. Ideally, each object instance in the scene should be represented by a single exemplar region but not every pixel will be represented by an exemplar. The second stage is treated as a multi-class semantic segmentation problem. Instead of assigning each pixel to a semantic class, however, each pixel is assigned to one of the exemplars.

Our second algorithm combines these two stages into a single joint inference operation. We perform inference using a novel Max Product Linear Programming (MPLP) formulation. While modeling the scene densely via both pixels and exemplars introduces a huge number of unknowns, we propose a series of efficient message updates that speed up inference. Additionally, we use a pruning scheme that greatly reduces the number of unknowns in practice.

6.1 Two Stage Inference

Our algorithm that performs instance segmentation in two stages proceeds as follows. In the first stage, we seek a single representative region, or exemplar, for each object instance in the scene. Due to the difficulty inherant in the tree proposal process (Section 5.9.2), the exemplars that we select will be imperfect. We hope, however, that they sufficiently approximate as many of the object instances as possible, such that the exemplars can be corrected later. In the second stage, we assign each pixel to one of the exemplars based on a set of features and learned weights that capture the similarity of each pixel to each exemplar.

6.1.1 Exempler Subselection

In the first stage of inference, our goal is to select a series of exemplars from a pool of regions from which our second stage will then perform pixel assignment. We use a similar model to our globally inferred instance-segmentation tree scheme (Section 5.5). To review, we are given a tree of regions and a cut of the tree is represented by $\{\mathbf{A} : A_i \in \{0,1\}, i = 1..R\}$, a vector indicating whether or not each of the R regions in the tree are selected. Let $\{\mathbf{C} : C_i \in \{1..K\}, i = 1..R\}$ be a vector indicating the semantic class (out of K) of each region. Finally let $y = \{\mathbf{A}, \mathbf{C}\}$ be the combined output of semantic labels and region instances.

Inference is formulated as a structured prediction problem whereby we seek the most compatible pairing of inputs x and outputs y from the space $\mathcal{Y}$ of all region selections and semantic class assignments for a given segmentation tree. Formally, we predict using

$$y^* = \arg\max_{y \in \mathcal{Y}} w^T \phi(x, y) \tag{6.1}$$

where the compatibly function $w^T \phi(x, y)$ is defined as:

$$w^T \phi(x, y) = w_{\text{reg}}^T \phi_{\text{reg}}(x, y) + w_{\text{sem}}^T \phi_{\text{sem}}(x, y) + w_{\text{card}}^T \phi_{\text{card}}(x, y) + \phi_{\text{tree}}(y) \tag{6.2}$$

where ϕ_{reg} are generic region potentials (Equation 5.3) and ϕ_{sem} are semantic region potentials (Equation 5.4). The **semantic cardinality potentials**, $\phi_{\text{card}}(x, y)$, are defined as:

$$\phi_{\text{card}}(x, y) = \sum_{k=1}^{K} w_{\text{count}(y,k)} \tag{6.3}$$

where $\text{count}(y, k)$ is the number of regions assigned to semantic class k and w_c is a class specific weight vector where c indexes counts. Finally the **tree consistency** potential $\phi_{\text{tree}}(y)$

is similar to one previously used (Equation 5.6) with one crucial difference: rather than require that all pixels be explained by some region, we allow pixels to be unexplained. Formally, we ensure that *at most* one region is selected along every path from leaf to root:

$$\phi_{\text{tree}}(y) = \sum_{\gamma \in \Gamma} -\infty \big[1 < \sum_{i \in \gamma} 1[y.A_i = 1]\big] \tag{6.4}$$

where γ is a path in the set Γ of all paths.

Defining the Ground Truth

Previously, we trained a tree cutting scheme to maximize the coverage score of the resulting set of selected regions. In our two-stage segmentation scheme, however, this is not necessarily ideal. In particular, two regions might have the same coverage score but might approximate an object in ways that are easier or harder to correct via pixel assignment. For example, if a region is an over-segmentation of an object instance, then the region may have somewhat uniform appearance or texture and it should be easy for the pixel assignment stage to grow this region. Conversely, if a region is an under-segmentation of an object instance, then the region will contains parts of multiple objects. This situation is not only ambiguous in terms of how the pixel assignment should proceed but also lessens the usefulness of the features as they will be computed from a mixture of multiple objects. An example of this problem is illustrated in Figure 6.1.

Training

The model is trained via a structured SVM and Block Coordinate Frank-Wolfe [25]. As illustrated in Section 6.1.1, our goal in training is not to find the set of regions that maximizes the coverage score but rather to find the set of regions that maximizes a combination of

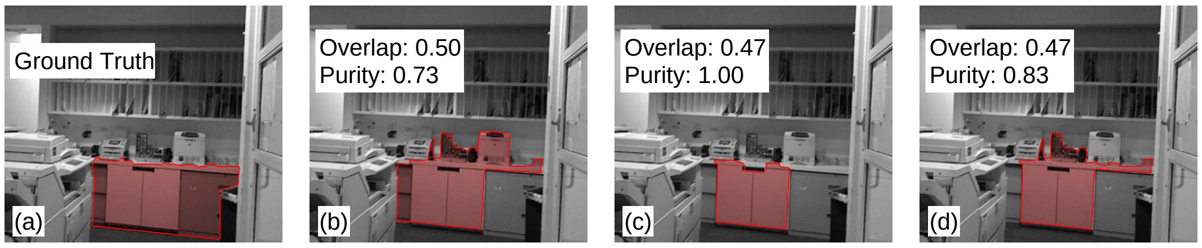


Figure 6.1: Regions with the highest overlap score are not necessarily the easiest to 'correct' via a subsequent stage: (a) A ground truth cabinet region. (b) The region with the highest overlap also contains several additional objects. How a subsequent region should proceed is ambiguous as it's unclear whether the resulting region should crop the pixels belonging to the cabinet, the printer, the stapler or the file holder. (c) A region with lower overlap but higher purity can be unambiguously identified as a cabinet and it is clear how to grow the region in the subsequent stage. (d) A region with lower overlap and lower purity is both a poorer approximation of the underlying region and also is marred by the ambiguity of determining the object instance being approximated.

coverage and purity. Consequently, rather than training using the coverage loss, we train the subselection model using margin-rescaling and hamming loss.

6.1.2 Pixel Assignment

Following exemplar selection, a set of regions and their associated semantic classes have been extracted from each image. To perform pixel assignment, we define a CRF over the image:

$$E(Y) = \sum_{i \in N} \phi_i(y_i) + \sum_{ij} \psi_{ij}(y_i, y_j) \tag{6.5}$$

where N is the number of pixels, ϕ_i defines the unary potentials over each pixel and $\psi_{ij}(y_i, y_j)$ defines the spatially varying pairwise potential over all pairs of neighboring pixels. While this formulation is common to semantic segmentation approaches, the labels y_i are typically semantic classes. In our case, the labels represent the exemplars from the exemplar subselection stage. Formally, if R exemplars have been selected from the subselection stage for a given image, then $y_i \in \{1 \dots R\}$. Note that R will typically vary from image to image so the solution space of Y is image dependant.

The unary features are defined as:

$$\phi_i(y_i) = w_{c(y_i)}^T f_{i,y_i}^u \tag{6.6}$$

where $c(y_i)$ is the semantic class of exemplar y_i inferred by the first stage (Section 6.1.1), w_c are a set of learned weights for class c and f_{i,y_i}^u are the unary pixel features that capture the similarity between pixel i and exemplar y_i. The pairwise transition potential is defined as:

$$\psi_{ij}(y_i, y_j) = \begin{cases} w_p^T f_{ij}^p, & \text{if } y_i \neq y_j. \\ 0, & \text{otherwise.} \end{cases} \tag{6.7}$$

where w_p^T are a set of learned pairwise transition weights and f_{ij}^p are pairwise features describing the transition between pixels i and j.

Pixel Assignment Features

The unary features in our model should capture the likelihood that a pixel belongs to the object represented by a particular exemplar. This likelihood is based on appearance features, such as color or texture, 3D proximity, and 3D shape information. The features used are listed in Table 6.1.

We also extract a set of transition features that capture the likelihood of a label transition between neighboring pixels. These features are listed in Table 6.2.

Pixel to Exemplar Unary Feature Descriptions	Dims
2D Distance	**4**
D1. Is pixel part of exemplar mask	1
D2. Log, Sqrt and value of in-plane distance from pixel to exemplar mask	3
3D Distance	**8**
E1. Planar disparity between pixel and planar approximation of exemplar	1
E2. Whether or not the planar disparity is under several thresholds.	5
E3. Distance along the ground plane between the pixel and the exemplar	1
E4. Distance along the ground plane between the pixel and the exemplar minus outlying points	1
Color	**10**
C1. Mean RGB/LAB difference: pixel's SLIC[1] superpixel and exemplar	2
C1. Mean RGB/LAB difference: pixel's EGBS[2] superpixel and exemplar	2
C1. Mean RGB/LAB difference: pixel's EGBS[3] superpixel and exemplar	2
C1. Mean RGB/LAB difference: pixel's EGBS[4] superpixel and exemplar	2
C1. Mean RGB/LAB difference: pixel's EGBS[5] superpixel and exemplar	2
Shape	**6**
S1. Normals: angular distance between pixel normal and planar approximation of exemplar normal	1
S2. Planes: whether pixel is part of the dominant horizontal or vertical plane of the exemplar	2
S3. Planes: whether the pixel is part of any detected horizontal or vertical planar surface	1
S4. Cubes: whether the exemplar forms a cuboid and the pixel is part of the cuboid	1
3D Position	**2**
P1. Distance from the top of the scene	1
P2. Distance from the bottom of the scene	1

Table 6.1: List of features used to capture the likelihood that a pixel belongs to a particular exemplar of a given semantic class.

Pixel to Exemplar Transition Feature Descriptions	**Dims**
RGB Features	**6**
R1. Exponential of difference of gaussians with several bandwidths	3
R2. Exponential of pairwise RGB differences with several bandwidths	3
Depth Features	**3**
D1. Exponential gradient of depth gradients with several bandwidths	3
Model Features	**2**
M1. Probability of Boundary Costs [15]	1
M2. Structured Forest Contours [?]	1
3D Position	**2**
P1. Distance from the top of the scene	1
P2. Distance from the bottom of the scene	1

Table 6.2: List of pairwise features used to penalize pixel assignment transitions.

Training the Pixel Assignment Model

We train our CRF model to perform pixel assignment using the structured SVM scheme of [?]. We define the training set as follows. For each image, we extract a segmentation tree (Section 5.4) but ignore the structure of the tree and consider the regions a structureless pool. For each ground truth region in an image, we select the region from the pool that maximizes the following:

$$\max_{s \in \mathcal{S}} \text{Overlap}(r_g, r_s) + \tau \text{Purity}(r_g, r_s) \tag{6.8}$$

where r_g is a ground truth region, $\mathcal{S}$ is the pool of proposed regions and r_s is one of the regions from the pool. The Overlap score is defined as (Overlap $= \frac{|r_g \cap r_s|}{|r_g \cup r_s|}$), and the region Purity captures the fraction of intersecting pixels over the size of the proposed region itself: Purity $= \frac{|r_g \cap r_s|}{|r_s|}$. We repeat this procedure for each one of the G ground truth region of the image which results in a set of G proposed regions which are generally not overlapping. These regions and the semantic classes of the ground truth regions will serve as the inputs to training from which features are extracted.

To define the ground truth, we simply map each pixel that belongs to a ground truth region to the exemplar which was selected as its best approximating exemplar when creating the input regions (Equation 6.8). More formally, if proposed region r_s was selected as the best approximating region of ground truth region r_g, then $y_i = s \ \forall i \subset r_g$.

6.2 Joint Exemplar Selection and Pixel Assignment

Rather than perform exemplar selection and pixel assignment in two disjoint stages, we can instead perform them jointly. Given an image with N pixels and M exemplars from a segmentation tree, our goal is to assign each pixel to an exemplar and each exemplar to a semantic class. Let $E = \{E_m | m = 1 \dots M, E_m \in \mathcal{C}\}\}$ refer to the set of variables encoding the semantic class of each exemplar where $\mathcal{C} = \{0, 1 \dots K\}$ represents the set of semantic classes and $E_m = 0$ indicates that exemplar E_m is not used and consequently that no pixels are assigned to it. If $E_m > 0$, then we will refer to that exemplar as having been *selected*.

The cost of assigning a pixel to a particular exemplar will depend not only on the characteristics of the pixel and exemplar but also on the exemplar's semantic class. For example, pixels that are too far from the planar fit of an exemplar will be unlikely to be assigned to that exemplar if its semantic class is 'wall' or 'picture'. However, the pixel might still belong to the exemplar if its class is 'sofa'. Consequently, rather than assign each pixel directly to an exemplar, each pixel will instead be assigned to a label in the label space of the outer product of exemplars and semantic classes. Let $\mathcal{L} = \mathcal{M} \times \mathcal{C}$ be the set of possible pixel labels and $P = \{P_i | i = 1...N, P_i \in \mathcal{L}\}$ refer to the set of variables encoding the label assignments for each pixel.

We introduce three functions to map between exemplars, semantic classes and the pixel space $\mathcal{L}$. Let $\mu : \mathcal{L} \rightarrow \mathcal{M}$ be a function that maps pixel labels to their corresponding exemplars and $\kappa : \mathcal{L} \to \mathcal{C}$ be a function that maps pixel labels to their corresponding semantic classes. Finally, let $\zeta : \mathcal{M} \times \mathcal{C} \to \mathcal{L}$ be the function that maps from an exemplar and semantic label to the corresponding pixel label in $\mathcal{L}$.

Given an image, we want to infer the labels $Y = \{P, E\}$ that maximize the following energy function:

$$\begin{aligned} E(Y) = &\sum_{m=1}^{M} \phi_m^{\text{ex}}(E_m) + \sum_{i=1}^{N} \phi_i^{\text{pix}}(P_i) + \phi^{\text{tree}}(E) \\ &+ \sum_{m=1}^{M} \phi_m^{\text{agree}}(E_m, P) + \sum_{k=1}^{K} \phi_k^{\text{card}}(E) \\ &+ \sum_{ij \in \mathcal{N}} \phi_{ij}^{\text{pair}}(P_i, P_j) \end{aligned} \tag{6.9}$$

The **exemplar potentials** $\phi_m^{\text{ex}}(E_m)$ incur the cost of assigning a semantic class to a particular exemplar:

$$\phi_m^{\text{ex}}(E_m) = \begin{cases} (w_k^{\text{ex}})^T f_m^{\text{ex}} & \text{if } E_m = k \\ 0 & \text{if } E_m = 0 \end{cases} \tag{6.10}$$

where (w_k^{ex}) are class specific exemplar weights and f_m^{ex} are region features. The **pixel potentials** $\phi_i^{\text{pix}}(P_i)$ incur the cost of assigning each pixel to a combination of exemplar and semantic class:

$$\phi_i^{\text{pix}}(P_i) = \begin{cases} (w_k^{\text{pix}})^T f_{i,m}^{\text{pix}} & \text{if } \mu(P_i) = m \wedge \kappa(P_i) = k \\ 0 & \text{if } P_i = 0 \end{cases} \tag{6.11}$$

where w_k^{pix} are class specific pixel weights and $f_{i,m}^{\text{pix}}$ are the pixel to exemplar features. The **tree potential** $\phi^{\text{tree}}(E)$ ensures that the selected regions form a valid cut of the tree but does not enforce completeness. In other words, at most a single region along each path from leaf to root can be selected. It is defined as:

$$\phi^{\text{tree}}(E) = \sum_{\gamma \in \Gamma} -\infty \big[1 < \sum_{i \in \gamma} 1[E_m \geq 1] \big] \tag{6.12}$$

where Γ represets the set of paths (each of which are a set of regions) from leaf to root. The **pixel-exemplar agreement potentials** $\phi_m^{\text{agree}}(E_m, P)$ ensure that if an exemplar is assigned a semantic class, then the pixel assignments agree with this assignment. It is defined as:

$$\phi_m^{\text{agree}}(E_m, P) = \begin{cases} -\infty, & \text{if } E_m = 0 \wedge \exists\, P_i = l, \mu(l) = m \\ -\infty, & \text{if } E_m = c \wedge \nexists\, P_i = l, \mu(l) = m, \kappa(l) = c \\ -\infty, & \text{if } E_m = c \wedge \exists\, P_i = l, \mu(l) = m, \kappa(l) \neq c \\ 0, & \text{otherwise} \end{cases} \tag{6.13}$$

The first condition ensures that if the exemplar is not selected, then no pixel can be assigned to it. The second condition ensures that if an exemplar is assigned to a semantic class, then there must exist at least one pixel assigned to that exemplar of the same semantic class. The third condition ensures that if the exemplar is selected and assigned to a particular semantic class, then if a pixel is assigned to the exemplar, it must be assigned via the pixel label of the correct semantic class. The **class-cardinality potentials** penalize the use of the number of object instances of each class type:

$$\phi_k^{\text{card}}(E) = w_{b,k} \tag{6.14}$$

where $w_{b,k}$ is the weight incured for assigning b instances to semantic class k. Finally, the

pairwise potentials penalize transitions over all pairs of neighboring pixels $ij \in \mathcal{N}$ using a learned spatially varying Potts model:

$$\phi_{ij}^{\text{pair}}(P_i, P_j) = \begin{cases} w_{\text{pair}}^T f_{ij}^{\text{pair}} & \text{if } P_i \neq P_j \\ 0 & \text{if } P_i = P_j \end{cases} \tag{6.15}$$

where w_{pair}^T are a set of learned pairwise weights and f_{ij}^{pair} is a feature vector describing the boundary between pixels i and j.

6.2.1 Dual Decomposition

Because Equation 6.9 cannot be efficiently optimized directly, we will instead optimize an upper bound of the energy. Let the parameterized dual function $L(\delta, \lambda, \lambda', \lambda'')$ be defined such that $L(\delta, \lambda, \lambda', \lambda'') \geq E(Y)$. We can then optimize for the dual variables $\delta, \lambda, \lambda', \lambda''$ that minimize L to make the bound as tight as possible. We define the dual function as:

$$
\begin{aligned}
L(\delta, \lambda, \lambda', \lambda'') = &\sum_{m=1}^{M} \max_{E_m} \left[\theta_m^{\text{ex}}(E_m) + \lambda(E_m) + \lambda'(E_m) + \sum_{k=1}^{K} \lambda''_{km}(E_m)\right] \\
&+ \sum_{i=1}^{N} \max_{P_i} \left[\theta_i^{\text{pix}}(P_i) + \sum_{f \in F : i \in f} \delta_{fi}(P_i) + \sum_{m=1}^{M} \delta'_{mi}(P_i)\right] \\
&+ \max_{E'} \left[\theta^{\text{tree}}(E') - \sum_{m=1}^{M} \lambda_m(E'_m)\right] \\
&+ \sum_{m=1}^{M} \max_{E''_m, P'} \left[\theta_m^{\text{agree}}(E''_m, P') - \lambda'_m(E''_m) - \sum_{i=1}^{N} \delta'_{mi}(P'_i)\right] \\
&+ \sum_{k=1}^{K} \max_{E'''} \left[\theta_k^{\text{card}}(E''') - \sum_{m=1}^{M} \lambda''_{km}(E'''_m)\right] \\
&+ \sum_{f \in F : ij \in f} \max_{P''_i, P''_j} \left[\phi_{ij}^{\text{pair}}(P''_i, P''_j) - \delta_{fi}(P''_i) - \delta_{fj}(P''_j)\right] \qquad (6.16)
\end{aligned}
$$

where F is the set of all pairwise edges between neighboring pixels. If we were to add constraints to ensure that $E = E' = E'' = E'''$ and $P = P' = P''$, we would obtain the original energy function $E(Y)$. By relaxing these constraints, each potential function can be separately optimized resulting in an upper bound of the original energy.

To optimize $L(\delta, \lambda, \lambda', \lambda'')$, we use the Max Product Linear Programming (MPLP) algorithm. This algorithm minimizes the value of the lagrangian L via block coordinate descent. During each iteration of the algorithm, the values of the dual variables are updated in blocks such that the dual objective after each variable block update decreases monotonically [**?**]. After each update, the value of the lagrangian objective is computed to determine whether or not the algorithm has converged.

While the block coordinate update steps, as well as the computation of the lagrangian, involve an exponential number of states, the variable updates and lagrangian computation can both be efficiently computed (Appendix A).

6.3 Pruning

In order to speed up our joint inference scheme, we reduce the solution space by pruning the set of allowable labels. We train models to prune both the exemplar label space and the pixel-to-exemplar label space.

6.3.1 Exemplar Label Pruning

Rather than consider all possible assignments of exemplars to semantic labels, we prune some of the exemplar labels. To prune the exemplars, we train a series of class-specific binary detectors via the following procedure. First, a set of exemplars are produced via two strategies. In the first, we simply consider each ground truth region an exemplar. In the second, we follow the algorithm of Gupta et al [15] to produce a segmentation tree from each image using the thresholds from [?] to produce 194 ± 90 regions per image. Exemplar features are then extracted from each of these regions producing around 80,000 training points.

For each class, we compute the overlap between every exemplar and each ground truth region of that class. Any exemplar with an overlap score of greater than .2 is labeled as a positive and any exemplar with an overlap score of 0 is labeled as a negative. Exemplars with overlap scores between 0 and .2 are discarded.

Before training, each feature descriptor is reduced in size via PCA and then normalized to 0 mean, unit variance. Finally, a logistic regressor is trained to predict each label.

During joint inference, each exemplar detector is run on each exemplar. We interpret the output of each logistic regressor, a number between 0 and 1, to be a pruning confidence. If a class detector's output confidecne is below .8, then exemplar label is pruned. Note that the joint-inference scheme requires a segmentation tree as input. It is indeed possible that the pruning *may* force certain exemplars to be completely removed from consideration as all of the class-specific detectors have pruned their corresponding labels. In practice, however, it is rare that an exemplar cannot be assigned to even a single semantic class after pruning.

6.3.2 Pixel-to-Exemplar Pruning

To prune the exemplar labels, we train class-specific classifiers via the following method. First, we extract a set of exemplars from the segmentation tree of [15]. Next, for each image and each ground truth region, we select a single exemplar that has maximal overlap (provided it has at least a purity score of .5 and minimum overlap of .2). Following this procedure, for an image with 10 ground truth regions, we have *at most* 10 exemplars, all of which have purity of at least .5 and overlap of at least .2 with one of the ground truth regions. Next, we extract pixel-to-exemplar features.

To label each pixel-to-exemplar features, we consider a pixel-to-exemplar feature a positive data point for a class if (a) the exemplar overlaps with a ground truth region of that class and (b) if the pixel is included in the ground truth map of the corresponding ground truth region. Otherwise, the data point is considered a negative.

For each class, we train a separate binary classifier to predict whether or not each pixel should be assigned to a given exemplar of that class.

6.4 Learning

Learning the weights of the model jointly is difficult as it requires that we indicate a *best* set of regions and their pixel assignments. In this case, its not clear how to define this set. Furthermore, because inference is slow, training the model jointly is computationally expensive. Instead, we train the components of the model piecewise and scale the weights using a held-out validation set. The exemplar weights are trained using the subselection training scheme (Section 6.1.1). The pixel to exemplar weights are trained using the pixel assignment training schme (Section 6.1.2).

6.5 Experiments

6.5.1 Stage 1: Subselection

To evaluate our region subselection routine, we analyze how well our subselection routine performs in comparison to the tree cutting algorithm previously discussed (Section 5.5). These models differ in two respects. Firstly, the previous tree cutting model is trained to produce a cut that is *complete*. That is, the result of cutting the tree is a set of regions that explains every pixel. Conversely, the output of the subselection stage is a *partial* explanation of the scene that leaves parts of the scene unexplained. Secondly, during training of the previous tree cutting model, the ground truth regions maximize the coverage score with respect to the ground truth. Conversely, when we train our subselection model, we are maximizing a combination of coverage and purity. Consequently, we can compare the resulting models by evaluating both coverage and purity:

Model	Weighted Coverage	Unweighted Coverage	Purity
Tree Cutting, Complete (Section 5.5)	62.8	47.3	73.6
Tree Cutting, Incomplete	53.3	38.6	75.5
Subselection Model	61.7	42.2	77.5

Table 6.3: Evaluating subselection models

As Table 6.3 indicates, when we relax the completeness constraints on the previously trained tree cutting model, the results are very poor. This is not completely unexpected as the weights learned for our prior model are effectively only used to rank possible cuts in the tree. In the current task, the weights must both rank cuts in the tree *and* not select regions that are of inferior quality. When we re-train a model to select the best subset of regions and to ignore or leave blank areas without good regions, we obtain a model that produces slightly inferior coverage scores but improved purity scores (an increase from 73.6 to 77.5). Our ability to select only good regions comes at a cost: our ability to select small objects drops. This is made clear by virtue of the fact that the unweighted coverage score drops from 47.3 to 42.2. Further evidence and qualitative results are illustrated in Figure 6.5.1.

6.5.2 Stage 2: Pixel and Superpixel Assignment

To evaluate our pixel and superpixel assignment algorithms, we compare the coverage scores of the input regions (Section 6.1.2) against the inferred regions. We also compare the effects of using a pixel-wise assignment strategy against a superpixel assignment strategy. In these experiments, we used SLIC superpixels with approximately 2000 superpixels per image.

	Input			Predictions	
Model	Weighted Cov.	Unweighted Cov.		Weighted Cov.	Unweighted Cov.
Pixels	67.6	58.3		74.0	60.5
Superpixels	68.5	54.5		74.5	56.6

Table 6.4: Evaluating Pixel and Superpixel Assignment

As Table 6.4 indicates, the use of superpixels over pixels actually seems to improve the weighted coverage but does worse in terms of unweighted performance. This is due to the fact that the act of superpixelation often results in boundaries whose quality is proportional to the object size. Consequently, after superpixelating the image, we reduce the quality of any subsequent superpixelation from which the model cannot recover.

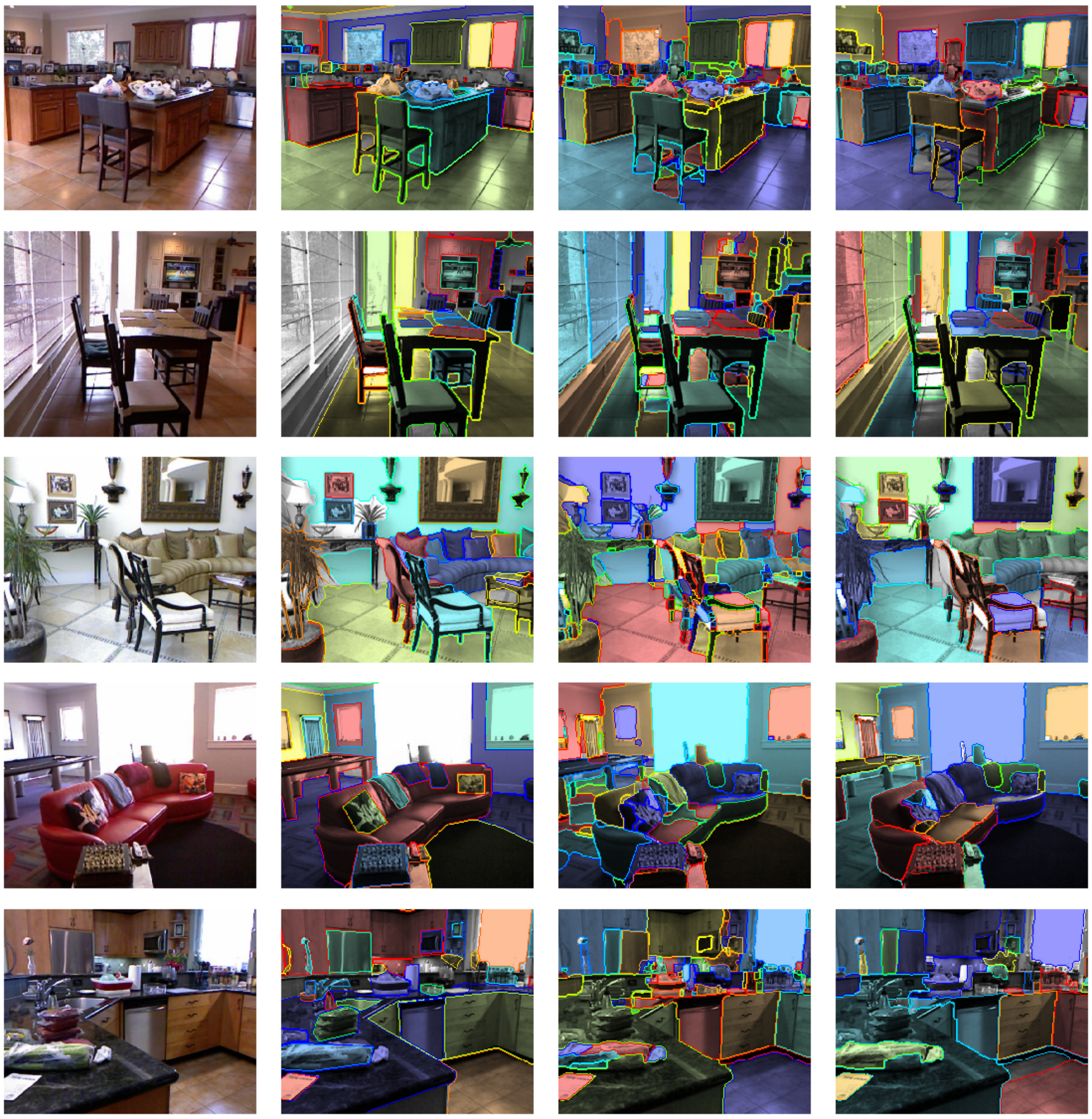


Figure 6.2: Random test images from the NYU Depth V2 dataset, overlaid with segmentations. 1st column: RGB Image. 2nd column: Ground Truth. 3rd column: Tree Cutting Model (Section 5.5). 4th column: Stage 1 Results. Note that the stage 1 subselection stage does not necessarily explain every pixel. For example, part of the couch in the 3rd row, last column is unexplained by any region as indicated by the grayscale hue.

6.5.3 Evaluating Two-Stage and Joint Inference

Finally, we evaluate the two stage and joint inference algorithms. For two-stage inference, we use the retrained subselection model for the first stage and the superpixel assignments for the second stage. Quantitative results are shown in Table 6.5 and qualitative results are shown in Figure 6.5.3. While both the two stage algorithm and the joint inference model perform marginally better in terms of weighted coverage, their inability to handle small objects is a major limitation. Additionally, both algorithms suffer from the inability to avoid selecting regions that represent the same object instance. While transition and cardinality potentials are meant to alleviate this issue, they are not strong enough to result in substantial improvements. Further work in the use of better higher-order potentials that penalized multiple regions representing the same object would be a boon to these types of algorithms.

Model	Weighted	Unweighted
Tree Cutting model: Greedy Inference	61.2	50.5
Tree Cutting Model: Global Inference	62.8	47.4
Two-Stage Exemplar Model	63.0	43.3
Joint Model, Unary and Agreement Potentials	62.4	42.5
Joint Model, Unary, Agreement and Pairwise Potentials	62.9	44.4

Table 6.5: Evaluating exemplar-based models using weighted and unweighted coverage scores.

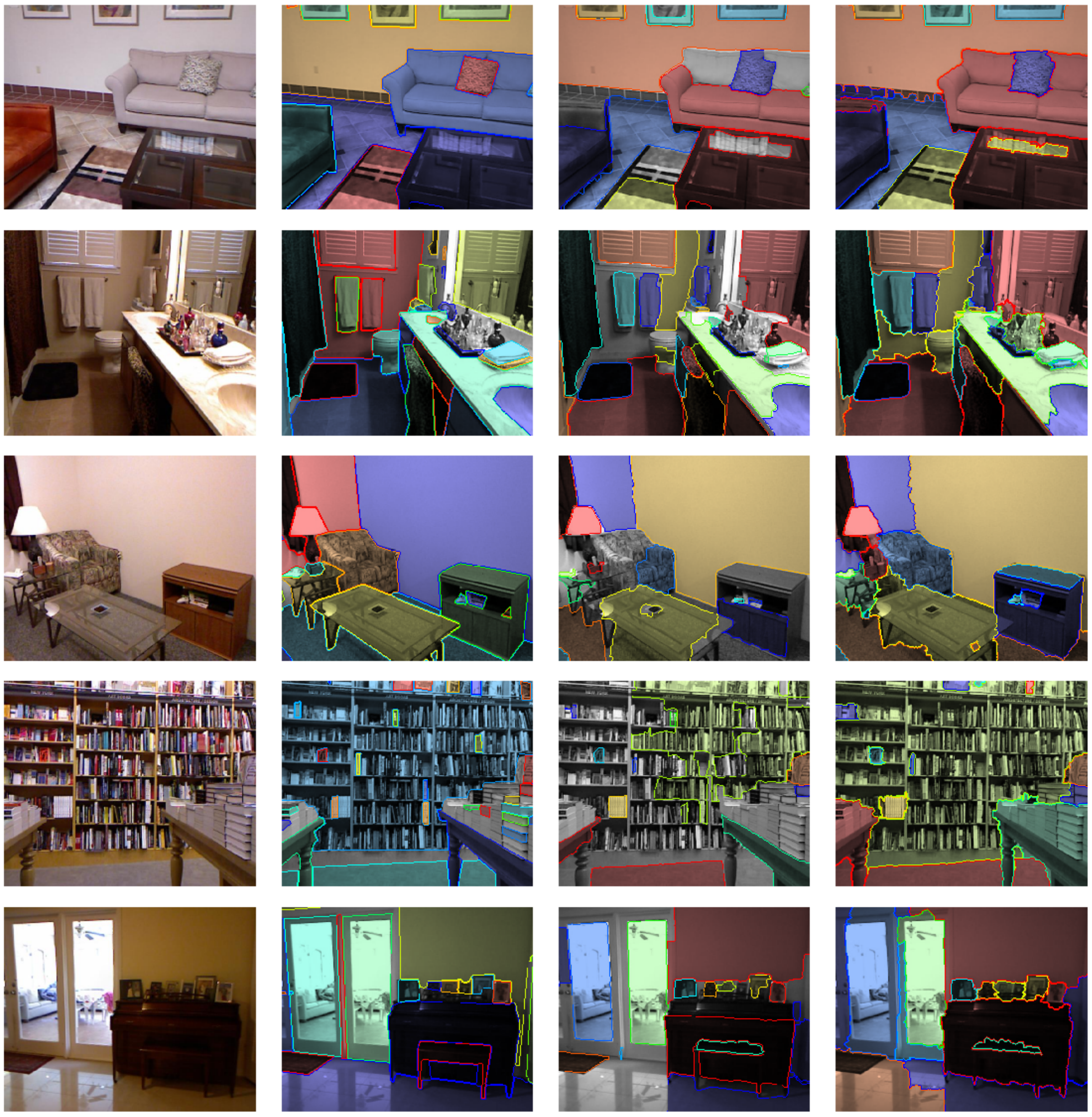


Figure 6.3: Visualize Stage 2 Results. 1st column: RGB Image. 2nd column: Ground Truth. 3rd column: The inputs used to train the model: regions that are decent but incomplete approximations. 4th column: Stage 2 Results. When the regions are good approximations of the ground truth, the model performs well. For example, the sofa in the top row and sofa chair in the 3rd row are completed correctly. The exemplar based model allows for reasoning across occlusion boundaries. For example, the floors in the 3rd and 4th rows are correctly completed across occlusions. Competing regions, however, still lead to mistakes. For example, the towel in the second row and the floor in the 5th row are incorrectly completed due to color similarities and reflective intensity boundaries, respectively.

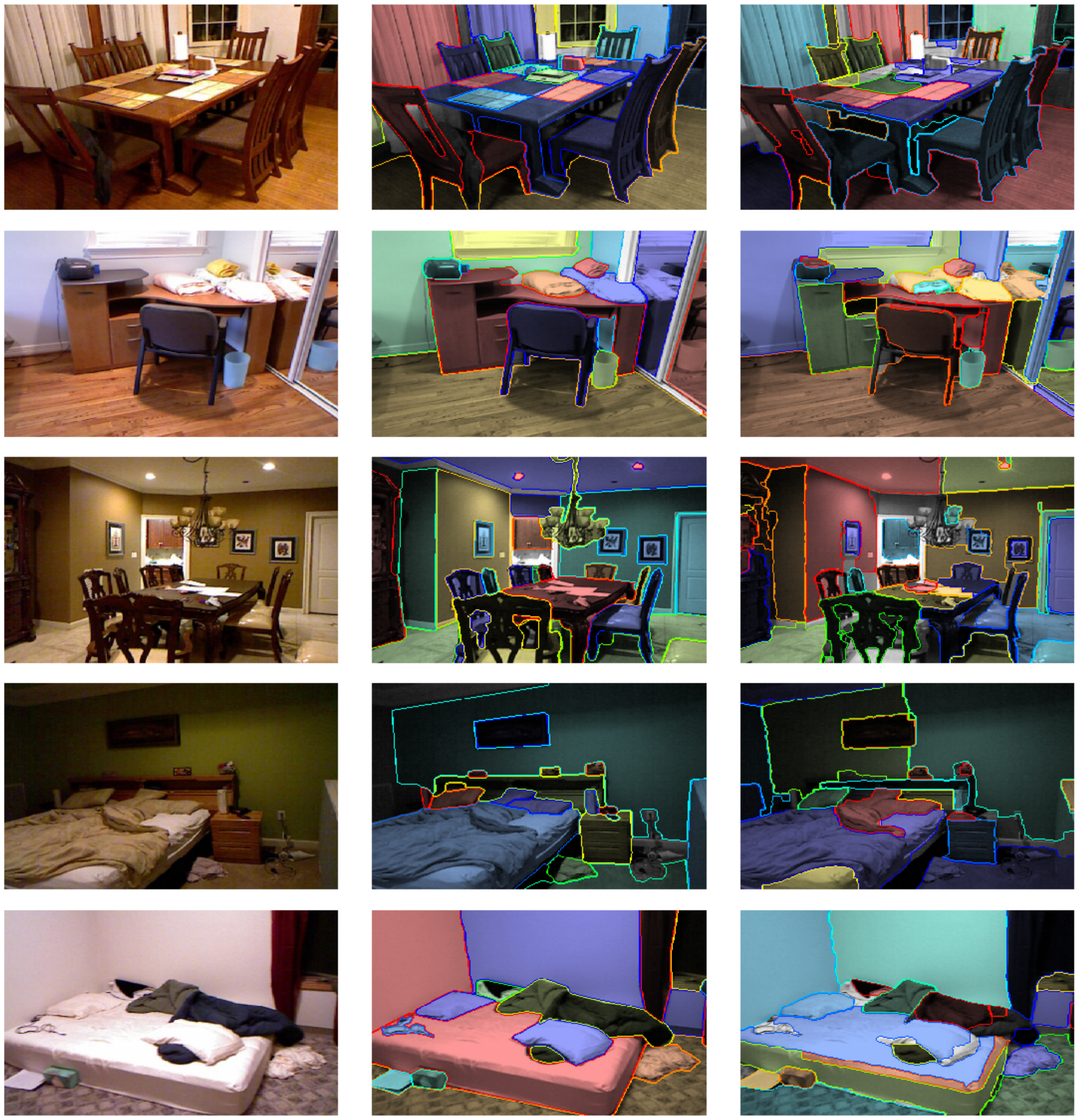


Figure 6.4: Visualizing Joint Exemplar Inference Results

Chapter 7

Inferring Physical Support Relations

7.1 Introduction

Traditional approaches to scene understanding aim to assign labels to each object in an image. However, this is an impoverished description since labels tell us little about the physical relationships between objects or possible actions that can be performed.

Many robotics and scene understanding applications require a physical parse of the scene into objects, surfaces, and their relations. A person walking into a room, for example, might want to find his coffee cup and favorite book, grab them, find a place to sit and then actually perform the actions of walking over, and sitting down. These tasks require parsing the scene into different objects and surfaces – the coffee cup must be distinguished from surrounding objects and the supporting surface for example. Some tasks also require understanding the interactions of scene elements: if the coffee cup is supported by the book, then the cup must be lifted first.

To this end, our goal is to provide such a physical scene parse: to segment visible regions into surfaces and objects and to infer their support relations. Our approach, illustrated in Fig. 7.1, is to first perform an instance segmentation of the scene and then to estimate the support relations of each object.

A strong prior on support relations exists via knowledge of an objects semantic class: a pillow generally is supported by a bed or sofa but very rarely is a bed or sofa supported by a pillow. While object detection is still not accurate enough to use fine grained classes, we introduce the concept of labeling objects with a set of *structural classes* that reflect their physical role in the scene: "ground"; "permanent structures" such as walls, ceilings, and columns; large "furniture" such as tables, dressers, and counters; and "props" which are easily movable objects. We show that these structural classes aid both segmentation and support estimation.

To reason about support, we introduce a principled approach that integrates physical constraints (e.g. is the object close to its putative supporting object?) and statistical priors on support relationships (e.g. mugs are often supported by tables, but rarely by walls). Our method is designed for real-world scenes that contain tens or hundred of objects with heavy occlusion and clutter. In this setting, interfaces between objects are often not visible and thus must be inferred. Even without occlusion, limited image resolution can make support ambiguous, necessitating global reasoning between image regions. Real-world images also contain significant variation in focal length. While wide-angle shots contain many objects, narrow-angle views can also be challenging as important structural elements of the scene, such as the floor, are not observed. Our scheme is able to handle these situations by inferring the location of invisible elements and how they interact with the visible components of the scene.

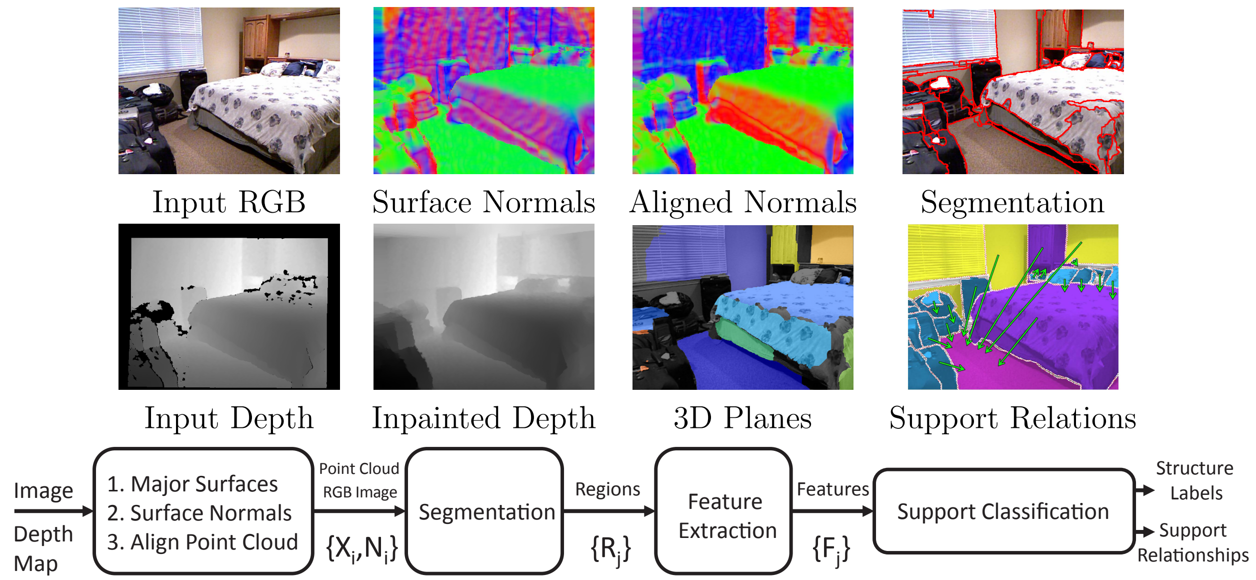


Figure 7.1: **Overview of algorithm.** Our algorithm flows from left to right. Given an input image with raw and inpainted depth maps, we compute surface normals and align them to the room by finding three dominant orthogonal directions. We then fit planes to the points using RANSAC and segment them based on depth and color gradients. Given the 3D scene structure and initial estimates of physical support, we then create a hierarchical segmentation and infer the support structure. In the surface normal images, the absolute value of the three normal directions is stored in the R, G, and B channels. The 3D planes are indicated by separate colors. Segmentation is indicated by red boundaries. Arrows point from the supported object to the surface that supports it.

7.2 Modeling Support Relationships

7.2.1 The Model

Given an image split into R regions, we denote by $S_i : i = 1..R$ the hidden variable representing a region's physical support relation. The basic assumption made by our model is that every region is either (a) supported by a region visible in the image plane, in which case $S_i \in \{1..R\}$, (b) supported by an object not visible in the image plane, $S_i = h$, or (c) requires no support indicating that the region is the ground itself, $S_i = g$. Additionally, let T_i encode whether region i is supported from below ($T_i = 0$) or supported from behind ($T_i = 1$).

When inferring support, prior knowledge of object types can be reliable predictors of the likelihoods of support relations. For example, it is unlikely that a piece of fruit is supporting a couch. However, rather that attempt to model support in terms of object classes, we model each region's *structure* class M_i, where M_i can take on one of the following values: Ground ($M_i = 1$), Furniture ($M_i = 2$), Prop ($M_i = 3$) or Structure ($M_i = 4$). We map each object in our densely labeled dataset to one of these four structure classes. Props are small objects that can be easily carried; furniture are large objects that cannot. Structure refers to non-floor parts of a room (walls, ceiling, columns). We map each object in our labeled dataset to one of these structure classes.

We want to infer the most probable joint assignment of support regions $\mathbf{S} = \{S_1, ...S_R\}$, support types $\mathbf{T} \in \{0,1\}^R$ and structure classes $\mathbf{M} \in \{1..4\}^R$. More formally,

$$\{\mathbf{S}^*, \mathbf{T}^*, \mathbf{M}^*\} = \arg\max_{\mathbf{S},\mathbf{T},\mathbf{M}} P(\mathbf{S}, \mathbf{T}, \mathbf{M}|I) = \arg\min_{\mathbf{S},\mathbf{T},\mathbf{M}} E(\mathbf{S}, \mathbf{T}, \mathbf{M}|I), \tag{7.1}$$

where $E(\mathbf{S}, \mathbf{T}, \mathbf{M}|I) = -\log P(\mathbf{S}, \mathbf{T}, \mathbf{M}|I)$ is the energy of the labeling. The posterior distribution of our model factorizes into likelihood and prior terms as

$$P(\mathbf{S}, \mathbf{T}, \mathbf{M}|I) \propto \prod_{i=1}^{R} P(I|S_i, T_i) P(I|M_i)\, P(\mathbf{S}, \mathbf{T}, \mathbf{M}) \tag{7.2}$$

to give the energy

$$E(\mathbf{S}, \mathbf{T}, \mathbf{M}) = -\sum_{i=1}^{R} \log(D_s(F^s_{i,S_i}|S_i, T_i) + \log(D_m(F^m_i|M_i)) + E_P(\mathbf{S}, \mathbf{T}, \mathbf{M}). \tag{7.3}$$

where F^S_{i,S_i} are the support features for regions i and S_i, and D_s is a Support Relation classifier trained to maximize $P(F^S_{i,S_i}|S_i, T_i)$. F^M_i are the structure features for region i and D_m is a Structure classifier trained to maximize $P(F^M_i|M_i)$. The specifics regarding training and choice of features for both classifiers are found in sections 7.2.3 and 7.2.4, respectively.

The **prior** E_P is composed of a number of different terms, and is formally defined as:

$$E_P(\mathbf{S}, \mathbf{T}, \mathbf{M}) = \sum_{i=1}^{R} \psi_{TC}(M_i, M_{S_i}, T_i) + \psi_{SC}(S_i, T_i) + \psi_{GC}(S_i, M_i) + \psi_{GGC}(\mathbf{M}). \tag{7.4}$$

The **transition** prior, ψ_{TC}, encodes the probability of regions belonging to different structure classes supporting each other. It takes the following form:

$$\psi_{TC}(M_i, M_{S_i}, T_i) \propto -\log \frac{\sum_{z \in supportLabels} \mathbb{1}[z = [M_i, M_{S_i}, T_i]]}{\sum_{z \in supportLabels} \mathbb{1}[z = [M_i, *, T_i]]} \tag{7.5}$$

The **support consistency** term, $\psi_{SC}(S_i, T_i)$, ensures that the supported and supporting regions are close to each other. Formally, it is defined as:

$$\psi_{SC}(S_i, T_i) = \begin{cases} (H_i^b - H_{S_i}^t)^2 & \text{if } T_i = 0, \\ V(i, S_i)^2 & \text{if } T_i = 1, \end{cases} \tag{7.6}$$

where H_i^b and $H_{S_i}^t$ are the lowest and highest points in 3D of region i and S_i respectively, as measured from the ground, and $V(i, S_i)$ is the minimum horizontal distance between regions i and S_i.

The **ground consistency** term ψ_{GC} encodes the hard constraint that region i does not require support ($S_i = g$) if and only if its structure class is 'ground':

$$\phi_{GC}(S_i, M_i) = \infty \Leftrightarrow S_i = g \quad \wedge \quad M_i \neq \text{`}ground\text{`} \tag{7.7}$$

The **global ground consistency** term $\psi_{GGC}(\mathbf{M})$ ensures that the region taking the floor label is lower than other regions in the scene. Formally, it is defined as:

$$\psi_{GGC}(\mathbf{M}) = \sum_{i=1}^{R} \sum_{j=1}^{R} \begin{cases} \kappa & \text{if } M_i = 1 \quad \wedge \quad H_i^b > H_j^b \\ 0 & \text{otherwise,} \end{cases} \tag{7.8}$$

7.2.2 Integer Program Formulation

The maximum a posteriori (MAP) inference problem defined in equation (7.1) can be formulated in terms of an integer program. This requires the introduction of boolean indicator variables to represent the different configurations of the unobserved variables **S**, **M** and **T**.

Let $R' = R+1$ be the total number of regions in the image plus a hidden region assignment. For each region i, let boolean variables $s_{i,j} : 1 \leq j \leq 2R' + 1$ represent both S_i and T_i as follows: $s_{i,j} : 1 < j \leq R'$ indicate that region i is supported from below ($T_i = 0$) by regions $\{1, ..., R, h\}$. Next, $s_{i,j} : R' < j \leq 2R'$ indicate that region i is supported from behind ($T_i = 1$) by regions $\{1, ..., R, h\}$. Finally, variable $s_{i,2R'+1}$ indicates whether or not region i is the ground ($S_i = g$).

Further, we will use boolean variables $m_{i,u} = 1$ to indicate that region i belongs to structure class u, and indicator variables $w_{i,j}^{u,v}$ to represent $s_{i,j} = 1$, $m_{i,u} = 1$ and $m_{j,v} = 1$. Using this over-complete representation we can formulate the MAP inference problem as an Integer Program using equations 7.9-7.15.

$$\arg\min_{\mathbf{s},\mathbf{m},\mathbf{w}} \quad \textstyle\sum_{i,j} \theta_{i,j}^{s} s_{i,j} + \sum_{i,u} \theta_{i,u}^{m} m_{i,u} \quad + \displaystyle\sum_{i,j,u,v} \theta_{i,j,u,v}^{w} w_{i,j}^{u,v} \tag{7.9}$$

$$\text{s.t.} \quad \textstyle\sum_{j} s_{i,j} = 1, \quad \sum_{u} m_{i,u} = 1 \quad \forall i \tag{7.10}$$

$$\textstyle\sum_{j,u,v} w_{i,j}^{u,v} = 1, \qquad \forall i \tag{7.11}$$

$$s_{i,2R'+1} = m_{i,1}, \qquad \forall i \tag{7.12}$$

$$\textstyle\sum_{u,v} w_{i,j}^{u,v} = s_{i,j}, \qquad \forall u, v \tag{7.13}$$

$$\textstyle\sum_{j,v} w_{i,j}^{u,v} \leq m_{i,u}, \qquad \forall i, u \tag{7.14}$$

$$s_{i,j},\ m_{i,u},\ w_{i,j}^{u,v} \in \{0,1\}, \qquad \forall i, j, u, v \tag{7.15}$$

The support likelihood D_s (eq. 7.3) and the support consistency ψ_{SC} (eq. 7.6) terms of the energy are encoded in the IP objective though coefficients $\theta^s_{i,j}$. The structure class likelihood D_m (eq. 7.3) and the global ground consistency ψ_{GGC} (eq. 7.8) terms are encoded in the objective through coefficients $\theta^m_{i,u}$. The transition prior ψ_{TC} (eq. 7.5) is encoded using the parameters $\theta^w_{i,j,u,v}$.

Constraints 7.10 and 7.11 ensure that each region is assigned a single support, type and structure label. Constraint 7.12 satisfies the Ground Consistency ϕ_{GC} term (eq. 7.7). Constraints 7.13 and 7.14 are marginalization and consistency constraints. Finally, constraint 7.15 ensure that all indicator variables take integral values. It is NP-hard to solve the integer program defined in equations 7.9-7.15. We reformulate the constraints as a linear program, which we solve using Gurobi's LP solver, by relaxing the integrality constraints 7.15 to:

$$s_{i,j},\ m_{i,u},\ w^{u,v}_{i,j} \in [0,1], \quad \forall i,j,u,v. \tag{7.16}$$

Fractional solutions are resolved by setting the most likely support, type and structure class to 1 and the remaining values to zero. In our experiments, we found this relaxation to be tight in that the duality gap was 0 in 1394/1449 images.

7.2.3 Support Features and Local Classification

Our support features capture individual and pairwise characteristics of regions. Such characteristics are not symmetric: feature vector $F^s_{i,j}$ would be used to determine whether i supports j but not vice versa. Geometrical features encode proximity and containment, e.g. whether one region contains another when projected onto the ground plane. Shape features are important for capturing characteristics of different supporting objects: objects that support others from below have large horizontal components and those that support from behind have large vertical components. Finally, location features capture the absolute 3d locations of the candidate objects. Since the support classifier must classify visible regions as well as a hidden region, we hallucinate hidden region features that would be extracted from a floor region not visible in the image plane. A full list of support features is found in Table 7.1.

Support Feature Descriptions	**Dims**
Geometry	**8**
G1. Minimum vertical and horizontal distance between the two volumes	2
G2. Absolute distance between the volumes' centroids	1
G3. Supported and supporting regions' heights above the ground	2
G4. Percentage of the supporting region that is farther from the viewer than the supported volume.	1
G5. Percentage of supported region contained inside convex hull of supporting region's projection onto the floor plane	1
G6. Percentage of supported region contained inside convex hull of supporting region's horizontal points when projected onto the floor plane	1
Shape	**7**
H1. Number and percentage of horizontal pixels in the supporting region	2
H2. Number and percentage of horizontal pixels in the supported region	2
H3. Number and percentage of vertical pixels in the supported region	2
H4. Chi-squared distance between histograms of each region's surface normals.	1
Region	**260**
E1. Ratio of number of pixels between the supported and supporting region	1
E2. Number of neighboring regions in the image plane for the supported region	1
E3. Whether or not the two regions are neighbors in the image plane	1
E4. Whether or not the region is hidden	1
Non-SIFT Structure-class features for the supported region	128
Non-SIFT Structure-class features for the supporting region	128

Table 7.1: **Support Features.** List of features using in classifying the support relationships between regions of the image.

To train D_s, a logistic regression classifier, each feature vector $F^S_{i,j}$ is paired with a label $Y^S \in \{1..4\}$ which indicates whether (1) i is supported from below by j, (2) i is supported from behind by j, (3) j represents the ground or (4) no relationship exists between the two regions. Predicting whether j is the ground is necessary for computing $D_s(S_i = g, T_i = 0; F^S_{i,g})$ such that $\sum_{S_i,T_i} D_s(S_i, T_i; F^S_{i,S_i})$ is a proper probability distribution.

7.2.4 Structure Class Features and Local Classification

Our structure class features are similar to those that have been used for object classification in previous works [**?**]. They include SIFT features, histograms of surface normals, 2D and 3D bounding box dimensions, color histograms [41] and relative depth (Section 4.2). A logistic regression classifier is trained to predict the correct structure class for each region of the image, and the output of the classifier is interpreted as probability for the likelihood term D_m. A full list of the features used is found in Table 7.2.

Structure Class Feature Descriptions	**Dims**
Color	**36**
C1: Color histograms: 10-bin histograms for each channel. [41]	30
C2: Mean and standard deviation of color channels	6
Shape	**1086**
A1: Sparse coded SIFT descriptor histograms	1000
A2: 2D Bounding box dimensions	2
A3: 3D Bounding box dimensions	3
A4: Pyramid of Surface normal histograms	78
A5: Mean, median, max of planar errors	3
Scene	**6**
N1: Distance to closest wall: absolute and normalized by room size	2
N2: Relative Depth: mean and variance relative depth over the region	2
N3: Height: minimum and maximum heights above the ground	2

Table 7.2: **Structure Class Features.** Used to classify each region of the image into one of four structure classes: Ground, Furniture, Prop and Structure.

7.3 Experiments

In our experiments, we use the NYU Depth V2 (Section 3.2.3) dataset as it contains both Depth information for every frame and support labels for pairs of regions. To perform instance segmentation, we use the greedy instance segmentation tree scheme introduced in Section 5.2.

7.3.1 Evaluating Support

Because the support labels are defined in terms of ground truth regions, we must map the relationships onto the segmented regions. To avoid penalizing the support inference for errors in the bottom up segmentation, the mapping is performed as follows: each support label from the ground truth region $[R_i^{GT}, R_j^{GT}, T]$ is replaced with a set of labels $[R_{a_1}^S, R_{b_1}^S, T] ... [R_{a_w}^S, R_{b_w}^S, T]$ where the overlap between supported regions (R_i^{GT},$R_{a_w}^S$) and supporting regions, (R_j^{GT},$R_{b_w}^S$) exceeds a threshold (.25).

We evaluate our support inference model against several baselines:

- **Image Plane Rules:** A Floor Classifier is trained in order to assign $S_i = g$ properly. For the remaining regions: if a region is completely surrounded by another region in the image plane, then a support-from-behind relationship is assigned to the pair with the smaller region as the supported region. Otherwise, for each candidate region, choose the region directly below it as its support from below.
- **Structure Class Rules:** A classifier is trained to predict each region's structure class. If a region is predicted to be a floor, $S_i = g$ is assigned. Regions predicted to be of Structure class Furniture or Structure are assigned the support of the nearest floor region. Finally, Props are assigned support from below by the region directly beneath them in the image plane.

- **Support Classifier:** For each region in the image, we infer the likelihood of support between it and every other region in the image using D_s and assign each region the most likely support relation indicated by the support classifier score.

The metric used for evaluation is the number of regions for which we predict a correct support divided by the total number of regions which have a support label. We also differentiate between *Type Agnostic* accuracy, in which we consider a predicted support relation correct regardless of whether the support type (below or from behind) matched the label and *Type Aware* accuracy in which only a prediction of the correct type is considered a correct support prediction. We also evaluate each method on both the ground truth regions and regions generated by the bottom up segmentation.

Results for support classification are listed in Table 7.3.1. When using the ground truth regions, the Image Plane Rules and Structure Class Rules perform well given their simplicity. Indeed, when using ground truth regions, the Structure Class Rules prove superior to the support classifier alone, demonstrating the usefulness of the Structure categories. However, both rule-based approaches cannot handle occlusion well nor are they particularly good at inferring the type of support involved. When considering the support type, our energy based model improves on the Structure Class Rules by 9% and 17% when using the ground truth and segmented regions, respectively, demonstrating the need to take into account a combination of global reasoning and discriminative inference.

Visual examples are shown in Fig 7.4. They demonstrate that many objects, such as the right dresser in the row3, column 3 and the chairs in row 5, column 1, are supported by regions that are far from them in the image plane, necessitating non-local inference. One of the main stumbling blocks of the algorithm is incorrect floor classification as show in the 3rd image of the last row. Incorrectly labeling the rug as the floor creates a cascade of errors since the walls and bed rely on this as support rather than using the true floor. Additionally, incorrect

Predicting Support Relationships					
Region Source	Ground Truth			Segmentation	
Algorithm	Type Agnostic	Type Aware		Type Agnostic	Type Aware
Image Plane Rules	63.9	50.7		22.1	19.4
Structure Class Rules	72.0	57.7		45.8	41.4
Support Classifier	70.1	63.4		45.8	37.1
Energy Min (LP)	75.9	72.6		55.1	54.5

Table 7.3: Results of the various approaches to support inference. Accuracy is measured by total regions whose support is correctly inferred divided by the number of labeled regions. Type Aware accuracy penalized incorrect support type and Type Agnostic does not.

structure class prediction can lead to incorrect support inference, such as the objects on the table in row 4, column 1.

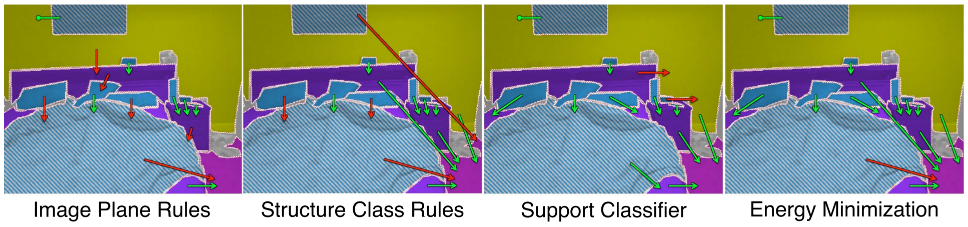


Figure 7.2: Comparison of support inference algorithms. Image Plane Rules incorrectly assigns many support relationships. Structure Class Rules corrects several support relationships for Furniture objects but struggles with Props. The Support classifier corrects several of the Props but infers an implausible Furniture support. Finally, our LP solution correctly assigns most of the support relationships. ($\rightarrow$: support from below, $\multimap$: support from behind, $+$: support from hidden region. Correct support predictions in green, incorrect in red. Ground in pink, Furniture in Purple, Props in Blue, Structure in Yellow, Grey indicates missing structure class label. Incorrect structure predictions are striped.)

7.3.2 Evaluating Structure Class Prediction

To evaluate the structure class prediction, we calculate both the overall accuracy and the mean diagonal of the confusion matrix. As 7.3 indicates, the LP solution makes a small improvement over the local structure class prediction. Structure class accuracy often struggles when the depth values are noisy or when the segmentation incorrectly merges two regions of different structure class.

Predicting Structure Classes					
	Overall			Mean Class	
Algorithm	G. T.	Seg.		G. T.	Seg.
Classifier	79.9	58.7		79.2	59.0
Energy Min (LP)	80.3	58.6		80.3	59.6

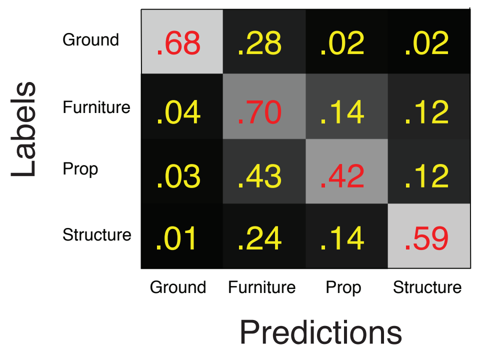


Figure 7.3: Accuracy of the structure class recognition.

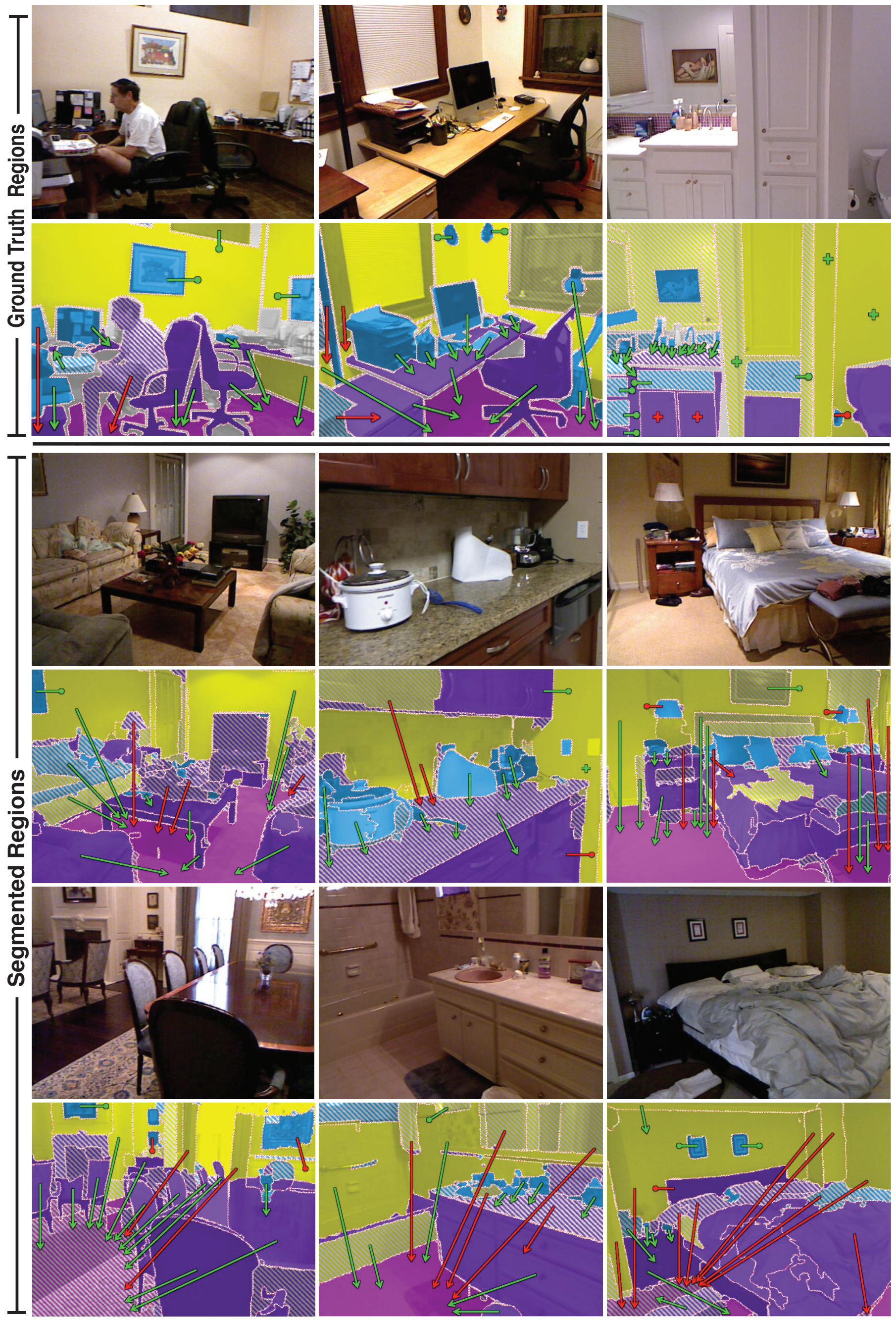


Figure 7.4: Examples of support and structure class inference with the LP solution. $\rightarrow$: support from below, $\multimap$: support from behind, $+$: support from hidden region. Correct support predictions in green, incorrect in red. Ground in pink, Furniture in Purple, Props in Blue, Structure in Yellow, Grey indicates missing structure class label. Incorrect structure predictions are striped.

Chapter 8

Extent Reasoning

8.1 Introduction

In addition to reason about support, an instance segmentation can also be used to reason about the true 3D extent of objects in scenes. This task has traditionally been cast as the problem of dense 3D reconstructions of scenes. Indeed the adoption of inexpensive 3D cameras has led to much progress in recent years. The combined use of depth and colour signals has been successfully demonstrated in the production of large-scale models of indoor scenes via both offline [?] and online [?] algorithms. Most RGB+D reconstruction methods require data that show the scene from a multitude of viewpoints and are not well suited for input sequences which contain a single-view or limited number of viewpoints. Moreover, these reconstruction methods are hindered by occlusion as they make no effort to infer the geometry of parts of the scene that are not visible in the input sequences. Consequently, the resulting 3D models often contain gaps or holes and do not capture certain basic elements of a scene, such as the true extent and shape of objects and scene surfaces.

Accurate and complete surface reconstruction is of special importance in Augmented Reality

(AR) applications which are increasingly being used for both entertainment and commerce. For example, a recently introduced gaming platform [?] asks users to scan an interior scene from multiple angles. Using the densely reconstructed model, the platform overlays graphically generated characters and gaming elements. Furniture retailers (such as IKEA) enable customers to visualize how their furniture will look when installed without having to leave their homes. These applications often require a high fidelity dense reconstruction so that simulated physical phenomenon, such as lighting, shadow and object interactions (e.g. collisions) can be produced in a plausible fashion. Unfortunately, such reconstructions often require considerable effort on the part of the user. Applications either demand that users provide video capture of sufficient viewpoint diversity or operate using an incomplete model of the scene.

Our goal is to complement the surface reconstruction for an input sequence that is limited in viewpoint, and "fill in" parts of the scene that are occluded or not visible to the camera. This goal is driven by both a theoretical motivation and a practical application. Firstly, basic scene understanding requires high level knowledge of how objects interact and extend in 3D spaces. While most scene understanding research is concerned with semantics and pixel labeling, relatively little work has gone into inferring object or surface extent, despite the prevalence and elemental nature of this faculty in humans. Second, online surface reconstruction pipelines such as KinectFusion [?] are highly suitable for AR applications, and could benefit from a scene completion phase, integrated into the pipeline.

Our approach assumes a partial dense reconstruction of the scene that is represented as a voxel grid where each voxel can be occupied, free or its state in unknown. We use the KinectFusion [?] method to compute this reconstruction which also assigns a surface normal and truncated signed distance function (TSDF) [?] to the voxels. Given this input, our method first detects planar surfaces in the scene and classifies each one as being part of the scene

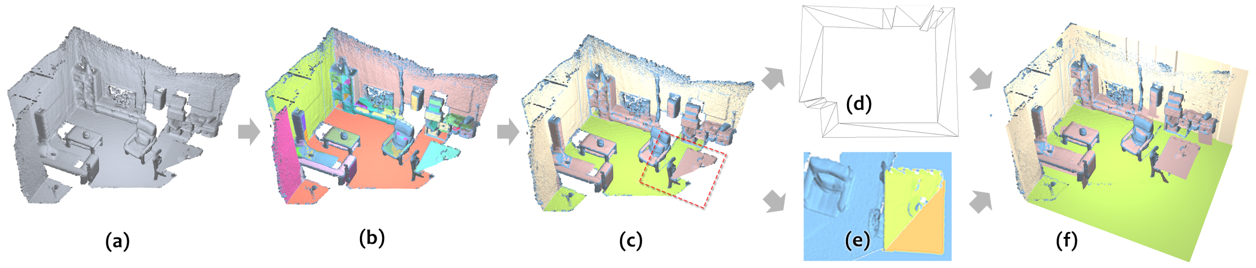


Figure 8.1: Overview of the pipeline: Given a dense but incomplete reconstruction (a) we first detect planar regions (b) and classify them into ceiling (not shown), walls (tan), floor (green) and interior surfaces (pink) (c) Non-planar regions are shown in light blue. Next, we infer the scene boundaries(d) and shape of partially occluded interior objects, such as the dining table (e) to produce a complete reconstructed scene (f).

layout (floor, walls ceiling) or part of an internal object. We then use the identities of the planes to extend them by solving a labeling problem using our Contour-completion random field (CCRF) model. Unlike pairwise Markov random fields, which essentially encourage short boundaries, the CCRF model encourages the discontinuities in the labeling to follow detected contour primitives such as lines or curves. We use this model to complete both the floor map for the scene and estimate the extents of planar objects in the room. This provides us with a watertight 3D model of the room. Finally we augment the original input volume account for our extended and filled scene. The stages of our algorithm are demonstrated in figure 8.1.

8.2 Plane Detection and Classification

We now describe how our method detects the dominant planes from the partial voxel based reconstruction. We denote the space of all possible 3D planes by $\mathcal{H}$, and the set of planes present in the scene by H. Let the set of all 3D points visible in the scene be denoted by $\mathcal{P} = \{p_1, ...p_N\}$. We estimate the most probable labeling for H by minimizing the following energy function:

$$H* = \arg\min_{H \subset \mathcal{H}} \sum_{i=1}^{N} f_i(H) + \lambda |H| \tag{8.1}$$

where λ is a penalty on the number of planes and f_i is a function that penalizes the number of points not explained by any plane:

$$f_i(H) = \min\{\min_{h \in H} [\delta(p_i, h), \lambda_b]\} \tag{8.2}$$

where the function δ returns a value of 0 if point p_i falls on plane h and is infinity otherwise. Minimizing the first term alone has the trivial degenerate solution where we include a plane for every point p_i in the set H. However, this situation is avoided by the second term of the energy which acts as a regularizer and adds a penalty that linearly increases with the cardinality of H.

Lemma 1. *The energy function defined in equation (8.1) is a supermodular set function.*

Proof. A set function F is super-modular if it satisfies the diminishing reduction property. Formally,

$$F(A) - F(A \cup \{a\}) \leq F(B) - F(B \cup \{a\}) \;\; \forall B \subset A \tag{8.3}$$

Substituting (8.1), we see the that cardinality term evaluates to 1 on both sides and cancels leaving us

$$\sum_i f_i(A) - \sum_i f_i(A \cup \{a\}) \leq \sum_i f_i(B) - \sum_i f_i(B \cup \{a\}) \tag{8.4}$$

The LHS counts the number of 3D points that did not fall in planes in A but fall on the new plane a, while the RHS counts the number of 3D points that did not fall in planes in B but fall on the new plane a. Since $B \subset A$, LHS lower bounds the RHS. $\square$

8.2.1 Computing the Optimal H

Minimization of a super-modular function is an NP-hard problem even when the set of possible elements is finite (which is not true in our case). We employ a greedy strategy, starting from an empty set and repeatedly adding the element that leads to the greatest energy reduction. This method has been observed to achieve a good approximation [**?**]. We begin by using the Hough transform [**?**] to select a finite set of planes. In our method, each 3D point and its surface normal votes for a plane equation parameterized by its azimuth θ, elevation ψ and distance from the origin ρ. Each of these votes is accrued in an accumulator matrix of size $A \times E \times D$ where A is the number of azimuth bins, E is the number of elevation bins and D is the number of distance bins [1]. After each point has voted, we run non-maximal suppression to avoid accepting multiple planes that are too similar.

Once we have a set of candidate planes we sort them in descending order by the number of votes they have received and iteratively associate points to each plane. A point can be associated to a plane if it has not been previously associated to any other plane and if its planar disparity and local surface normal difference are small enough [2]. As an additional heuristic, each new plane and its associated points are broken into a set of connected components ensuring that planes are locally connected.

8.2.2 Semantic Labeling

Once we have a set of planes, we classify each one independantly into one of four semantic classes: Floor, Wall, Ceiling and Internal. To do so, we train a Random Forest Classifier to predict each plane's class based on a set of 3D-only features (Table 5.1), which capture

[1]We use A=128, E=64 and D is found dynamically by spacing bin edges of size 5cm apart between the max and minimum points

[2]Planar disparity threshold=.1, angular disparity threshold = .1

attributes of each plane including its height in the room, size and surface normal distribution. Planes classified as one of Floor, Wall and Ceiling will be used for inferring the floor plan and scene boundaries (section 8.3.6), whereas Internal planes will be extended and filled in a subsequent step (section 8.3.7).

8.3 Scene Completion

Given the set of detected and classified planes we infer the true extent of the scene, *ie.* obtain a water-tight room structure, and extend interior planes based on evidence from the scene itself.

8.3.1 Completion as a Labeling Problem

We now describe how to estimate the boundaries of planes as seen from a top-down view. We formulate boundary completion as a pixel labeling problem. Consider a set $\mathcal{S}$ of nodes that represent grid locations in the top-down view of the scene. We assume that a partial labeling of nodes $i \in \mathcal{S}$ in the grid can be observed and is encoded by variables $y_i; i \in \mathcal{S}$ where $y_i = 1$, $y_i = 0$ and $y_i = -1$ represent that i belongs to the plane, does not belong to the plane, and its membership is uncertain respectively. Given $\mathbf{y}$, we want to estimate the true extent of the plane which we denote by $\mathbf{x}$. Specifically, we will use the binary variable x_i to encode whether the plane covers the location of node i in the top-view. $x_i = 1$ represents that node i belongs to the plane while $x_1 = 0$ represents that it does not.

The traditional approach for pixel labeling problems is to use a pairwise Markov Random Field (MRF) model. The energy of any labeling $\mathbf{y}$ under the pairwise MRF model is defined as:

$$E(\mathbf{x}) = \sum_{i \in \mathcal{S}} \phi_i(x_i) + \sum_{ij \in \mathcal{N}} \phi_{ij}(x_i, x_j) \tag{8.5}$$

where ϕ_i encode the cost of assigning a label x_i and ϕ_{ij} are pairwise potentials that encourage neighboring ($\mathcal{N}$) nodes to take the same label, and K is a constant. The unary potential functions force the estimated labels $\mathbf{x}$ to be consistent with the observations $\mathbf{y}$, *ie.* $\phi_i(x_i) =$ inf if $y_i \neq -1$ and $x_i \neq y_i$, and $\phi_i(y_i) = 0$ for all other cases, while the pairwise potentials take the form an Ising model. The Maximum a Posteriori (MAP) labeling under the model can be computed in polynomial time using graph cuts. However, the results are underwhelming as the pairwise model does not encode any information about how boundaries should be completed. It simply encourages a labeling that has a small number of discontinuities.

8.3.2 Contour Completion Random Field

Unlike the standard MRF which penalizes the number of transitions in the labeling, our Contour Completion Random Field (CCRF) model adds a penalty based on the least number of curve primitives that can explain all the transitions. We implement this by introducing higher order potentials in the model. These potentials are defined over overlapping sets of edges where each set follows some simple (low-dimensional) primitive curve shape such as a line or a circle. Formally, the energy function for the CCRF model can be written as:

$$E(\mathbf{x}) = \sum_{i \in \mathcal{S}} \phi_i(x_i) + \sum_{g \in \mathcal{G}} \Psi_g(\mathbf{x}) \tag{8.6}$$

where Ψ_g are our curve completion potentials, and $\mathcal{G}$ is a set where each curve g represents a set of nodes (edges) that follow a curve. The curve completion potentials have a diminishing returns property. More formally,

$$\Psi_g(\mathbf{x}) = F\Big(\sum_{ij\in\mathcal{E}_g} \psi_{ij}(x_i, x_j)\Big), \tag{8.7}$$

where $\mathcal{E}_g$ is the set of edges that defines the curve or edge group g. F is a non-decreasing concave function. In our experiments, we defined F as an upper-bounded linear function *ie.* $F(t) = \min\{\lambda * t, \theta\}$ where λ is the slope of the function and θ is the upper-bound. It can be seen that once a few edges are cut $t \geq \frac{\theta}{\lambda}$, the rest of the edges in the group can be cut without any penalty. This behavior of the model does not prevent the boundary in the labeling from including large number of edges as long as they belong to the same group (curve). The exact nature of these groups are described below.

8.3.3 Defining Edge Groups

We consider two types of edge groups: straight lines and parabolas. While previous work has demonstrated the ability of the hough transform [?] to detect other shapes, such as circles and ellipses, such high parameter shapes require substantially more memory and computation and we found lines and parabolas sufficiently flexible to capture most of the cases we encountered.

Detecting Lines To detect lines, we used a modified Hough transform to not only detect lines in the image, but also the direction of the transition (the plane to free space or vice-versa). We use an accumulator with 3 parameters: ρ, the distance from the origin to the line, θ, the angle between the vector from the origin to the line and the X axis, and a quaternary variable d, which indicates the direction of the transition (both bottom-top and left-right directions) [3]. Following the accumulation of votes, we run non-maximal suppression and create an edge group for each resulting line.

Detecting Parabolas The standard Hough transform for parabolas requires 4 parameters. To avoid the computational and memory demands of such a design, we introduce a novel and simple heuristic illustrated in figure 8.2. First, we identify each point in the input image (8.2(a)) which falls at the intersection of free space, occupied and unknown pixels. We refer to these intersections as *contact points*. Furthermore, we will refer to all pixels occupied by a plane and bordering free space as *contour points* (see figure 8.2(b)).

To close an occluded contour, a parabola either connects a set of contact points or continues at a contact point until it reaches the end of the image. We note that only contact points that border the same occlusion region can possibly be bridged. Therefore, we create a set of contact point pairs ξ from which parabolas will be estimated. If multiple contact points all border the same occlusion region, we just pair each contact point with its nearest neighbor. For each pair of contact points, we fit a parabola to all of the contour points that immediately border each contact point. Because the parabola may be rotated, we first rotate these bordering contour points so that the normal of the line joining the contact points is aligned with the Y-axis (8.2(c)). To avoid over or under fitting the contour, we sample the contour points using several radii and keep the parabola most consistent with the observed contour (8.2(d)-(f)). If a contact point cannot be paired, or if a pair provides only poor fits, a parabola is fit using its immediate contour point neighbors for each side separately. This algorithm is summarized in Algorithm 1.

[3]We use 400 angular bins for θ and evenly spaced bins for ρ 1 unit apart. The minimum number of votes allowed was set to 10.

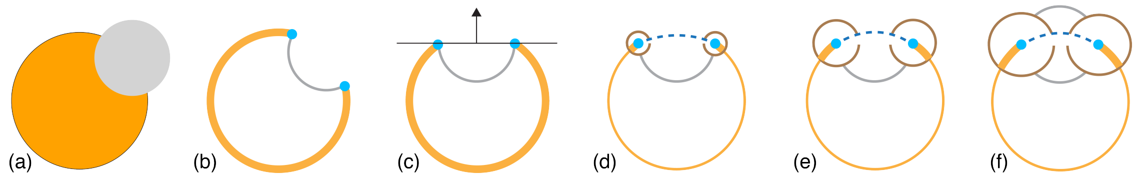


Figure 8.2: Our method for fast parabola fitting, as described in section 8.3.3. Starting with an input image (a), we find the contact points (b) labeled in blue and contour edges in orange. Next, we align the contact points (c) and fit parabola using several radii (d-f) to find the best fit.

```
Data: Ternery Image Y, set of search radii $\mathcal{T}$
Result: List of Edge Groups $G$
C = FindContactPoints(Y);
$\xi$ = FindPairsOfContactPoints(C, Y);
for $v_i, v_j \in \xi$ do
    Initialize vote vector to all zeros;
    $\theta_\tau = 0 \; \forall \, \tau \in |\mathcal{T}|$;
    for $\tau \in \mathcal{T}$ do
        P = FindNeighboringContourPoints($v_i$, $v_j$,$\tau$);
        P' = RotatePoints(P);
        $\alpha$ = LeastSquaresFit(P');
        Q' = sampleParabola($\alpha$);
        Q = rotateBackToImageCoordinates(Q);
        g = defineEdgeGroup(Q);
        $\theta_\tau$ = CountVotesFromEdgeGroup(g, Y);
    end
    Append(G, BestParabola($\theta$));
end
```

Algorithm 1: Fast Parabola Fitting Algorithm

8.3.4 Hierarchical Edge Groups

While using detected lines or curves may encourage the correct surface boundaries to be inferred in many cases, in others, there is no evidence present in the image of how a shape should be completed. For example see the right side of the shape in figure 8.3 and the synthetic examples in figure 8.4(b). Motivated by the gestalt laws of perceptual grouping, we attempt to add edge groups whose use in completion would help provide for shapes that exhibited simple closure and symmetry. More specifically, for each observed line detected,

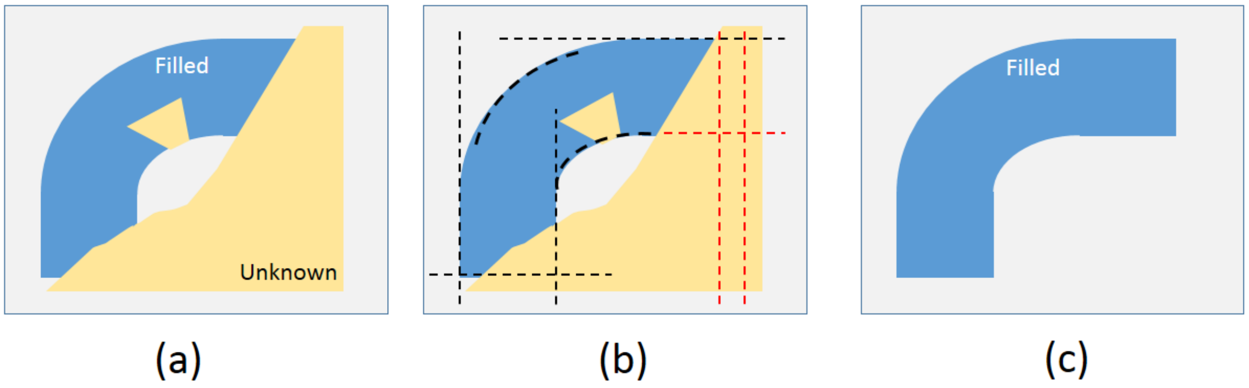


Figure 8.3: Contour Completion Random Field: (a) A top-down view of a partially occluded plane (b) We detect lines and parabolas along the contour of the known pixels (stippled black lines), and hallucinate parallel lines (in red) (c) We apply CCRF inference to extend the plane.

we add additional parallel edge groups on the occluded side of the shape.

It is clear that defining edge groups that completely cover another edge group would lead to double counting. To prevent this, we modify the formulation to ensure that only one group from each hierarchy of edge groups is counted. To be precise, our CCRF model allows edge groups to be organized hierarchically so that a set of possible edges have a parent and only a single child per parent may be active. This formulation is formalised as:

$$E(\mathbf{x}) = \sum_{i \in \mathcal{S}} \phi_i(x_i) + \sum_{g \in \mathcal{G}} \min_{k \in c(g)} \Psi_k(\mathbf{x}) \tag{8.8}$$

where $c(g)$ denotes the set of child edge groups for each parent g.

To summarize, our edge groups are obtained by fitting lines and parabolas to the input image thus encouraging transitions that are consistent with these edges. As indicated in Equation 8.8, not all edge groups can be active simultaneously and in particular, any line used to hallucinate a series of edges is considered the parent to its child hallucinated lines. Consequently, we constrain only a single hallucinated line to be active (at most) at a time.

8.3.5 Inference with Hierarchical Edge Groups

Inference under higher order potentials defined over edges groups was recently shown to be NP-hard even for binary random variables by Jegelka *et al.* [?]. They proposed a special purpose iterative algorithm for performing approximate inference in this model. Later, Kohli *et al.* [?] proposed an approximate method that could deal with multi-label variables. Their method transformed edge group potentials into a sum of pairwise potentials with the addition of auxiliary variables that allowed the use of standard inference method like graph cuts. However, both these algorithms are unsuitable for CCRF because of the special hierarchical structure defined over our edge groups.

Inspired from [?], we transformed the higher-order curve completion potential (8.7) to the following pairwise form:

$$\Psi_g^p(\mathbf{x}) = T + \min_{h_g,\mathbf{z}} \big\{ \sum_{ij\in\mathcal{E}_g} \theta_{ij}((x_i + x_j - 2z_{ij})h_g - 2(x_i + x_j)z_{ij} + 4z_{ij}) - Th_g \big\}. \tag{8.9}$$

where h_g is the binary auxiliary corresponding to the group g, and $z_{ij}, \forall ij \in \mathcal{E}_g$ are binary auxiliary variables corresponding to the edges that constitute the edge group g. However, this transformation deviates from the energy of the hierarchical CCRF (equation 8.8) as it allows multiple edge groups in the hierarchy to be all active at once.

To enure that only one edge group is active in each edge group hierarchy, we introduce a series of constraints on the binary auxiliary variables corresponding to the edge groups. More formally, we minimize the following energy :

$$E(\mathbf{x},\mathbf{h}) = \sum_{i\in\mathcal{S}} \phi_i(x_i) + \sum_{g\in\mathcal{G}} \Psi_g^p(\mathbf{x}) \qquad \text{s.t.}\forall g, \qquad \sum_{k\in c(g)} h_k \leq 1 \tag{8.10}$$

where $c(g)$ denotes the set of child edge groups for each parent edge group g. The minimum energy configuration of this formulation is equivalent to that of hierarchical CCRF (equation 8.8).

In order to find the MAP solution, we now need to minimize the constrained pairwise energy function (equation 8.10). We observe that on fixing the values of the group auxiliary variable (h's) the resulting energy becomes unconstrained and submodular, and thus, can be minimized using graph cuts. We use this observation to do inference by exhaustively searching over the space of edge group auxiliary variables and minimizing the rest of the energy using graph cuts. However, we can make the algorithm even more efficient by not allowing the activity of a child edge group to be explored if its parent is not already active. In other words, we start by exhaustively searching over the auxiliary variables of the parent edge groups (at the top of the hierarchy), and if a group variable is found to be active, we check if its child variables can be made active instead of it.

8.3.6 Inferring Scene Boundaries

To extend and fill the scene boundaries, we begin by projecting the free space of the input TSDF and the Wall planes (predicted by our classifier) onto the floor plane. Given a 2D point cloud induced by these projections, we discretize the points to form a projection image illustrated by figure 8.3 where each pixel y_i takes on the value of free space, wall or unknown. To infer the full scene layout, we apply the CCRF (Equation 8.6) to infer the values of the unknown pixels. In this case, we consider free space to be the area to be expanded ($y_i = 1$) and the walls to be the surrounding area to avoid being filled ($y_i = 0$). We first detect the lines and curves of the walls to create a series of edge groups. Next, we set $\phi_i(x_i = 1) = \infty$ if $y_i = 0$ and $\phi_i(x_i = 0) = \infty$ if $y_i = 1$. Finally, we add a slight bias [**?**] to assigning free space $\phi_i(x_i = 0) = \epsilon$ [4].

[4] ϵ=1e-6

8.3.7 Extending Planar Surfaces

Once the scene boundary has been completed, we infer the full extent of internal planar surfaces. For each internal plane, we project the TSDF onto the detected 2D plane as follows. First we find a coordinate basis for the plane using PCA and estimate the major and and minor axes of the plane, M and N, respectively. Next, we create an image of size $2N+1 \times 2M+1$ where the center pixel of the image corresponds to the centroid of the plane. We sample a grid along the plane basis of size $2N + 1 \times 2M + 1$ where the TSDF values sampled in each grid location are used to assign each of the image's pixels. If the sampled TSDF value is occupied, y_i is set to 1, if its free space y_i is set to 0 and if its unknown, y_i is set to -1. In practice, we also sample several voxels *away* from the plane (along the surface normal direction). This heuristic has the effect of reducing the effects of sensor noise and error from plane fitting.

Once Y has been created, we detect all lines and parabolas in the image and hallucinate the necessary lines to create our edge groups. Next, we assign the local potentials in the same manner as described in Section 8.3.6.

8.4 Augmenting the Original Volume

The result of our scene completion is a water-tight scene boundary, and extended interior planes. As the final step in our pipeline we augment the original TSDF we imported. For the scene boundary we simplify the resutling polyline representing the boundary, and sample points along this boundary from floor to ceiling height. For the interior planes we sample points in the extended parts of the planes. For each sampled point (sampled densely as required, in our case γ) we traverse a bresenham-line in the volume from the voxel closest to the point, and in two directions, its normal and the inverse to its normal. For each encountered voxel, we update the TSDF value with the distance to the surface. If the dimensions of the original volume do not suffice to hold the new scene boundaries, we create a new larger volume and copy the original TSDF to it, before augmenting it.

The augmented TSDF, in the originating surface reconstruction pipeline, is continuously updated with new evidence (e.g. as the user moves). Augmented areas are phased out as the voxels which they filled become known.

8.5 Experiments

We evaluate our method in several ways. First, we demonstrate the effectiveness of the CCRF model by comparing it to several baselines for performing in-painting of binary images. Second, we compare our results to a recently introduced method [**?**] that performs extent-reasoning. Third, we demonstrate qualitative results on a newly collected set of indoor scenes.

8.5.1 Synthetic validation

We generated a dataset of synthetic images inspired by shapes of planar surfaces commonly found in indoor scenes. The dataset contained 16 prototypical shapes meant to resemble objects like desks, conference tables, beds, kitchen counters, etc. The set of 16 protypes were first divided into training and test sets. Each of the 8 training and testing prototypes were then randomly rotated and occluded 10 times resulting in 80 total training and 80 test images.

Since we are primarily concerned with how these methods perform in predicting the boundaries of the input images, we use the evaluation metric of [22] in which we only evaluate the correctness of pixels immediately near the boundary areas. We computed evaluation scores for various widths of the evaluation region. Quantitative results can be found in Figure 8.4. We compare against several baseline methods that are commonly used for in-painting binary images. These include both 4 connected and 8 connected graph cuts and a non-parametric patch-matching algorithm for in-painting. The ideal parameters for each algorithm were fine tuned on the training set and then applied to the test set.

8.5.2 Single Frame Scene Completion

We compare our approach to the work of Ruiqi and Hoiem [?] to demonstrate its applicability to single frame scene completion. While we produce a binary label indicating the voxels that are occupied, [?] output a heat map. To perform a fair comparison, we followed the following steps. For each frame of the NYU V2 dataset we computed the filled voxels using our regular pipeline. Since KinectFusion was not designed for single image inputs, a number of the images failed to be fused and were ignored during evaluation. For those test images that succeeded in being fused by Kinect Fusion, we used the same metric as in [?] and report the accuracy of [?] using the same false positive rate as in our method. The quantitative results in Table 8.5.2 demonstrate that while our single frame scene completion ability is not as strong as [?], our scheme does not have the benefit of using RGB data, whereas [?] do. In particular, our method breaks down in single frame completion when a sizable portion of the depth frame is missing due to noise or surfaces with low albedo. In these cases, we do not detect any plane and do not fill that area. Nevertheless, even with a single frame, our method is still very capable of inferring reasonable boundaries for the rooms, extending the beds up to the wall, and correctly inferring the size of the sofa as shown in Figure 8.5.

Algorithm	Accuracy
Ruiqi and Hoiem [?]	62.6
Our method	60.6

Table 8.1: Evaluating Single Frame Scene Completion

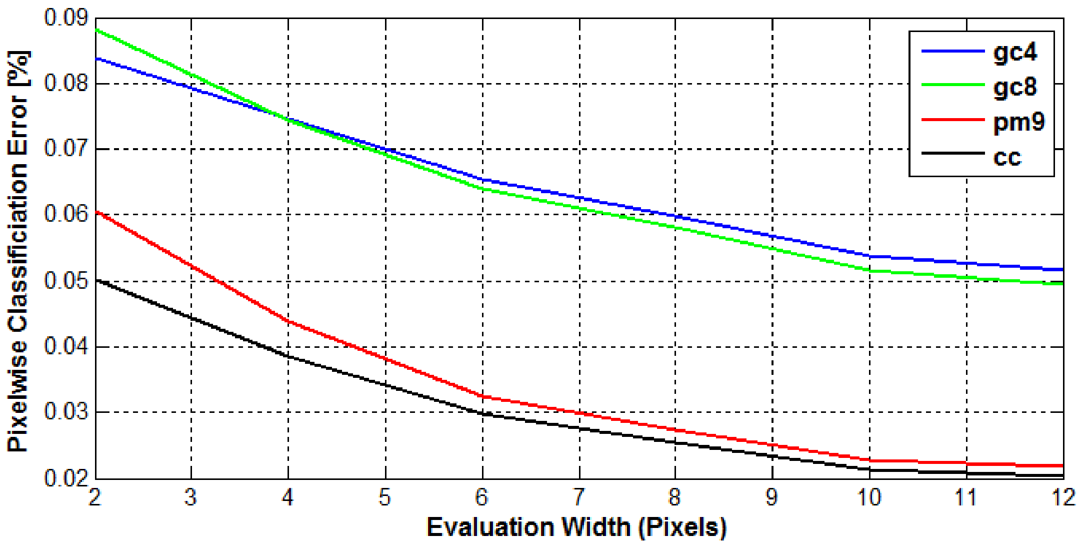
8.5.3 Qualitative Analysis

Using a Kinect Sensor attached to a notebook computer, we captured more than 100 scenes. Each scene is a collection of color and depth frames ranging from hundreds to thousands of frames. Each of these sequences was input into the KinectFusion pipeline, where for a large number of scenes we performed multiple reconstructions, sharing a starting frame, but varying in number of frames processed. This gave us a baseline for a qualitative and progressive evaluation of our results.

For each such scene we were able to evaluate the original augmented and original reconstruction at each step. As more evidence is revealed, extended planes are replaced with their physical counterparts, and the scene boundaries are updated to reflect reality. An example of this progression can be seen in Figure 8.6. Further results are found in Figure 8.8(top row).

We implemented a complete AR system in the Unity 3D framework, in which a user is able to navigate a captured scene, place virtual objects on horizontal supporting planes, and throw balls which bounce around the scene. As seen in figure 8.7, with scene completion, we are able to place figures on occluded planes, and bounce balls realistically off completed surfaces.

Failure cases may occur in different stages along our pipeline. Failure to detect or correctly identify planes will result in the plane not being extended or not being used in the boundary extent inference step. In some cases we received faulty volumetric information from the Kinect sensor, mostly due to reflectance and transparencies in the scene. This causes both eroneous inputs and also noisy contours which may hamper the CCRF from finding a good line or parabola fit. Examples can be seen in the bottom row of Figure 8.8.



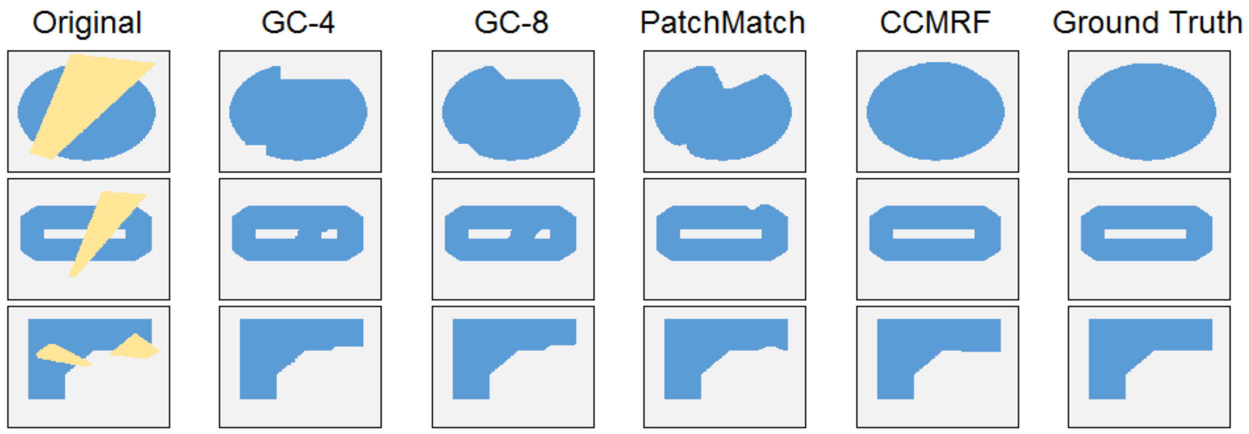


Figure 8.4: (Top) Pixel-wise classification error on our synthetic dataset. The plot demonstrates performance as a function of the width of the evaluation region. (Bottom) Some examples from our dataset of synthetic images showing classification results for occluded pixels (yellow) using four methods.

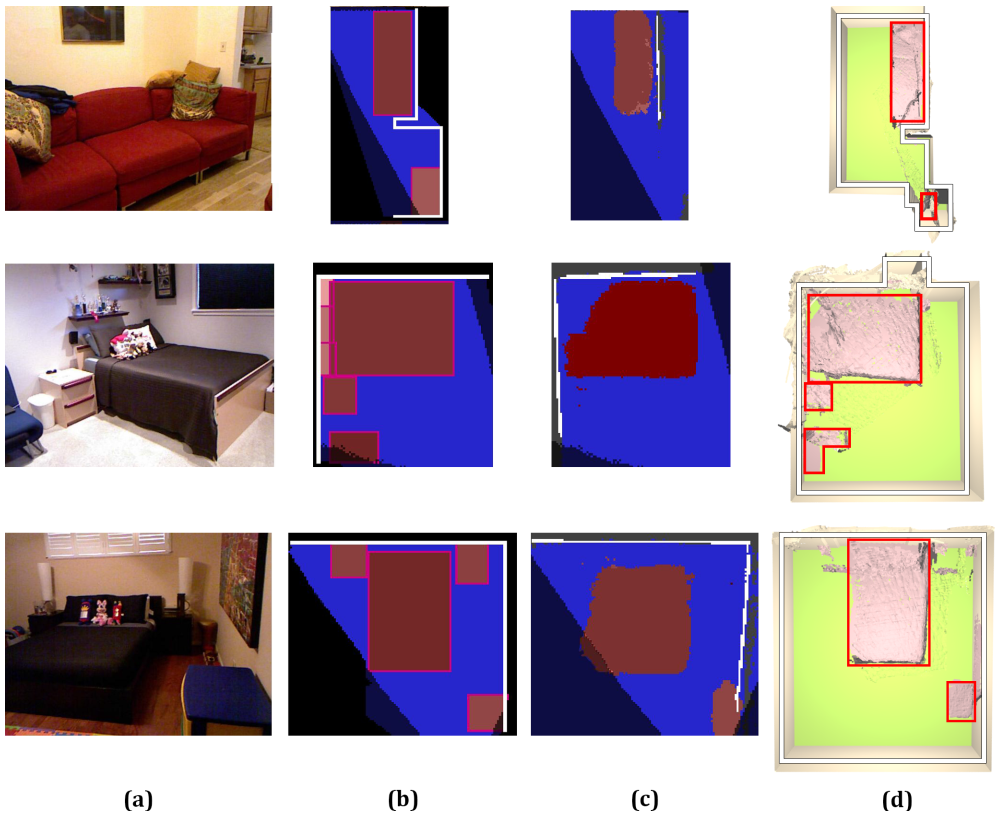


Figure 8.5: Comparison with [?]: (a) Frames from the NYU2 dataset (b) Ground truth support extent (c) Predicted extent from [?] (d) A top down view of our completed scene where inferred scene boundaries are highlighted in white and extended interior objects are highlighted in red.

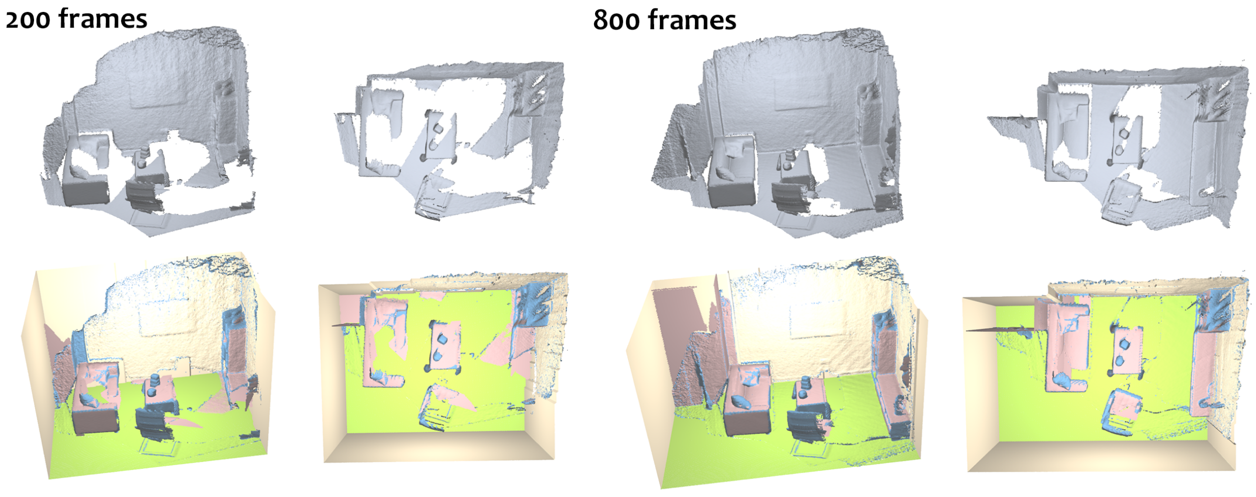


Figure 8.6: Progressive surface reconstruction and scene completion. After 200 frames large parts of the room are occluded, and filled in by the completion model. After 800 previously occluded areas are observed and replace the completion, and overall the structure of the room is more accurate.

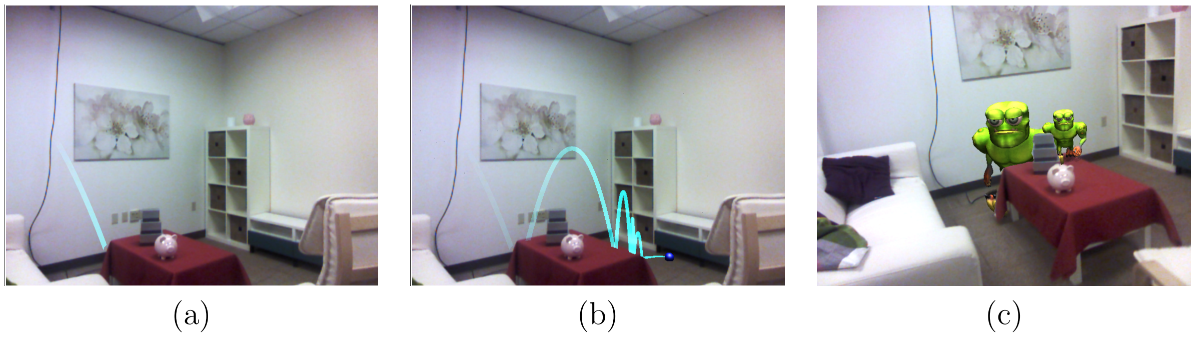


Figure 8.7: We created an augmented reality application to demonstrate the efficacy of our method. (a) a ball bouncing on an occluded floor falls through implausibly (b) a ball bouncing on the floor in a completed scene bounces realistically (c) Virtual objects may be placed on any surface, including occluded planes in the completed scene.

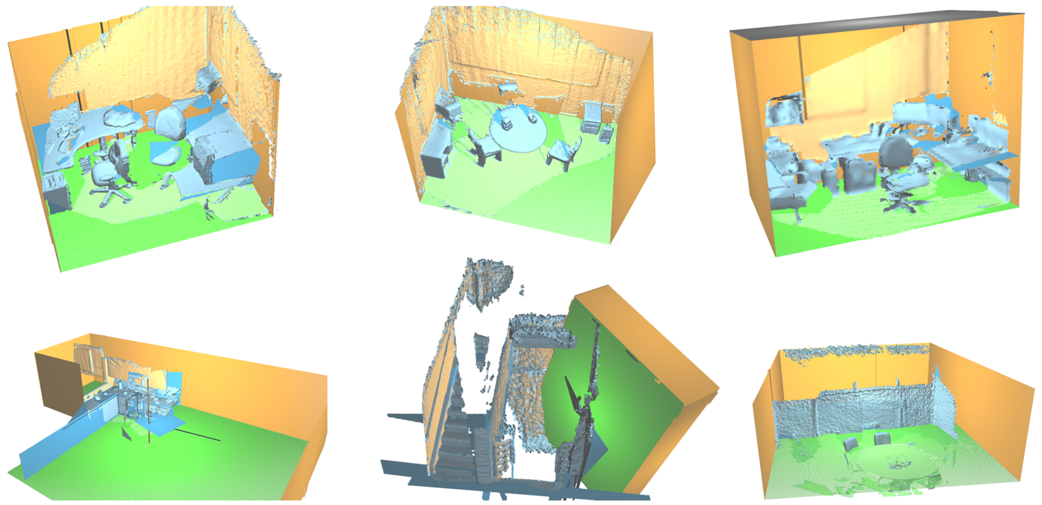


Figure 8.8: Example scenes: darker colors represent completed areas, hues signify plane classification. The top row contains mostly successful completions. The bottom row contains failure cases, on the left the room is huge due to many reflective surfaces, the middle image is of a staircase, vertical in nature, which leads to incorrect plane classifications. And on the right a high ceiling and windows cause misclassification of the floor plane.

Chapter 9

Conclusions

To promote and further research in RGBD scene understanding, we have introduced two new indoor RGBD datasets that have become the de facto standards for RGBD scene understanding tasks.

We have introduced and explored several algorithms for performing instance segmentation in RGBD scenes. Our greedy inference scheme is quick but makes mistakes from which it can never recover. Our instance-segmentation trees improve upon this by allowing the model to reason across different levels of segmentation trees but are still limited by the number of regions that can be explored. Finally, our exemplar-based instance segmentation model improves upon both of these models by reasoning at both the exemplar and pixel level.

Given an instance segmentation of a scene, various tasks can then be performed using the instance segmentation as input. We have introduced the task of physical support reasoning and demonstrated how to extend planar object instances in 3D space.

While this work has concentrated on the algorithms for instance segmentation, improvement may be sought via better features. Deep Learning models continue to provide better feature representations of regions which may lead to improved instance segmentation results. The use of 3D object models may lead to improved reasoning about which (super) pixels belong to which exemplars.

Furthermore, various higher-order potentials may serve to improve exemplar-based instance segmentations. For example, learned potentials over co-planar or similar regions may inhibit multiple exemplars that represent the same object instance from being chosen.

Appendix A

MPLP Message Passing

To optimize the dual objective of Equation 6.16, we use a block coordinate descent algorithm (MPLP) in which each potential function defines a block of unknowns which are all updated simultaneously [?].

During each iteration of MPLP, we update the values of the dual variables in blocks and then compute the value of the dual objective in order to determine whether or not the algorithm has converged. We present a series of algorithms that allow for efficient updates of these variables. Furthermore, since the dual objective (Equation 6.16) is the sum of a series of maximizations over separate variables, computing the dual can be broken up into a series of subproblems which we refer to as *maximizing subproblems*.

Common to both the variable updates and maximizing subproblems are searches over two parameterized subspaces of $\mathcal{L}$. Let $\mathcal{L}_1(m, c)$ be a subspace of $\mathcal{L}$ parameterized by exemplar m and class c such that every label in L maps to either an exemplar other than m or to exemplar m and class c. Formally, $\mathcal{L}_1(m, c) = \{l : (\mu(l) \neq m) \vee (\mu(l) = m \wedge \kappa(l) = c)\}$. Additionally, let $\mathcal{L}_2(m)$ be the subspace parameterized by exemplar m such that each every label in $\mathcal{L}_2(m)$ maps to an exemplar other than m. Formally, $\mathcal{L}_2(m) = \{l : (\mu(l) \neq m)$.

A.1 Maximizing Subproblems: Exemplar Potentials

The exemplar potentials' maximizing subproblem is:

$$\max_{E_m} \left[\theta_m^{\text{ex}}(E_m) + \lambda(E_m) + \lambda'(E_m) + \sum_{k=1}^{K} \lambda''_{km}(E_m)\right] \tag{A.1}$$

The maximization is trivially optimized by iterating over the possible values of E_m. In practice, we store the following variables in memory as in [**?**]:

$$\bar{\theta}_m^{\text{ex}} = \theta_m^{\text{ex}}(E_m) + \lambda(E_m) + \lambda'(E_m) + \sum_{k=1}^{K} \lambda''_{km}(E_m) \tag{A.2}$$

A.2 Maximizing Subproblems: Pixel Potentials

The pixel potentials maximizing subproblem is:

$$\max_{P_i} \left[\theta_i^{\text{pix}}(P_i) + \sum_{f:i\in f} \delta_{fi}(P_i) + \sum_{m=1}^{M} \delta'{}_{mi}(P_i)\right] \tag{A.3}$$

The maximization is trivially optimized by iterating over the possible values of P_i. In practice, we store the following variables in memory as in [**?**]:

$$\bar{\theta}_m^{\text{pix}} = \theta_i^{\text{pix}}(P_i) + \sum_{f:i\in f} \delta_{fi}(P_i) + \sum_{m=1}^{M} \delta'{}_{mi}(P_i) \tag{A.4}$$

A.3 Maximizing Subproblems: Tree Potentials

To maximize the tree potential (Equation 6.12), we formulate a novel labeling problem which can be efficiently and exactly optimized using max-sum belief propagation: Let each region in the tree be a node in a graph. Define edges between each parent and child region in the graph and let each region take one of three states: (0) which indicates that a region is not selected nor are its ancestors, (1) which indicates that an ancestor of the region is selected and (2) the region itself is selected. The unary potentials are defined as:

$$\phi^{\text{bp}}(x_m) = \begin{cases} -\infty, & \text{if isRoot(m)} \wedge x_m = 1 \\ 0, & \text{if } x_m \neq 2 \\ -\max_c \lambda(E_m = c), & \text{if } x_m = 2 \end{cases} \tag{A.5}$$

and the transition potentials are defined as:

		Parent Label		
		0	1	2
	0	0	$-\infty$	$-\infty$
Child	1	$-\infty$	0	0
Label	2	0	$-\infty$	$-\infty$

Table A.1: Transition Potential Values for our Belief-Propagation Based tree selection algorithm.

As indicated in Table A.1, if the child node's label is 0, indicating that neither the node nor its ancestors are selected, then its parent node's label must necessarily be set to 0 as well. If the child node's label is 1, indicating that an ancestor is selected, then the parent node's label must necessarily be set to 1 or 2. Finally, if the child node's label is 2, indicating that the child node is selected, then the parent node's label must necessarily be set to 0.

Note that this tree-cutting algorithm can also be used to cut the instance segmentation trees defined in Section 5.5.

A.4 Maximizing Subproblems: Pixel-Exemplar Agreement

In our original Lagrangian formulation (Equation 6.16), the pixel-exemplar agreement potentials are very expensive. The inclusion of duplicate variables P'_i for each of the M potentials introduces a very large number of messages, increasing both the overhead of computing and storing them. In particular, each potential θ_m^{agree} sends messages to every pixel in the scene and sends a different message for each one of the possible pixel labels.

However, since each pixel-exemplar agreement potential need only ensure agreement between the pixels and a single exemplar's label (E_m), we don't necessarily need a distinct message for every pixel label that is not consistent with that exemplar's label. Formally, we don't necessarily need a separate message ${\delta'}_{mi}(a)$ for each $\{a : \mu(a) \neq m\}$ which all correspond to values which can't possibly contradict the value of E_m. Consequently, we can constrain the Lagrangian such that ${\delta'}_{mi}(a) = {\delta'}_{mi}(b) \forall\ \{a, b : \mu(a) \neq m, \mu(b) \neq m\}$. In principle, this can only serve to raise the value of the Lagrangian and corresponds to creating a coarse partitioning of messages [**?**].

Using this partitioning, we can compute the messages for θ_m^{agree} using the following label space: Let Q^m be the set of labels $\{q_1, ...q_C, \beta\}$. Then $Q^m[l]$ is a mapping from $l \in \mathcal{L}$ to Q^m such that $Q^m[l] = q_c\ \forall\ l \in \{l : \mu(l) = m \wedge \kappa(l) = c\}$ and $Q^m[l] = \beta\ \forall\ l \in \{l : \mu(l) \neq m\}$.

Using such a partitioning of the variables we can re-express the pixel-exemplar agreement subproblem as:

$$\max_{E''_m, Q} \left[\theta_m^{\text{agree}}(E''_m, Q) - \lambda'_m(E''_m) - \sum_{i=1}^{N} \delta'_{mi}(Q)\right] \tag{A.6}$$

The pixel labeling subproblem can rewritten as:

$$\max_{P_i} \left[\phi_i^{\text{pix}}(P_i) + \sum_{f:i\in f} \delta_{fi}(P_i) + \sum_{m=1}^{M} \delta'_{mi}(Q[P_i]) \right] \tag{A.7}$$

While maximizing over the consistency potentials (Equation A.6) involves an exponential number of states, it can be optimized in $O(NMK + M)$ time.

Algorithm 2: Maximizing the Pixel-Exemplar Agreement Potential Subproblem

```
Data: Values $\delta'_{mi}(Q_i)\ \forall\ i \in N,\quad \lambda'_m(E''_m)$
Result: Maximum value $v*$ for $\max_{Q,E''_m} \left[\phi_m^{\text{agree}}(E''_m, Q) - \lambda'_m(E''_m) - \sum_{i=1}^{N} \delta'_{mi}(Q_i)\right]$
/* Compute the value of $E''_m = 0$                                              */
1 $v_0 = -\lambda'_m(0) + \sum_{i=1}^{N} \max_{l\in\mathcal{L}_2(m)} -\delta'_{mi}(l)$
2
/* Compute the values of $E''_m = c$                                              */
3 for $c = 1 \dots K$ do
      /* Find the pixel most likely to be assigned to $m$ if $E''_m = c$, where $\beta$
         indicates the assignment of pixel $i$ to an exemplar other than $m$.    */
4     $\rho_c = \arg\max_i \left[\delta'_{mi}(\beta) - \delta'_{mi}(\zeta(m,c))\right]$
5
      /* Next, compute the value of assigning $E''_m = c$                         */
6     $v_c = -\lambda'_m(c) - \delta'_{m\rho_c}(\zeta(m,c)) + \sum_{j\neq\rho_c} \max_{l\in\mathcal{L}_1(m,c)} -\delta'_{mj}(l)$
7 end
/* Finally, find the max over assignments to $E''_m$                              */
8 $v* = \max_{c\in\{0,1\dots K\}} v_c$
```

The label β represents the pixel label that a pixel i would be assigned to if it were assigned to an exemplar other than m. In other words for a pixel i, $\beta = \max_{l:\mu(l)\neq m} \delta'_{mi}(l)$. Step 1, 4, and 6 can all be done in $O(NMK)$ time and step 8 is $O(M)$.

Lemma 1. The maximum value in Equation A.6 is obtained via Algorithm 2.

Proof:

Condition 1: Let $\{A, E_m = c\}$ be a maximizing solution not obtained by the algorithm.

By definition $\mu(A_{\rho_c}) = m$ and $\kappa(A_{\rho_c}) = c$ since if it were otherwise, there would be a pixel η such that $\mu(A_\eta) = m$ and $\kappa(A_\eta) = c$ and we could swap the values of A_{ρ_c} and A_η and increase the value of the solution which contradicts our assumption that $\{A, E_m = c\}$ is the maximizing solution. Furthermore, since Algorithm 2 draws all other variables $A_i : i \neq \rho_c$ from $\mathcal{L}_1(m, c)$, then there must be a pixel $A_i = l$ where $l \notin \mathcal{L}_1(m, c)$ which would incur the cost of $-\infty$ (Equation A.6). In this case, we could switch the assigment of A_i to any other label in $\mathcal{L}_1(m, c)$ and raise the value of the solution which violates our assumption that $\{A, E_m = c\}$ is the maximizing solution.

Condition 2: Let $\{A, E_m = \emptyset\}$ be a maximizing solution not obtained by the algorithm. Because Algorithm 2 draws all values from $\mathcal{L}_2(m)$ when $E_m = \emptyset$, then there must be some pixel assignment $A_i = l$ where $l \notin \mathcal{L}_2(m)$ which would incur the cost of $-\infty$. If so, then we could reassign A_i to any label in $\mathcal{L}_2(m)$ and raise the value which contradicts the assumption that $\{A, E_m = \emptyset\}$ is a maximizing solution.

A.5 Maximizing Subproblems: Class-Cardinality Potentials

In our original formulation (Equation 6.16), each class specific cardinality potential maintains messages for each exemplar and semantic class. However, rather than maintain K messages for each, each potential need only maintain 2 messages for each exemplar indicating whether or not the exemplar takes the class of the potential or not. Formally, $A^k[E_m]$ is a mapping from $\{1 \ldots M\}$ to $\{0, 1\}$. Using such a partitioning of the variables, we can re-express the class-cardinality potential subproblem as:

$$\max_{A} \left[\theta_k^{\text{card}}(A) - \sum_{m=1}^{M} \lambda''_{km}(A_m)\right] \tag{A.8}$$

and the exemplar labeling subproblem can be rewritten as:

$$\max_{E_m} \left[\theta_m^{\text{ex}}(E_m) + \lambda(E_m) + \lambda'(E_m) + \sum_{k=1}^{K} \lambda''_{km}(A^k[E_m])\right) \tag{A.9}$$

The cardinality potential subproblems (Equation A.8) can be efficiently maximized using Algorithm 3. The algorithm is based on the efficient cardinality potential of Tarlow *et al.* [**?**]. However, rather than apply a single cardinality potential for binary segmentation, we apply a similar algorithm to multi-class segmentation:

Algorithm 3: Maximizing the Class Cardinality Subproblem.

```
Data: Semantic class $k$, Values $\lambda''_k \in \mathbb{R}^{M\times 1}$
Result: Maximum value $v^*$ for $\max_A \left[\phi_k^{\text{card}}(A) - \sum_{m=1}^{M} \lambda''_k(A_m)\right]$
/* Compute the energy differences between assigning each exemplar to the
   current class $k$ or not:                                              */
1  for $m = 1 \ldots M$ do
2      $\eta_m = \lambda''_{km}(0) - \lambda''_{km}(1)$
3  end
/* Sort energy differences in decreasing order:                           */
4  $\pi = \text{sort}(\eta)$
/* Compute cumulative sum of assigning exemplars to current class:        */
5  $\tau_0 = \sum_{m=1}^{M} \lambda''_{km}(0)$
6  for $i = 1 \ldots M$ do
7      $\tau_i = \tau_{i-1} - \eta_{\pi[i]}$
8  end
/* Accrue class-cardinality potential values (Equation 6.14)              */
9  for $i = 0 \ldots M$ do
10     $\tau_i = \tau_i + w_{k,i}$
11 end
12 $v* = \max_i \tau_i$
```

A.6 Maximizing Subproblems: Pairwise Potentials

The pairwise potential subproblem can be efficiently optimized with three linear passes by splitting the problem into two parts: maximizing over the labels where P_i and P_j are equal and maximizing over the labels where they are different. Algorithm 4 contains the details of the maximization strategy.

Algorithm 4: Maximizing the Pairwise Potentials Subproblem.

```
Data: Values $\delta_{fi}, \delta_{fj}$
Result: Maximum value $v = \max_{P_i,P_j} \left[\phi_{ij}^{\text{pair}}(P_i, P_j) - \delta_{fi}(P_i) - \delta_{fj}(P_j)\right]$
   /* Compute the maximum value where $P_i = P_j$:                        */
 1 $l_{\text{eq}} = \arg\max_l [-\delta_{fi}(l) - \delta_{fj}(l)]$
   /* Compute the top two maximizing labels for pixel $i$                 */
 2 $l_{i1} = \arg\max_l -\delta_{fi}(l)$
 3 $l_{i2} = \arg\max_{l \neq l_{i1}} -\delta_{fi}(l)$
   /* Compute the top two maximizing labels for pixel $j$                 */
 4 $l_{j1} = \arg\max_l -\delta_{fj}(l)$
 5 $l_{j2} = \arg\max_{l \neq l_{j1}} -\delta_{fj}(l)$
 6 if $l_{i1} = l_{j1}$ then
 7     $v_{ne} = \max(-\delta_{fi}(l_{i1}) - \delta_{fj}(l_{j2}), -\delta_{fi}(l_{i2}) - \delta_{fj}(l_{j1}))$
 8     $v = \max([-\delta_{fi}(l_{\text{eq}}) - \delta_{fj}(l_{\text{eq}})], [\phi_{ij}^{\text{pair}}(l_{i1}, l_{j1}) + v_{ne}])$
 9 else
10     $v = \max([-\delta_{fi}(l_{\text{eq}}) - \delta_{fj}(l_{\text{eq}})], [\phi_{ij}^{\text{pair}}(l_{i1}, l_{j1}) - \delta_{fi}(l_{i1}) - \delta_{fj}(l_{j1})])$
11 end
```

A.7 Block Coordinate Update for Tree Potential's λ Variables

For the purposes of tree cutting, we need only maintain messages about whether a region is selected or not, rather than maintain messages about each selected region's semantic class. Consequently, rather than have the tree potential maintain a separate variable for each possible semantic class $E' = \{E'_m : m = 1 \ldots M, E'_m \in \{0 \ldots K\}\}$, we instead only maintain binary variables $F = \{F_m : m = 1 \dots M, F_m \in \{0, 1\}\}$. Consequently, variable $F_m(0)$ represents the value of exemplar m not being selected in the cut and $F_m(1)$ represents the value of exemplar m being selected in the cut.

The block coordinate update for the tree potential variable $\lambda(F_m)$ is:

$$\lambda_m(F_m) = -\lambda_m^{-m}(F_m) + \frac{1}{M} \max_{F \setminus m} \left[\phi^{\text{tree}}(F) + \sum_{\hat{m}} \lambda_{\hat{m}}^{-m}(F_{\hat{m}}) \right] \tag{A.10}$$

where $\lambda_m^{-m}(F_m)$ is defined as:

$$\lambda_m^{-m}(0) = \bar{\theta}_m^{\text{ex}}((0)) - \lambda_m(0) \tag{A.11}$$

$$\lambda_m^{-m}(1) = \max_{E_m \neq 0} \bar{\theta}_m^{\text{ex}}(E_m) - \lambda_m(E_m) \tag{A.12}$$

Efficiently computing the Updates

We can avoid computing $\max_{F \setminus m}$ separately for each $\lambda_m(F_m)$ via the following algorithm. Let $G = \{G_m : m = 1 \dots M, G_m \in \{0, 1, 2\}\}$ be a tree structured graph of variables representing the segmentation tree such that each variable assignment G_m indicates whether exemplar m is selected ($G_m = 2$), has an ancestor that is selected ($G_m = 1$) or is neither selected nor has an ancestor that is selected ($G_m = 0$). Furthermore, let the each variable take on the following costs:

$$G_m(0) = \bar{\theta}_m^{\text{ex}}((0)) - \lambda_m(0) \tag{A.13}$$

$$G_m(1) = \bar{\theta}_m^{\text{ex}}((0)) - \lambda_m(0) \tag{A.14}$$

$$G_m(2) = \max_{k=1 \dots K} \bar{\theta}_m^{\text{ex}}((k)) - \lambda_m(k) \tag{A.15}$$

and let the edges between nodes of G be assigned costs from Table A.1. We can infer the states with the highest cost by running Max-Sum Belief Propagation. Because the graph contains no loops, the beliefs of each node in the graph are the exact marginal probabilities of each variable. Consequently, by solving for the values of G with the highest cost once we can compute the values of message updates from Equation A.10 in one pass. Let $H \in \mathbb{R}^{M\times 3}$ represent the max marginals (the beliefs of each variable in G produced by the message passing of Max-Sum Belief Propagation). We can compute the updates to $\lambda_m(F_m)$ by running a single pass of BP and using the following equations:

$$\lambda_m(0) = -\lambda_m^{-m}(0) + \frac{1}{M}\max[H_m(0), H_m(1)] \tag{A.16}$$

$$\lambda_m(1) = -\lambda_m^{-m}(1) + \frac{1}{M}H_m(2) \tag{A.17}$$

A.8 Block Coordinate Update for Agreement Potentials λ' Variables

The block coordinate update step for each of the λ' variables is computed as follows:

$$\lambda'_m(E''_m) = -\lambda'^{-m}_m(E''_m) + \frac{1}{N+1}\max_Q\left[\phi_m^{\text{agree}}(Q, E''_m) + \sum_{\hat{i}=1}^{N}\delta'^{-m}_{\hat{i}}(Q_{\hat{i}}) + \lambda'^{-m}_m(E''_m)\right] \tag{A.18}$$

The algorithm used is the same as in Algorithm 2 except that we enforece the assignment of E''_m.

A.9 Block Coordinate Update for Agreement Potentials δ' Variables

The block coordinate update step for each of the $\delta'_{mi}(Q_i)$ variables is computed as follows:

$$\delta'_{mi}(Q_i) = -\delta_i'^{-m}(Q_i) + \frac{1}{N+1} \max_{Q\backslash i, E_m} \left[\phi_m^{\text{agree}}(E_m, Q) + \sum_{\hat{i}=1}^{N} \delta_{\hat{i}}'^{-m}(Q_{\hat{i}}) + \lambda_m^{-m}(E_m) \right] \quad \text{(A.19)}$$

where:

$$\delta_i^{-m}(Q_i) = \max_{P_i \in Q_i} \left[\phi_i^{\text{pix}}(P_i) + \sum_{f:ij\in\mathcal{E}} \delta_{fi}(P_i) + \sum_{\hat{m}\neq m} \delta'_{\hat{m}i}(P_i) \right] \quad \text{(A.20)}$$

The algorithm used is the same as in Algorithm 2 except that we enforce the assignment of Q_i.

A.10 Block Coordinate Update for Cardinality Potentials λ'' Variables

As previously explained (Section A.5), rather than have each cardinality potential maintain a message for each possible value of E_m for each exemplar, we maintain only two messages for each exemplar indicating whether or not the exemplar takes the class of the potential or not using the variable $A = \{A_m : m = 1 \dots M, A_m \in \{0, 1\}\}$. The block coordinate update step for the class cardinality potentials is:

$$\lambda''_{km}(A_i) = -\lambda_m''^{-k}(A_i) + \frac{1}{M} \max_{A\backslash m} \left[\phi_k^{\text{card}}(A) + \sum_{\hat{m}=1}^{M} \lambda_{\hat{m}}''^{-k}(A_{\hat{m}}) \right] \quad \text{(A.21)}$$

The algorithm for computing the max is identical that of Algorithm 3 except that in the cases when $A_m = 0$, we enforce that the exemplar cannot be selected and in the cases when $A_m = 1$, we force that the exemplar is selected.

A.11 Block Coordinate Update for Pairwise Potentials δ Variables

The block coordinate update for pairwise variables is computed as:

$$\delta_{fi}(P_i) = -\delta_i^{-f}(P_i) + \frac{1}{2}\max_{P_j}\left[\phi_{ij}^{\text{pair}}(P_i, P_j) + \delta_i^{-f}(P_i) + \delta_j^{-f}(P_j)\right] \tag{A.22}$$

By keeping $\bar{\theta}_m^{\text{pix}}$ in memory (Section A.2) each δ_i^{-f} can be computed as:

$$\delta_i^{-f}(P_i) = \bar{\theta}_m^{\text{pix}}((P_i)) - \delta_{fi}((P_i)) \tag{A.23}$$

and the update can be computed in linear time $O(\mathcal{L})$.

Bibliography

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels. *École Polytechnique Fédéral de Lausssanne (EPFL), Tech. Rep*, 149300, 2010.

[2] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *TPAMI*, 2012.

[3] P. Arbelaez. Boundary extraction in natural images using ultrametric contour maps. In *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on*, pages 182–182. IEEE, 2006.

[4] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. From contours to regions: An empirical evaluation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2294–2301. IEEE, 2009.

[5] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(5):898–916, 2011.

[6] W. Brendel and S. Todorovic. Segmentation as maximumweight independent set. In *Neural Information Processing Systems*, volume 4, 2010.

[7] J. Carreira and C. Sminchisescu. Constrained parametric min-cuts for automatic object segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE*

Conference on, pages 3241–3248. IEEE, 2010.

[8] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619, 2002.

[9] A. E. Derek Hoiem and M. Hebert. Geometric context from a single image. In *International Conference on Computer Vision*, 2005.

[10] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[11] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Scene parsing with multiscale feature learning, purity trees, and optimal covers. *arXiv preprint arXiv:1202.2160*, 2012.

[12] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2), 2004.

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv preprint arXiv:1311.2524*, 2013.

[14] R. Guo and D. Hoiem. Beyond the line of sight: Labeling the underlying surfaces. In *ECCV*, 2012.

[15] S. Gupta, P. Arbelaez, and J. Malik. Perceptual organization and recognition of indoor scenes from rgb-d images. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 564–571. IEEE, 2013.

[16] D. Hoiem, A. A. Efros, and M. Hebert. Recovering occlusion boundaries from an image. *International Journal of Computer Vision*, 91(3):328–346, 2011.

[17] D. Hoiem, A. A. Efros, and M. Hebert. Recovering occlusion boundaries from an image. *Int. J. Comput. Vision*, 91:328–346, February 2011.

[18] A. Ion, J. Carreira, and C. Sminchisescu. Image segmentation by figure-ground composition into maximal cliques. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2110–2117. IEEE, 2011.

[19] Z. Jia, A. Gallagher, A. Saxena, and T. Chen. 3d-based reasoning with blocks, support, and stability. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[20] T. Joachims, T. Finley, and C.-N. J. Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

[21] S. Kim, S. Nowozin, P. Kohli, and C. D. Yoo. Higher-order correlation clustering for image segmentation. In *NIPS*, pages 1530–1538, 2011.

[22] P. Kohli, P. H. Torr, et al. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision*, 82(3):302–324, 2009.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4, 2012.

[24] M. P. Kumar and D. Koller. Efficiently selecting regions for scene understanding. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3217–3224. IEEE, 2010.

[25] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate frank-wolfe optimization for structural svms. *arXiv preprint arXiv:1207.4747*, 2012.

[26] L. Ladicky, C. Russell, P. Kohli, and P. H. Torr. Associative hierarchical crfs for object class image segmentation. In *Computer Vision, 2009 IEEE 12th International Confer-*

ence on, pages 739–746. IEEE, 2009.

[27] V. Lempitsky, A. Vedaldi, and A. Zisserman. A pylon model for semantic segmentation. *NIPS*, 2011.

[28] A. Levin, D. Lischinski, and Y. Weiss. Colorization using optimization. In *SIGGRAPH*, 2004.

[29] M. Maire, P. Arbeláez, C. Fowlkes, and J. Malik. Using contours to detect and localize junctions in natural images. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[30] T. Malisiewicz and A. Efros. Improving spatial support for objects via multiple segmentations. In *BVMC*, 2007.

[31] D. Munoz, J. Bagnell, and M. Hebert. Stacked hierarchical labeling. *Computer Vision–ECCV 2010*, pages 57–70, 2010.

[32] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[33] C. Pantofaru, C. Schmid, and M. Hebert. Object recognition by integrating multiple image segmentations. *Computer Vision–ECCV 2008*, pages 481–494, 2008.

[34] Z. Ren and G. Shakhnarovich. Image segmentation by cascaded region agglomeration. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2011–2018. IEEE, 2013.

[35] B. C. Russell, W. T. Freeman, A. A. Efros, J. Sivic, and A. Zisserman. Using Multiple Segmentations to Discover Objects and their Extent in Image Collections. In *Computer Vision and Pattern Recognition*, 2006.

[36] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[37] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Computer Vision–ECCV 2006*, pages 1–15. Springer, 2006.

[38] N. Silberman and R. Fergus. Indoor scene segmentation using a structured light sensor. In *Proceedings of the International Conference on Computer Vision - Workshop on 3D Representation and Recognition*, 2011.

[39] D. Sontag, A. Globerson, and T. Jaakkola. Introduction to dual decomposition for inference. In S. Sra, S. Nowozin, and S. J. Wright, editors, *Optimization for Machine Learning*. MIT Press, 2011.

[40] D. Tarlow and R. S. Zemel. Structured output learning with high order loss functions. In *International Conference on Artificial Intelligence and Statistics*, pages 1212–1220, 2012.

[41] J. Tighe and S. Lazebnik. Superparsing: scalable nonparametric image parsing with superpixels. In *ECCV*, pages 352–365, Berlin, Heidelberg, 2010. Springer-Verlag.

[42] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large Margin Methods for Structured and Interdependent Output Variables. *J. Mach. Learn. Res.*, 6:1453–1484, Dec. 2005.

[43] J. Yao, S. Fidler, and R. Urtasun. Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 702–709. IEEE, 2012.