Object Recognition and Semantic Scene Labeling for RGB-D Data

Kevin Kar Wai Lai

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2013

Reading Committee:

Dieter Fox, Chair

Xiaofeng Ren

Steven M. Seitz

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Object Recognition and Semantic Scene Labeling for RGB-D Data

Kevin Kar Wai Lai

Chair of the Supervisory Committee:
Professor Dieter Fox
Computer Science and Engineering

The availability of RGB-D (Kinect-like) cameras has led to an explosive growth of research on robot perception. RGB-D cameras provide high resolution (640×480) synchronized videos of both color (RGB) and depth (D) at 30 frames per second. This dissertation demonstrates the thesis that combining of RGB and depth at high frame rates is helpful for various recognition tasks including object recognition, object detection, and semantic scene labeling. We present the RGB-D Object Dataset, a large dataset of 250,000 RGB-D images of 300 objects in 51 categories, and 22 RGB-D videos of objects in indoor home and office environments. We introduce algorithms for object recognition in RGB-D images that perform category, instance, and pose recognition in a scalable manner. We also present HMP3D, an unsupervised feature learning approach for 3D point cloud data, and demonstrate that HMP3D can be used to learn hierarchies of features from different attributes including color, gradient, shape, and surface normal orientation. Finally, we present a scene labeling approach for scenes constructed from RGB-D videos. The approach uses features learned from both individual RGB-D images and 3D point clouds constructed from entire video sequences. Through these applications, this thesis demonstrates the importance of designing new features and algorithms that specifically utilize the advantages of RGB-D cameras over traditional cameras and range sensors.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Dieter Fox for his guidance. Dieter taught me how to do high quality research and I would not have completed this dissertation without his support. I also thank other members of my supervisory committee including James J. Little, Xiaofeng Ren, and Steven M. Seitz, and graduate school representative Ming-Ting Sun. I would also like to thank Liefeng Bo, collaborator and mentor who has provided invaluable advice and support. Finally, I would like to thank fellow students, staff, and faculty of the Department of Computer Science and Engineering for creating a supportive and productive environment.

DEDICATION

To my family and friends who have given me unconditional love and support.

Chapter 1

INTRODUCTION

Recognizing objects in natural scenes is a crucial capability for an autonomous robot that can understand and interact with the world and be of use in everyday scenarios. In order to operate safely and efficiently in populated urban and indoor environments, autonomous robots must be able to distinguish between objects such as cars, people, buildings, trees, kitchenware, chairs, and furniture. Applications include autonomous cars driving on urban streets as well as robots navigating through pedestrian areas or operating in indoor environments. Domestic housekeeping and elderly care robots will also need the ability to detect, classify and locate common objects found in indoor environments if they are to perform useful tasks for people.

Considerable progress has been made towards object recognition in robotics applications. The DARPA Urban Challenge encouraged advancements in the perception mechanisms of autonomous vehicles. Papers including [132] and [85] have investigated detecting and tracking cars, pedestrians, and street signs using the Velodyne rotating laser array as the sensor. Others have looked at object recognition on mobile robots navigating in urban environments [130, 143]. Aside from robots, more and more interactive systems require object recognition capabilities to interact with users in an intelligent way [72]. Object recognition has been a central and heavily studied research topic in computer vision, with a lot of progress made in recent years. Modern object recognition systems can distinguish between hundreds of categories and detect a variety of objects in complex real-world images on the Web [42].

The Microsoft Kinect [108, 75], released in 2011, has precipitated an explosive growth of research on robot perception. RGB-D cameras like the Kinect are capable

of providing high resolution (640×480) synchronized videos of both color (RGB) and depth (D) at 30 frames per second. Microsoft's use of the Kinect for the burgeoning gaming market has created economies of scale never before seen for a 3D sensor. At 100 United States Dollars as of 2013, the Microsoft Kinect is much more affordable than laser rangefinders of comparable resolution, such as the SwissRanger SR4000, which in the same year costs 4000 USD. This depth sensing technology represents an opportunity to dramatically increase the object recognition, manipulation, navigation, and interaction capabilities in robots.

Since the release of the Kinect, there has been a number of works on shape and appearance based object matching [64, 113]. A limitation of these approaches is that they are designed for recognizing particular objects that the system has seen. Neither demonstrated the performance of their features in recognizing objects that are similar, but not identical, to objects that the system has already seen and placed into its database (e.g. recognizing an object as being a coffee mug even though it has only seen examples of other coffee mugs, but not this particular one).

1.1 RGB-D Cameras

An RGB-D camera captures images where each pixel is a four-dimensional vector $[r, g, b, d]$, where r, g, b is the red, green, blue color values, and d is the depth, i.e. the perpendicular distance to that pixel from the image plane. RGB-D cameras widely available as of 2013 include the first and second generation Microsoft Kinect, ASUS Xtion PRO LIVE, the PrimeSense Carmine, and the Creative Interactive Gesture Camera (see Fig. 1.1). There are two dominant depth sensing technologies in use in these consumer depth cameras. The second generation Kinect and the Creative camera use a time-of-flight camera. Time-of-flight cameras operate by emitting pulses of light and measuring the time it takes for them to reflect back to estimate the depth [45].

The first generation Kinect, PrimeSense Carmine 1.08 and the ASUS Xtion PRO

Figure 1.1: Five RGB-D cameras widely available on the market as of 2013. Clockwise from top left, they are: first generation Microsoft Kinect, second generation Microsoft Kinect, Creative Interactive Gesture Camera, PrimeSense Carmine 1.08, and ASUS Xtion PRO LIVE.

LIVE share similar hardware with each other and operate on the principle of stereo triangulation. Each camera has a standard CMOS sensor for capturing RGB images, an infrared CMOS sensor, and an infrared emitter. The infrared (IR) emitter emits a fixed, known IR pattern which the IR sensor sees. The IR emitter and sensor form a stereo pair and the depth of each pixel is triangulated using a stereo matching algorithm [120]. Stereo matching requires solving the correspondence problem: the depth of a pixel in one sensor image can only be computed if the corresponding pixel in the other sensor image originating from the same physical location is found. It is only possible to find this correspondence when there are distinctive and stable features that reliably appear in both sensors. Passive stereo cameras rely on the environment itself to provide these distinctive features, and hence fail when looking at featureless environments like a blank wall. On the other hand, RGB-D cameras replace one of the passive sensors with an active IR emitter. The emitter always emits a unique and distinctive pattern, so that even a blank wall appears to have distinctive features in

Figure 1.2: An example RGB (left) and depth (right) image pair captured with an RGB-D camera. The RGB image is in true color, while the depth image is in greyscale, where lighter is farther away.

the IR spectrum. This significantly improves the quality of depth data obtained from RGB-D cameras over passive stereo pairs.

Fig. 1.2 shows an example RGB and depth image pair captured with an RGB-D camera. The RGB-D camera captures RGB and depth images at 640×480 and 30 frames per second. One can see from this frame that RGB-D cameras are able to estimate depth even on featureless surfaces like the kitchen drawers and the walls. The depth estimation fails for reflective surfaces like the drawer handles, parts of the kitchen sink, and the flashlight's reflector. There are missing depth values at the edges of the image because this depth image has been aligned to the RGB image, and the two sensors have an offset and hence slightly different fields of view.

1.2 Thesis Outline

This dissertation demonstrates the following thesis:

> *The combination of RGB and depth at high frame rates made possible by RGB-D cameras significantly improves performance on various recognition tasks including object recognition, object detection, and semantic scene la-*

beling.

We study the problem of object recognition, object detection, and semantic scene labeling using RGB-D data. Here we define these three related but different recognition tasks.

In common usage, *object recognition* refers to the identification of objects in a scene. In the shape retrieval literature, object recognition refers to grouping objects into different classes, and then when presented with a new object, the system must determine its class [8]. This is similar to the task that is called object classification or image classification in the computer vision community: the system must output the identity of an object that predominantly occupies an image. Since we work with both images and 3D point clouds in the case of RGB-D data, this dissertation defines the task of *object recognition* as follows: the system is presented with a scene (e.g. an image or a 3D point cloud) that is predominantly occupied by one object, and the system's task is to determine the identity of that dominant object.

In both object detection and scene labeling, the system must determine the identity of multiple objects that are present in the scene (also an image or a 3D point cloud). In *object detection*, the system's task is to detect and localize all objects of interest with enclosing bounding boxes. The system's task in *scene labeling* is to assign a label to every pixel or 3D point. Hence, scene labeling requires more precise segmentation and labeling of the scene than the other two tasks. Object recognition, as defined in this dissertation, only involves determining the identity of one object, while scene labeling and object detection both require determining the identity and location of multiple objects. Hence, object recognition as defined in this dissertation can conceptually be seen as a subroutine of scene labeling and object detection.

Each of the three recognition tasks can be performed at different levels of granularity: 1) *category recognition* is determining the category of an object even though the system has not seen it before (e.g. a soda can), 2) *instance recognition* is determining the identity of a specific object with a specific appearance that the system

Figure 1.3: Examples of objects from the RGB-D Object Dataset.

has seen before (e.g. a pepsi can), and 3) *pose recognition* is determining the object's six-dimensional pose (translation and rotation) relative to the camera.

The combination of color images and depth images produced by RGB-D cameras presents an opportunity to drastically improve object recognition and scene labeling. However, it also presents some interesting challenges: What feature representation should we use for RGB-D data? How can we efficiently distinguish between a large number of objects? How can we leverage the high frame rate capabilities of the sensor? This thesis investigates all of the above questions. The main contributions of this thesis are:

- We collect the RGB-D Object Dataset, a large dataset of 250,000 RGB-D images of 300 objects in 51 categories, and 22 RGB-D videos of a subset of these objects in indoor home and office environments (Chapter 2). The dataset is annotated with the ground truth labels of objects and furniture pieces. Example images of 45 objects is shown in Fig. 1.3. The dataset is publicly available at http://www.cs.washington.edu/rgbd-dataset

Figure 1.4: Example semantic scene labeling from the RGB-D Scenes Dataset output by the scene labeling MRF described in Chapter 8. (Left) The 3D scene point cloud in true color.(Right) The 3D scene point cloud colored according to semantic label: bowl (red), cap (green), coffee mug (yellow), soda can (cyan), coffee table (purple), sofa (brown), background (grey).

- We present techniques for RGB-D object recognition based on existing state-of-the-art features and techniques that were designed for RGB images or 3D point cloud data (Chapter 3). The performance of these techniques on the RGB-D Object Dataset serve as a baseline for comparison.

- We present an exemplar-based distance learning approach for object recognition in 3D point cloud data (Chapter 4). This method combines multiple features and weighs their influence appropriately so as to maximize discrimination between object classes. It also uses domain adaptation to combine training data from a web repository of 3D CAD models and point clouds collected in the real world.

- We then introduce a distance learning approach for RGB-D object recognition, which provides a method for combining features extracted from the RGB image and features extracted from 3D point cloud data (Chapter 5). The approach is able to discard uninformative exemplars from the training data in order to speed up distance function evaluation at test-time.

- We present a novel tree-based object recognition and pose estimation technique that simultaneously addresses object recognition at all three levels of granularity and scales logarithmically with the number of objects that need to be distinguished (Chapter 6).

- We present an unsupervised feature learning approach for 3D point cloud data that learns a hierarchy of features from different attributes including color, gradient, shape, and surface normal orientation (Chapter 7).

- We present a scene labeling approach for scenes constructed from RGB-D videos. The approach leverages both image-based object detection techniques that run on a single RGB-D image and 3D-based object recognition techniques that run on the scene constructed from the entire video sequence (Chapter 8). Fig. 1.4 shows an example result from our technique.

- Finally, we conclude this thesis with our insights and directions for future work in Chapter 9.

Chapter 2

THE RGB-D OBJECT DATASET

2.1 Background

The availability of public image repositories on the Web, such as Google Images and Flickr, as well as visual recognition benchmarks like Caltech 101 [40], LabelMe [112] and ImageNet [34] has enabled rapid progress in visual object category and instance detection in the past decade. Similarly, the robotics dataset repository RADISH [65] has greatly increased the ability of robotics researchers to develop and compare their SLAM techniques. In this chapter, we present a large-scale, hierarchical multi-view object data set collected using an RGB-D camera, which we first introduced in [80]. The dataset and an associated set of software tools have been made publicly available to the research community to enable rapid progress based on this promising technology. The dataset is available at http://www.cs.washington.edu/rgbd-dataset.

Many existing recognition benchmarks are annotated only with category-level labels of the object [126, 68]. In these datasets, it is impossible to keep track of whether objects in different images are physically the same object, our dataset consists of multiple views of a known set of objects. We refer to each distinct object as an instance. This is similar to the 3D Object Category Dataset presented by Savarese et al. [117], which contains 8 object categories, 10 objects in each category, and 24 distinct views of each object. Since the release of the RGB-D Object Dataset, there have also been other datasets that are collected using RGB-D cameras. For example, 3DNet [142] and the Willow Garage Solutions in Perception Challenge dataset [51] both contain scenes containing objects recorded using RGB-D cameras. However, the RGB-D Object Dataset presented here is, as of publication of this dissertation (2013), still the

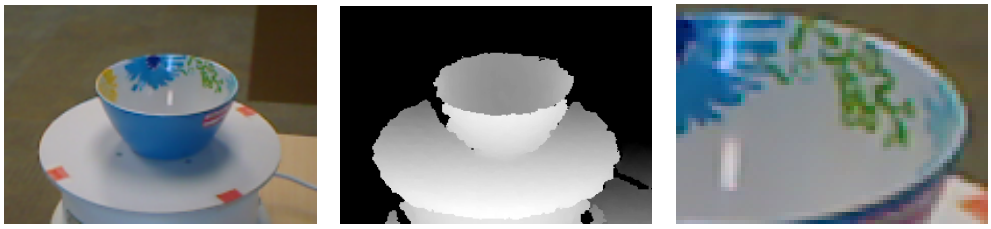

Figure 2.1: (Left) Each RGB-D frame consists of an RGB image and (middle) the corresponding depth image. (Right) A zoomed-in portion of the bowl showing details captured by the RGB-D camera.

largest published object dataset of its kind. The dataset contains RGB and depth video sequences of 300 common everyday objects from multiple view angles totaling 250,000 RGB-D images. The objects are organized into a hierarchical category structure using WordNet hyponym/hypernym relations.

2.2 Data Collection

The RGB-D Object Dataset contains visual and depth images of 300 physically distinct objects taken from multiple views. The chosen objects are commonly found in home and office environments, where personal robots are expected to operate. Objects are organized into a hierarchy taken from WordNet hypernym/hyponym relations and is a subset of the categories in ImageNet [34]. Fig. 2.2 shows several subtrees in the object category hierarchy. *Fruit* and *Vegetable* are both top-level subtrees in the hierarchy. *Device* and *Container* are both subtrees under the *Instrumentation* category that covers a very broad range of man-made objects. Each of the 300 objects in the dataset belong to one of the 51 leaf nodes in this hierarchy, with between three to fourteen instances in each category. The leaf nodes are shaded blue in Fig. 2.2 and the number of object instances in each category is given in parentheses. Fig. 2.3 shows some example objects from the dataset. Each shown object comes from one of the

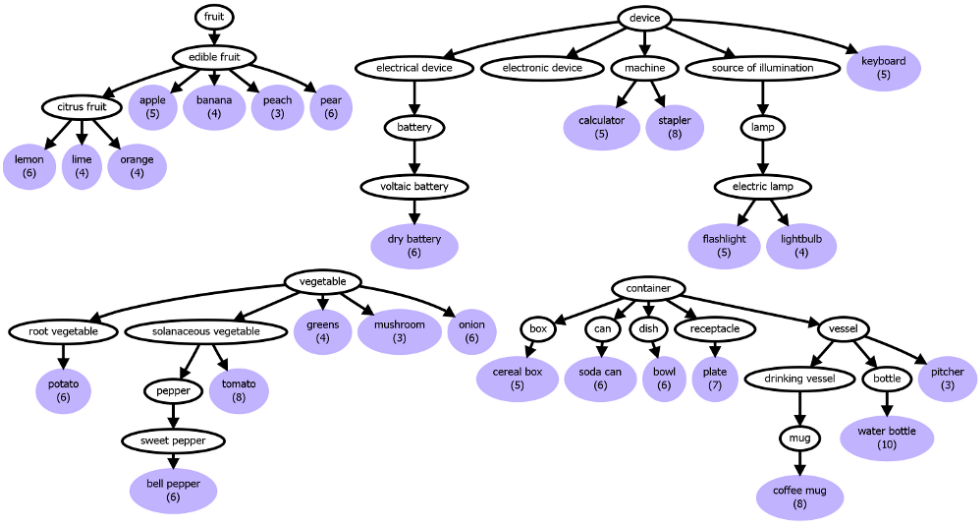

Figure 2.2: The fruit, device, vegetable, and container subtrees of the RGB-D Object Dataset object hierarchy. The number of instances in each leaf category (shaded in blue) is given in parentheses.

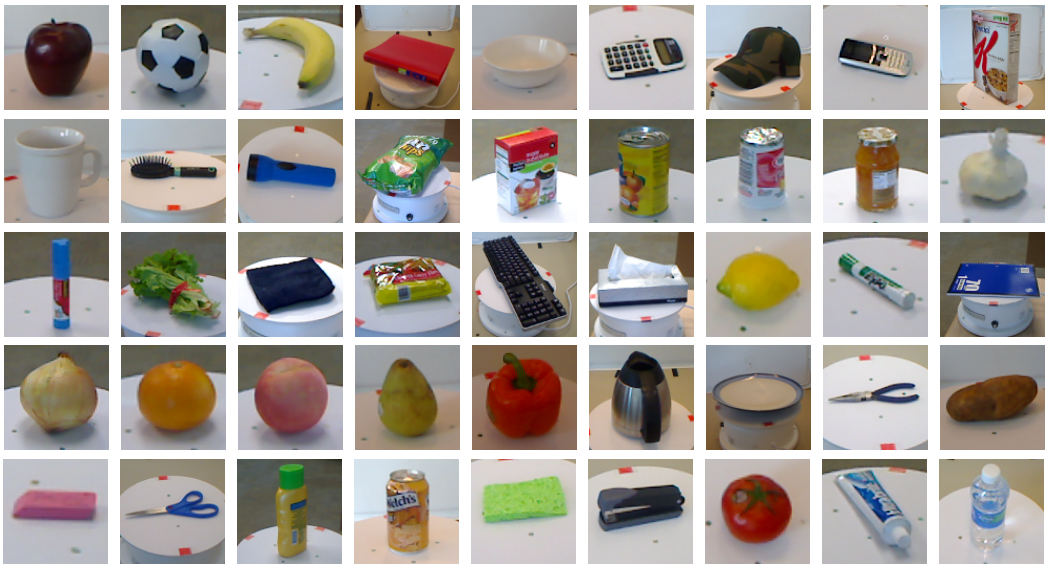

Figure 2.3: Objects from the RGB-D Object Dataset. Each object shown here belongs to a different category.

51 object categories. Although the background is visible in these images, the dataset also provides segmentation masks (see Fig. 2.4). The segmentation procedure using combined visual and depth cues is described in Section 2.3.

The dataset is collected using a sensing apparatus consisting of a prototype RGB-D camera manufactured by PrimeSense [108]. The RGB-D camera simultaneously records both color and depth images at 640×480 resolution. In other words, each 'pixel' in an RGB-D frame contains four channels: red, green, blue and depth. The 3D location of each pixel in physical space can be computed using known sensor parameters. The RGB-D camera creates depth images by continuously projecting an invisible infrared structured light pattern and performing stereo triangulation. Compared to passive multi-camera stereo technology, this active projection approach results in much more reliable depth readings, particularly in textureless regions. Fig. 2.1 (top) shows a single RGB-D frame which consists of both an RGB image and a depth image. Driver software provided with the RGB-D camera ensures that the RGB and depth images are aligned and time-synchronous. The RGB-D camera collects data at around 20 Hz.

Using this camera setup, we record video sequences of each object as it is spun around on a turntable at constant speed. The cameras are placed about one meter from the turntable. This is the minimum distance required for the RGB-D camera to return reliable depth readings. Data was recorded with the cameras mounted at three different heights relative to the turntable, at approximately 30°, 45° and 60° above the horizon. One revolution of each object was recorded at each height. Each video sequence is recorded at 20 Hz and contains around 250 frames, giving a total of 250,000 RGB + Depth frames in the RGB-D Object Dataset. The video sequences are all annotated with ground truth object pose angles between $[0, 2\pi]$ by tracking the red markers on the turntable. A reference pose is chosen for each category so that pose angles are consistent across video sequences of objects in a category. For example, all videos of coffee mugs are labeled such that the image where the handle

is on the right is 0°.

2.3 Segmentation

Without any post-processing, a substantial portion of the RGB-D video frames is occupied by the background. We use visual cues, depth cues, and knowledge of the configuration of the turntable and camera to produce fully segmented objects from the video sequences.

The first step in segmentation is to remove most of the background by taking only the points within a 3D bounding box where we expect to find the turntable and object, based on the known distance between the turntable and the camera. This prunes most pixels that are far in the background, leaving only the turntable and the object. Using the fact that the object lies above the turntable surface, we can perform RANSAC plane fitting [44] to find the table plane and take points that lie above it to be the object. This procedure gives very good segmentation for many objects in the dataset, but is still problematic for small, dark, transparent, and reflective objects. Due to noise in the depth image, parts of small and thin objects like rubber erasers and markers may get merged into the table during RANSAC plane fitting. Dark, transparent, and reflective objects cause the depth estimation to fail, resulting in pixels that contain only RGB but no depth data. These pixels would be left out of the segmentation if we only used depth cues. Thus, we also apply vision-based background subtraction to generate another segmentation. The top row of Fig. 2.4 shows several examples of segmentation based on depth. Several objects are correctly segmented, but missing depth readings cause substantial portions of the water bottle, jar and the marker cap to be excluded.

To perform vision-based background subtraction, we applied the adaptive gaussian mixture model of KaewTraKulPong et al. [71] and used the implementation in the OpenCV library. Each pixel in the scene is modeled with a mixture of K gaussians that is updated as the video sequence is played frame-by-frame. The model is adaptive

and only depends on a window W of the most recent frames. A pixel in the current frame is classified as foreground if its value is beyond σ standard deviations from all gaussians in the mixture. For our object segmentation we used $K = 2$, $W = 200$, and $\sigma = 2.5$. The middle row of Fig. 2.4 shows several examples of visual background subtraction. The method is very good at segmenting out the edges of objects and can segment out parts of objects where depth failed to do so. However, it tends to miss the centers of objects that are uniform in color, such as the peach in Fig. 2.4, and pick up the moving shadows and markers on the turntable.

Since depth-based and vision-based segmentation each excel at segmenting objects under different conditions, we combine the two to generate our final object segmentation. We take the segmentation from depth as the starting point. We then add pixels from the visual segmentation that are not in the background nor on the turntable by checking their depth values. Finally a filter is run on this segmentation mask to remove isolated pixels. The bottom row of Fig. 2.4 shows the resulting segmentation using combined depth and visual segmentation. The combined procedure provides high quality segmentations for all the objects. Fig. 2.5 shows examples of the resulting segmentation of objects from different categories. The left side shows 16 different objects each belonging to a different category, and the right side shows a pitcher and a leaf vegetable from eight different viewpoints.

2.4 Video Scene Annotation

In addition to the views of objects recorded using the turntable, the RGB-D Object Dataset also includes 22 video sequences of indoor environments, which we call the RGB-D Scenes Dataset. The scenes cover two office desk workspaces, two meeting areas, a living room, a coffee room, and a kitchen (see Fig. 2.6). The video sequences were recorded by holding the RGB-D camera at approximately human eye-level while walking around in each scene. Each video sequence contains several objects from the RGB-D Object Dataset. The objects are visible from different viewpoints and

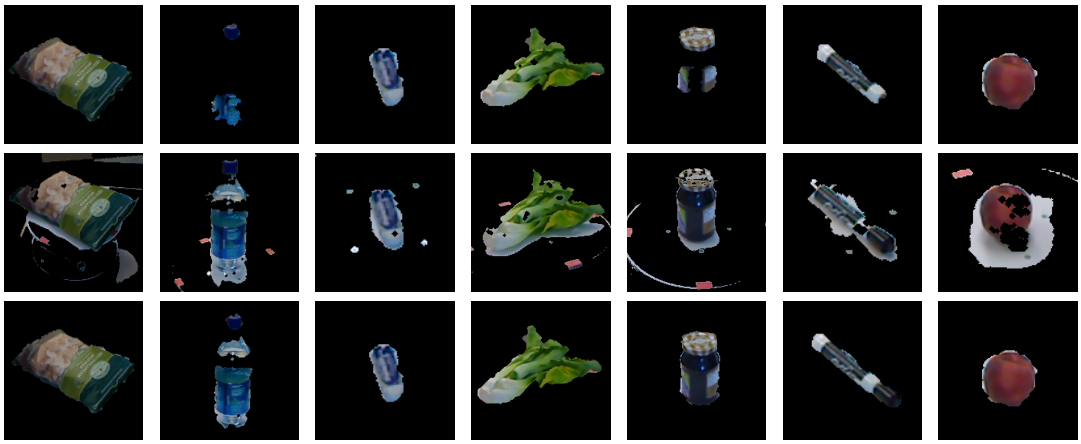

Figure 2.4: Segmentation examples, from left to right: bag of chips, water bottle, eraser, leaf vegetable, jar, marker and peach. Segmentation using depth only (top row), visual segmentation via background subtraction (middle row), and combined depth and visual segmentation (bottom row).

Figure 2.5: Segmented objects from the RGBD Object Dataset. (Left) 16 different objects from the dataset. (Right) 8 views of a pitcher (top) and 8 views of a leaf vegetable (bottom).

Figure 2.6: The RGB-D Scenes Dataset containing 22 indoor scenes. Each scene is a 3D point cloud created from an RGB-D video. Point-wise ground truth annotations of objects and furniture are available.

Video Sequence	# of Frames	# of Objects	# of Furniture
Desk_1	1748	3	0
Desk_2	1949	3	0
Desk_3	2328	4	0
Kitchen_small_1	2359	8	0
Meeting_small_1	3530	13	0
Table_1	2662	8	0
Table_small_1	2037	4	0
Table_small_2	1776	3	0
Living_room_1	888	5	3
Living_room_2	834	5	3
Living_room_3	861	5	3
Living_room_4	868	5	3
Meeting_area_1	1128	4	3
Meeting_area_2	1048	5	3
Meeting_area_3	943	5	3
Meeting_area_4	925	4	3
Coffee_room_1	732	3	4
Coffee_room_2	716	3	4
Coffee_room_3	640	3	4
Coffee_room_4	723	3	4
Workspace_1	462	4	1
Workspace_2	659	4	2

Table 2.1: Number of frames, objects, and furniture pieces in the 22 annotated videos of indoor scenes in the RGB-D Scenes Dataset.

Figure 2.7: (Left) 3D scene reconstruction of a kitchen scene with a cap highlighted in blue and a soda can in red using the labeling tool. (Right) Ground truth bounding boxes of the cap (top) and soda can (bottom) obtained by labeling the reconstruction.

distances and may be partially or completely occluded in some frames. 14 of the 22 videos also contain large pieces of furniture including chairs, coffee tables, tables, and sofas. Fig. 2.1 summarizes the number of frames and number of objects and furniture pieces in each video sequence. In Section 3.3 we demonstrate that the RGB-D Object Dataset can be used as training data for performing object detection in these videos. Here we will first describe how we annotated these videos with the ground truth label for every 3D point and bounding boxes of objects in the RGB-D Object Dataset. The standard way to do this would be to annotate video sequences one frame at a time using software like the LabelMe annotation tool [112] and more recently, *vatic* [139]. Temporal interpolation across video frames can somewhat alleviate this, but is only effective for a short sequence of frames if the camera trajectory is complex. Crowd-sourcing (e.g. Mechanical Turk) can also shorten annotation time, but does so merely by distributing the work across a larger number of people. We propose an alternative approach that is possible for RGB-D videos. Instead of labeling each video frame, we first stitch together the video sequence to create a 3D reconstruction of the entire scene, while simultaneously estimating the camera pose of each video frame. We label every 3D point in this reconstruction by hand. Fig. 2.7 (left) shows the reconstruction of a kitchen scene with a cap labeled in blue and a soda can labeled in red. This

semantic labeling of every point in the scene serves as ground truth for scene labeling experiments in Chapter 8. The labeled 3D points can also be projected back into each video frame using the estimated camera poses. This segmentation can be used to compute an object bounding box. While the estimated camera poses are not perfect, we found it to be accurate enough for obtaining good bounding boxes on the dataset in most cases without the need for manual correction. Fig. 2.7 (right) shows some bounding boxes obtained by projecting the labeled 3D points into several video frames.

Our labeling tool uses the technique proposed by Henry et al. [62] to construct 3D scene models from the RGB-D video frames. The Patch Volumes mapping approach builds 3D scene reconstructions by dividing the environment into patches, each of which is a locally planar piece of the scene. Each patch is reconstructed in much the same way as the KinectFusion [102] system. G2O [79] is used to perform pose graph optimization over the set of Patch Volumes to yield a globally-consistent scene reconstruction of room-sized environments. Please consult [62] for details about Patch Volumes Mapping.

2.5 Summary

In this chapter, we have presented a large-scale, hierarchical multi-view object dataset collected using an RGB-D camera. We have shown that depth information is very helpful for background subtraction, and video ground truth annotation via 3D reconstruction. The RGB-D Object Dataset and a set of software tools are publicly available at http://www.cs.washington.edu/rgbd-dataset. In subsequent chapters, we will use this dataset to evaluate features and algorithms for object recognition, object detection, and scene labeling of RGB-D data.

Chapter 3

FEATURES, ALGORITHMS, AND A LARGE-SCALE BENCHMARK

3.1 Background

Object recognition is the ability to recognize an object as being a member of a class. In other words, given a set of possible class labels and an object to classify, the object recognition task is to assign one of the labels to the object. This capability is a prerequisite for many applications, from web image search to a robot capable of understanding and executing the command "clear all the plates off the dinner table, but leave the bowls." In this chapter, we consider object recognition at two levels of granularity: category and instance. The granularity determines the set of classes that the system must distinguish. Category recognition involves recognizing an object as being a member of a certain semantic category of objects, for example coffee mug or soda can. In category recognition, the system has seen examples of objects from a fixed set of classes, and must now label objects that it may not have seen before as belonging to one of these classes. In contrast, the goal of instance recognition is to identify an object based on its unique appearance. For this task, the system must have already seen the object before, and must recall which particular object it is now seeing again. For example, suppose that a system has seen both Amelia's coffee mug and Bob's coffee mug, which have different appearance. The system is now presented with one of the mugs again, possibly in a new environment, and must now determine whether the mug is Amelia's mug or Bob's mug. Both category and instance recognition are useful in robotics applications. For example, category recognition is necessary for semantic understanding, such as responding to the command "clear all

the plates off the dinner table, but leave the bowls." On the other hand, instance recognition is required for responding to specific requests, such as when Amelia says "Fetch me my coffee mug."

In this chapter, we explore techniques that use existing state-of-the-art features and classifiers to address category and instance recognition in RGB-D data. We will consider techniques for both the object recognition and the object detection task. In object recognition, the system is presented with an image in which there is one dominant object, and the system must return that object's category and/or instance label. In object detection, the system is presented with an image that can contain zero or more objects, and the system must return not only the category and/or instance labels of all objects, but also their locations in the image by specifying bounding boxes.

We evaluate the techniques described in this chapter on the RGB-D Object Dataset to demonstrate the utility of combining RGB and 3D point cloud data. In addition, these results will also serve as a performance baseline that we will improve upon in subsequent chapters by designing new features and learning algorithms for RGB-D data.

3.2 Object Recognition Using the RGB-D Object Dataset

In this section, we introduce techniques for RGB-D based object recognition and detection and demonstrate that combining color and depth information can substantially improve the results achieved on our dataset. We evaluate our techniques on both category and instance recognition. The goal of these experiments is to test whether combining RGB and depth is helpful when well segmented or cropped object images are available. To the best of our knowledge, the RGB-D Object Dataset presented here is the largest multi-view dataset of objects where both RGB and depth images are provided for each view. To demonstrate the utility of having both RGB and depth information, in this section we present object recognition results on the

RGB-D Object Dataset using several different classifiers with only shape features, only visual features, and with both shape and visual features.

In object recognition the task is to assign a label (or class) to each query image. The possible labels that can be assigned are known ahead of time. State-of-the-art approaches to tackling this problem are usually supervised learning systems. A set of images is annotated with ground truth labels and given to a classifier, which learns a model for distinguishing between the different classes. We evaluate object recognition performance at two levels: category recognition and instance recognition. In category recognition, the system is trained on a set of objects. At test time, the system is presented with an RGB and depth image pair containing an object that was not present in training and the task is to assign a category label to the image (e.g. coffee mug or soda can). In instance recognition, the system is trained on a subset of views of each object. The task here is to distinguish between object instances (e.g. Pepsi can, Mountain Dew can, or Aquafina water bottle). At test time, the system is presented with an RGB and depth image pair that contains a previously unseen view of one of the objects and must assign an instance label to the image.

3.2.1 Data Setup

We subsampled the turntable data from the RGB-D Object Dataset by taking every fifth video frame, giving around 45000 RGB-D images. For category recognition, we randomly leave one object out from each category for testing and train the classifiers on all views of the remaining objects. For instance recognition, we consider two scenarios:

- Alternating contiguous frames: Divide each video into 3 contiguous sequences of equal length. There are 3 heights (videos) for each object, so this gives 9 video sequences for each instance. We randomly select 7 of these for training and test on the remaining 2.

- Leave-sequence-out: Train on the video sequences of each object where the camera is mounted 30° and 60° above the horizon and evaluate on the 45° video sequence.

We average accuracies across 10 trials for category recognition and instance recognition with alternating contiguous frames. There is no randomness in the data split for leave-sequence-out instance recognition so we report numbers for a single trial.

3.2.2 Feature Extraction

Each image is a view of an object and we extract one set of features capturing the shape of the view and another set capturing the visual appearance. In this chapter we use existing state-of-the-art features including spin images [70] from the shape retrieval community and SIFT descriptors [92] from the computer vision community. Shape features are extracted from the 3D locations of each depth pixel in physical space, expressed in the camera coordinate frame. We first compute spin images for a randomly subsampled set of 3D points. Each spin image is centered on a 3D point and captures the spatial distribution of points within its neighborhood. The distribution, captured in a two-dimensional 16×16 histogram, is invariant to rotation about the point normal. We use these spin images to compute efficient match kernel (EMK) features using random Fourier sets as proposed in [17]. EMK features are similar to bag-of-words (BOW) features in that they both take a set of local features (here spin images) and generate a fixed length feature vector describing the bag. EMK features approximate the gaussian kernel between local features, yielding a continuous measure of similarity that is more fine-grained than BOW. They were shown to outperform BOW features on several image-based visual recognition datasets including Scene-15, Caltech-101, and Caltech-256 [17]. To incorporate spatial information, we divide an axis-aligned bounding cube around each view into a $3 \times 3 \times 3$ grid. We compute a 1000-dimensional EMK feature in each of the 27 cells separately. We perform principal

component analysis (PCA) on the EMK features in each cell and take the first 100 components. Finally, we include as shape features the width, depth and height of a 3D bounding box around the view. This gives us a 2703-dimensional shape descriptor.

To capture the visual appearance of a view, we extract SIFT on a dense grid of 8×8 cells. To generate image-level features and capture spatial information we compute EMK features on two image scales. First we compute a 1000-dimensional EMK feature using SIFT descriptors from the entire image. Then we divide the image into a 2×2 grid and compute EMK features separately in each cell from only the SIFT features inside the cell. We perform PCA on each cell and take the first 300 components, giving a 1500-dimensional EMK SIFT feature vector. Additionally, we extract texton histograms [88] features, which capture texture information using oriented gaussian filter responses. The texton vocabulary is built from an independent set of images on LabelMe [112]. Finally, we include a color histogram and also use the mean and standard deviation of each color channel as visual features.

3.2.3 Classifiers

We evaluate the category and instance recognition performance of three state-of-the-art classifiers: linear support vector machine (LinSVM), gaussian kernel support vector machine (kSVM) [38, 25], and random forest (RF) [21, 47]. The parameters of these classifiers were tuned on a small subset of the RGB-D Object Dataset containing only five categories. We found that the SVM classifiers were insensitive to their parameters. The Random Forest classifier required significantly more tuning, and it may be possible to improve its results with additional parameter tuning.

3.2.4 Evaluation

Table 3.1 shows the classification performance of these classifiers using only shape features, only visual features, and using both shape and visual features. Overall visual features are more useful than shape features for both category level and instance level

recognition. However, shape features are relatively more useful in category recognition, while visual features are relatively more effective in instance recognition. This is exactly what we should expect, since a particular object instance has a fairly constant visual appearance across views, while objects in the same category can have different texture and color. On the other hand, shape tends to be stable across a category in many cases. The most interesting and significant conclusion is that combining both shape and visual features gives higher overall category-level performance regardless of classification technique. The features compliment each other, which demonstrates the value of a large-scale dataset that can provide both shape and visual information. For alternating contiguous frames instance recognition, using visual features alone already gives very high accuracy, so including shape features does not lead to further increase in performance. The leave-sequence-out evaluation is much more challenging, and here combining shape and visual features significantly improves accuracy.

We also ran a nearest neighbor classifier under the same experimental setup and using the same set of features and found that it performs much worse than learning-based approaches. For example, its performance on leave-sequence-out instance recognition when using all features is 43.2%, much worse than the accuracies reported in Table 3.1.

3.2.5 Remarks

The experiments in this section demonstrate that combining color and shape information is helpful. Gathering data from both sensor modalities traditionally required fusing data from two separate sensors, for example a camera and a laser rangefinder. When using two separate sensors, an elaborate calibration procedure is required to align the data streams from the two sensors in both space and time. With RGB-D cameras, this alignment is much easier and can in fact be accomplished in the SDK provided by PrimeSense, the sensor manufacturer. While the simple combination of existing image and point cloud based features described in this section achieves

Classifier	Shape	Vision	All
	Category		
LinSVM	53.1 ± 1.7	74.3 ± 3.3	81.9 ± 2.8
kSVM	64.7 ± 2.2	74.5 ± 3.1	83.8 ± 3.5
RF	66.8 ± 2.5	74.7 ± 3.6	79.6 ± 4.0
	Instance (Alternating contiguous frames)		
LinSVM	32.4 ± 0.5	90.9 ± 0.5	90.2 ± 0.6
kSVM	51.2 ± 0.8	91.0 ± 0.5	90.6 ± 0.6
RF	52.7 ± 1.0	90.1 ± 0.8	90.5 ± 0.4
	Instance (Leave-sequence-out)		
LinSVM	32.3	59.3	73.9
kSVM	46.2	60.7	74.8
RF	45.5	59.9	73.1

Table 3.1: Category and instance recognition performance of various classifiers on the RGB-D Object Dataset using shape features, visual features, and with all features. LinSVM is linear SVM, kSVM is gaussian kernel SVM, RF is random forest.

good results, Chapter 7 will demonstrate that features learned from RGB-D data specifically can further boost recognition accuracy.

3.3 Object Detection Using the RGB-D Object Dataset

Given an image, the object detection task is to identify and localize all objects of interest. Like in object recognition, the objects belong to a fixed set of class labels. The object detection task can also be performed at both the category and the instance level. The difference between object recognition and object detection is that in object recognition, the image is known to contain only one object and the system just has to identify which object it is. In object detection, the image can contain multiple objects and the system must localize all objects of interest by drawing bounding boxes around each object. In this section, we demonstrate how to use the RGB-D Object Dataset to perform object detection in real-world scenes.

3.3.1 Formulation

In this chapter, we consider an object detection system based on the existing standard sliding window approach [31, 43, 61], where the system evaluates a score function for all positions and scales in an image, and thresholds the scores to obtain object bounding boxes. Each detector window is of a fixed size and we search across 20 scales on an image pyramid. For efficiency, we consider a linear score function (so convolution can be applied for fast evaluation on the image pyramid). We perform non-maximum suppression to remove multiple overlapping detections.

Let H be the feature pyramid and p the position of a subwindow. p is a three-dimensional vector: the first two dimensions are the top-left position of the subwindow and the third one is the scale of the image. Our score function is

$$s_w(p) = w^\top \phi(H, p) + b \tag{3.1}$$

where w is the filter (weights), b the bias term, and $\phi(H, p)$ the feature vector at

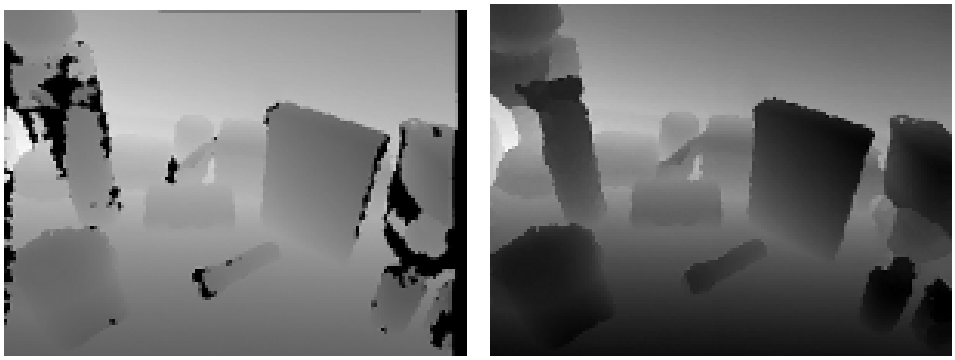

Figure 3.1: Original depth image (left) and filtered depth image using a recursive median filter (right). The black pixels in the left image are missing depth values.

position p. We train the filter w using a linear support vector machine (SVM):

$$L(w) = \frac{w^\top w}{2} + C \sum_{i=1}^{N} \max(0, 1 - y_i(w^\top x_i + b)) \tag{3.2}$$

where N is the training set size, $y_i \in \{-1, 1\}$ the labels, x_i the feature vector over a cropped image, and C the trade-off parameter.

3.3.2 Features

As features we use a variant of histogram of oriented gradients (HOG) [43], which has been found to work slightly better than the original HOG. This version considers both contrast sensitive and insensitive features, where the gradient orientations in each cell (8×8 pixel grid) are encoded using two different quantization levels into 18 ($0° - 360°$) and 9 orientation bins ($0° - 180°$), respectively. This yields a $4 \times (18 + 9) = 108$-dimensional feature vector. A 31-D analytic projection of the full 108-D feature vectors is used. The first 27 dimensions correspond to different orientation channels (18 contrast sensitive and 9 contrast insensitive). The last 4 dimensions capture the overall gradient energy in four blocks of 2×2 cells.

Aside from HOG over the RGB image, we also compute HOG over the depth image where each pixel value is the actual object-to-camera distance. Before extracting HOG features, we need to fill up the missing values in the depth image. Since the missing values tend to be grouped together, we use a recursive median filter. Instead of considering all neighboring pixel values, we take the median of the non-missing values in a 5×5 grid centered on the current pixel. We apply this median filter recursively until all missing values are filled. An example original depth image and the filtered depth image are shown in Fig. 3.1.

Finally, we also compute a feature that is sensitive to the true size of the object. While the depth of an object alone tells us nothing about its true size, consider the depth of an object divided by the size of its image bounding box. A larger object must be farther away (have larger depth) if it occupies the same amount of pixels on an image as a smaller object. Equivalently, if a pixel at a certain depth belongs to a particular object, that object's bounding box must be a certain width and height. Assuming that the depth is correct, this would allow one to compute the exact feature pyramid scale in which to search for the object. However, the depth is noisy and so we do not use it to constrain the scales that we search over. Instead, at each scale we compute the normalized depth $\frac{d}{s}$ of every pixel, where d is the depth, and s is the scale. We extract histograms of normalized depths in a regular grid over the image where each cell is 8×8 pixels. We used a histogram of 20 bins with each bin having a range of 0.15m. The normalized depth only matches the depth seen in training data if we are in the correct scale, but degrades gracefully as the scale is varied. This gives us a feature that is sensitive to the object's true size. Helmer et al. [61] also used depth information to inform object classification. However, the method of exploiting depth information is different from our approach: Helmer et al. used depth information to define a score function that is used as a prior while we construct a scale histogram feature from normalized depth values.

3.3.3 Training Procedure

The performance of the classifier heavily depends on the data used to train it. For object detection, there are many potential negative examples. A single image can be used to generate 10^5 negative examples for a sliding window classifier. Therefore, we use a bootstrapping hard negative mining procedure similar to Felzenszwalb et al. [43]. The positive examples are object windows we are interested in. The initial negative examples are randomly chosen from background images and object images from other categories/instances. The trained classifier is used to search images and select the false positives with the highest scores (hard negatives). These hard negatives are then added to the negative set and the classifier is retrained. This procedure is repeated 5 times to obtain the final classifier.

3.3.4 Evaluation

We evaluated the above object detection approach on the eight video sequences described in Section 2.4 that contain only tabletop objects and no annotated furniture. Since consecutive frames are very similar, we subsample the video data and run our detection algorithm on every 5th frame. We constructed 4 category-level detectors (bowl, cap, coffee mug,and soda can) and 20 instance-level detectors from the same categories. We follow the PASCAL Visual Object Challenge (VOC) evaluation metric. A candidate detection is considered correct if the size of the intersection of the predicted bounding box and the ground truth bounding box is more than half the size of their union. Only one of multiple successful detections for the same ground truth is considered correct, the rest are considered as false positives. We report precision-recall curves and average precision, which is computed from the precision-recall curve and is an approximation of the area under this curve. For multiple category/instance detections, we pool all candidate detection across categories/instances and images to generate a single precision-recall curve.

In Fig. 3.2 we show precision-recall curves comparing detection performance with a classifier trained using image features only (red), depth features only (green), and both (blue). We found that depth features (HOG over depth image and normalized depth histograms) are much better than HOG over RGB image. The main reason for this is that in depth images strong gradients are mostly from true object boundaries (see Fig. 3.1), which leads to many fewer false positives compared to HOG over RGB image, where color change can also lead to strong gradients. The best performance is attained by combining image and depth features. The combination gives higher precision across all recall levels than image only and depth only, if not comparable. In particular, combining image and depth features gives much higher precision when high recall is desired.

Fig. 3.3 shows multi-object detection results in three scenes. The leftmost scene contains three objects observed from a viewpoint significantly different than was seen in the training data. The multi-category detector is able to correctly detect all three objects, including a bowl that is partially occluded by a cereal box. The middle scene shows category-level detections in a very cluttered scene with many distracter objects. The system is able to correctly detect all objects except the partially occluded white bowl that is far away from the camera. Notice that the detector is able to identify multiple instances of the same category (caps and soda cans). The rightmost scene shows instance-level detections in a cluttered scene. Here the system was able to correctly detect both the bowl and the cap, even though the cap is partially occluded by the bowl. The current single-threaded implementation takes approximately 10 seconds to run the four object detectors to label each scene. However, both feature extraction over a regular grid and evaluating a sliding window detector are easily parallelizable. Hence, a GPU-based implementation of the described approach should be able to significantly speed up the system.

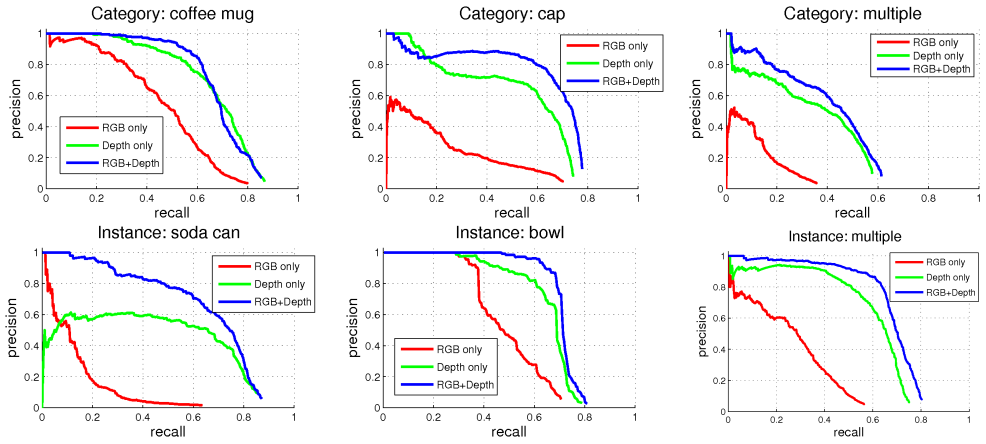

Figure 3.2: Precision-recall curves comparing performance with image features only (red), depth features only (green), and both (blue). The top row shows category-level results. From left to right, the first two plots show precision-recall curves for two binary category detectors, while the last plot shows precision-recall curves for the multi-category detector. The bottom row shows instance-level results. From left to right, the first two plots show precision-recall curves for two binary instance detectors, while the last plot shows precision-recall curves for the multi-instance detector.

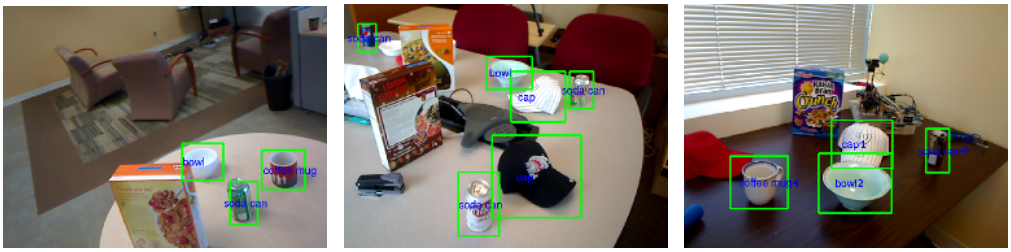

Figure 3.3: Three detection results in multi-object scenes. From left to right, the first two images show multi-category detection results, while the last image shows multi-instance detection results.

3.3.5 Remarks

In this section, we demonstrated that the depth channel provides useful information for object detection. Specifically, HOG features computed over depth images captures gradients that much more reliably correspond to the object silhouette than gradients extracted from RGB images. The silhouettes of objects like coffee mugs and soda cans is quite distinctive, leading to very good object detection performance using the depth channel alone. Nevertheless, as was the case for the object recognition task, combining color and depth also improves results over using either sensor modality alone for the object detection task.

3.4 Summary

In this chapter, we introduced techniques for object recognition and object detection based on existing state-of-the-art features and classifiers. The experimental results on category, instance, and pose recognition on the RGB-D Object Dataset demonstrate that combining color and depth information leads to better performance than just using either sensor modality alone. This strong result makes a convincing case for using RGB-D cameras in computer vision and robotics applications.

One limitation of the techniques presented in this chapter is that they do not explicitly address the problem of combining multiple features. Features can have different magnitudes and dimensionality, and those with larger magnitude and more dimensions will have more influence on the classifier, which can lead to suboptimal performance. While normalizing the features, which was done in the techniques presented here, can minimize the problem, it is not a perfect solution. In the next two chapters, we present a distance learning approach that uses training data to learn classifiers that weigh each feature so as to maximize discrimination between classes. In Chapter 4, we demonstrate how to use this technique for object recognition in 3D point clouds, where the scarcity of labeled data motivates us to use domain adapta-

tion to combine 3D models downloaded from a web repository and 3D point clouds collected from the real world. In Chapter 5, we modify the distance learning technique to work on RGB-D data.

Chapter 4

OBJECT RECOGNITION IN 3D POINT CLOUDS USING WEB DATA AND DOMAIN ADAPTATION

4.1 Background

A key problem in object recognition is the availability of sufficient labeled training data to learn classifiers. Typically, this is done by manually labeling data collected by the robot, eventually followed by a procedure to increase the diversity of that data set [116]. However, data labeling is error prone and extremely tedious. We thus conjecture that relying solely on manually labeled data does not scale to the complex environments robots will be deployed in.

The goal of this chapter is to develop techniques that significantly reduce the need for labeled training data for classification tasks in robotics by leveraging data available on the World Wide Web. Unfortunately, this is not as straightforward as it seems. A key problem is the fact that the data available on the World Wide Web is often very different from that collected by a mobile robot. For instance, a robot navigating through an urban environment will often observe cars and people from very close range and at angles different from those typically available in data sets such as LabelMe [112]. Furthermore, weather and lighting conditions might differ significantly from web-based images.

The difference between web-based data and real data collected by a robot is even more obvious in the context of classifying 3D point cloud data. A number of online shape databases have emerged in recent years, including the Princeton Shape Benchmark [124] and the Trimble 3D Warehouse [136]. The Trimble 3D Warehouse is particularly promising, as it is a publicly available database where anyone can con-

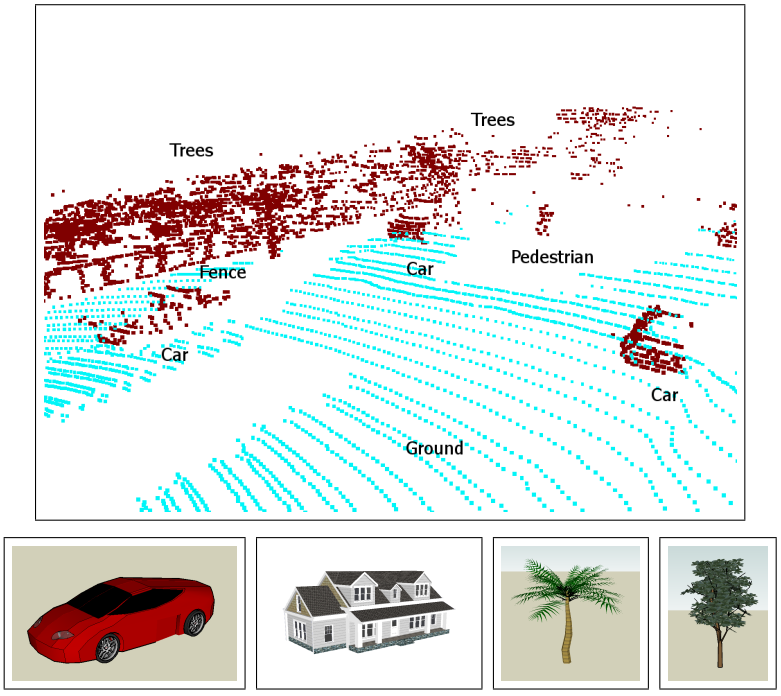

Figure 4.1: (Upper row) Part of a 3D laser scan taken in an urban environment (ground plane points shown in cyan). The scan contains multiple cars, a person, a fence, and trees in the background. (lower row) Example mesh models from the Trimble 3D Warehouse.

tribute models created using Trimble's SketchUp 3D modeling program. It already contains tens of thousands of user-contributed models such as cars, people, street signs, and many other objects. In this chapter, we use objects from Trimble 3D Warehouse to help classify 3D point clouds collected by mobile robots in urban terrain (see Fig. 4.1). We would like to leverage such an extremely rich source of freely available and labeled training data. However, virtually all objects in this dataset are created manually and thus do not accurately reflect the data observed by actual range sensing devices.

The aim of domain adaptation is to use large sets of labeled data from one domain along with a smaller set of labeled data from the target domain to learn a classifier that works well on the target domain. In this chapter we show how domain adaptation can be applied to the problem of object detection in 3D point clouds. The key idea of our approach is to learn a classifier based on objects from Trimble 3D Warehouse along with a small set of labeled point clouds. Our classification technique builds on an exemplar-based approach developed for visual object recognition [95]. To obtain a final labeling of individual 3D points, our system first performs segmentation on the point cloud based on the 3D point locations to generate individual segments, or object hypotheses. We then generate a soup of segments [94] by considering merges of these segments. Each segment is classified based on the labels of exemplars that are "close" to it. Closeness is measured via a learned distance function for spin image signatures [70, 5] and other shape features. We show how the learning technique can be extended to enable domain adaptation. In the experiments we demonstrate that additional data taken from the 3D Warehouse along with our domain adaptation greatly improves the classification accuracy on point clouds of real-world environments.

In this chapter, we describe an exemplar-based approach to semantic scene labeling in 3D point clouds first published in [84] and [85]. We enhance a technique developed for visual object recognition with 3D shape features and introduce a probabilistic, exemplar-based classification method. Our resulting approach significantly

outperforms alternative techniques such as boosting and support vector machines. We demonstrate how to leverage large, human-generated datasets such as Trimble 3D Warehouse to further increase the performance of shape-based object recognition. To do so, we introduce two techniques for *domain adaptation*, one based on previous work done in the context of natural language processing and one we developed specifically for our exemplar-based approach.

4.2 Learning Exemplar-based Distance Functions for 3D Point Clouds

In this section we describe the details of our approach to point cloud classification. We review the exemplar-based recognition technique introduced in [95]. While the approach was developed for vision-based recognition tasks, we show how to adapt the method to object recognition in point clouds. In a nutshell, the approach takes a set of labeled segments and learns a distance function for each segment, where the distance function is a linear combination of feature differences. The weights of this function are learned such that the decision boundary maximizes the margin between the associated subset of segments belonging to the same class and segments belonging to other classes.

4.2.1 Point Cloud Segmentation and Feature Extraction

Given a 3D point cloud of a scene, we first segment out points belonging to the ground from points belonging to potential objects of interest. In our indoor dataset, we assume that the objects are located on a table, which allows us to extract the ground plane via straightforward RANSAC plane fitting. For the more complex outdoor scenes, we first bin the points into grid cells of size $25 \times 25 \times 25 cm^3$, and run RANSAC plane fitting [44] on each cell to find the surface orientations of each grid cell. We take only the points belonging to grid cells whose orientations are less than 30 degrees with the horizontal and run RANSAC plane fitting again on all of these points to obtain the final ground plane estimation. The assumption here is that the ground has a

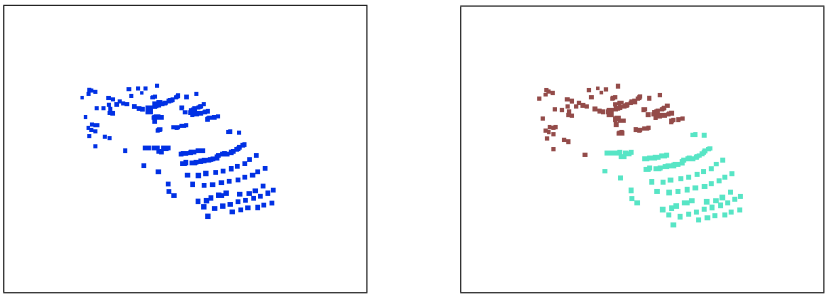

Figure 4.2: (left) Point cloud of a car extracted from a laser scan. (right) Segmentation via mean-shift. The soup of segments additionally contains a merged version of these segments.

slope of less than 30 degrees, which is usually the case and certainly for our data sets. Points close to the extracted plane are labeled as "ground" and not considered in the remainder of our approach. Fig. 4.1 displays a Velodyne LIDAR scan of a street scene with the extracted ground plane points shown in cyan.

Since the extent of each object is unknown, we perform segmentation based on the 3D locations of points to obtain individual object hypotheses. We experimented with the mean-shift clustering [28] and normalized cuts [123] algorithms at various parameter settings and found that the former provided better segmentation. In the context of vision-based recognition, [94] showed that it is beneficial to generate multiple possible segmentations of a scene, rather than relying on a single, possibly faulty segmentation. Similar to their technique, we generate a "soup of segments" using mean-shift clustering and considering merges between clusters of up to 3 neighboring segments. An example segmentation of a car automatically extracted from a complete scan is shown in Fig. 4.2. The soup also contains a segment resulting from merging the two segments.

We next extract a set of features capturing the shape of a segment. For each point, we compute spin image features [70], which are 16×16 matrices describing the local

shape around that point. Following the technique introduced by [5] in the context of object retrieval, we compute for each point a spin image signature, which compresses information from its spin image down to an 18-dimensional vector. Representing a segment using the spin image signatures of all its points would be impractical, so the final representation of a segment is composed of a smaller set of spin image signatures. In [5], this final set of signatures is obtained by clustering all spin image signatures describing an object. The resulting representation is rotation-invariant, which is beneficial for object retrieval. However, in our case the objects of concern usually appear in a constrained range of orientations. Cars and trees are unlikely to appear upside down, for example. The orientation of a segment is actually an important distinguishing feature and so unlike in [5], we partition the points into a $3 \times 3 \times 3$ grid and perform k-means clustering on the spin image signatures within each grid cell, with a fixed $k = 3$. Thus, we obtain for each segment $3 \cdot 3 \cdot 3 = 27$ shape descriptors of length $3 \cdot 18 = 54$ each. We also include as features the width, depth and height of the segment's bounding box, as well as the segment's minimum height above the ground. This gives us a total of $27 + 4 = 31$ descriptors.

To obtain similar representations of models in the 3D Warehouse, we perform ray casting on the models to generate point clouds and then perform the same procedure described in the previous paragraph (see Fig. 4.3).

4.2.2 Learning the Distance Function

Assume we have a set of n labeled point cloud segments, $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$. We refer to these segments as *exemplars*, e, since they serve as examples for the appearance of segments belonging to a certain class. Let $\mathbf{f}_e$ denote the features describing an exemplar e, and let $\mathbf{f}_z$ denote the features of an arbitrary segment z, which could also be an exemplar. $\mathbf{d}_{ez}$ is the vector containing component-wise, L_2 distances between individual features describing e and z: $\mathbf{d}_{ez}[i] = ||\mathbf{f}_e[i] - \mathbf{f}_z[i]||$. In our case, features $\mathbf{f}_e$ and $\mathbf{f}_z$ are the 31 descriptors describing segment e and segment z, respectively. $\mathbf{d}_{ez}$ is a

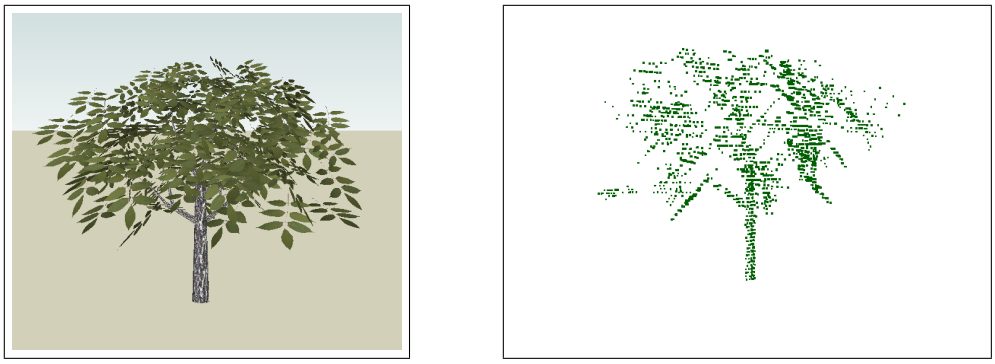

Figure 4.3: (left) Tree model from the 3D Warehouse and (right) point cloud extracted via ray casting.

31+1 dimensional distance vector where each component, i, is the L_2 distance between feature i of segments e and z, with an additional bias term as described in [95]. A more fine-grained division of features, such as computing L_2 distance between each feature dimension separately may yield better performing distance functions, but at the cost of more computation. Distance functions between two segments are linear functions of their distance vector. Each exemplar has its own distance function, D_e, specified by the weight vector $\mathbf{w}_e$:

$$D_e(z) = \mathbf{w}_e^{\mathsf{T}} \mathbf{d}_{ez} \tag{4.1}$$

Note that since each exemplar has its own set of weights, the functions do not define a true distance metric, as it is asymmetric. Instead, a given exemplar e's function evaluates the similarity of other exemplars to e based on a particular weighing of feature differences learned specifically for e. This is advantageous since different exemplars may have different sets of features that are better for distinguishing itself from other exemplars.

To learn the weights of this distance function, we follow the approach first proposed in [95]. We define a binary vector $\boldsymbol{\alpha}_e$, the length of which is given by the number

of exemplars with the same label as e. During learning, $\boldsymbol{\alpha}_e$ is non-zero for those exemplars that are in e's class and that should be similar to e, and zero for those that are in the same class but considered irrelevant for exemplar e. The key idea behind these vectors is that even within a class, different segments can have very different feature appearance. This could depend, for example, on the angle from which an object is observed.

The values of $\boldsymbol{\alpha}_e$ and $\mathbf{w}_e$ are determined for each exemplar separately by the following optimization:

$$\begin{aligned}\{\mathbf{w}_e^*, \boldsymbol{\alpha}_e^*\} = \underset{\mathbf{w}_e, \boldsymbol{\alpha}_e}{\operatorname{argmin}} \sum_{i \in \mathcal{C}_e} \boldsymbol{\alpha}_{ei} L(-\mathbf{w}_e^\mathsf{T} \mathbf{d}_{ei}) + \sum_{i \notin \mathcal{C}_e} L(\mathbf{w}_e^\mathsf{T} \mathbf{d}_{ei}) \\ \text{subject to } \mathbf{w}_e \geq 0;\ \boldsymbol{\alpha}_{ei} \in \{0, 1\};\ \sum_i \boldsymbol{\alpha}_{ei} = K \end{aligned} \tag{4.2}$$

Here, $\mathcal{C}_e$ is the set of examplars that belong to the same class as e, $\boldsymbol{\alpha}_{ei}$ is the i-th component of $\boldsymbol{\alpha}_e$, and L is the squared hinge loss function, $L(a) = \max(0, 1 - a)^2$. The constraints ensure that K values of $\boldsymbol{\alpha}_e$ are non-zero. In (4.2), the K positive exemplars are considered via the non-zero terms in the first summation, and the negative exemplars are given in the second summation. The resulting optimization aims at maximizing the margin of a decision boundary that has K segments from e's class on one side, while keeping exemplars from other classes on the other side. The optimization procedure alternates between two steps. The $\boldsymbol{\alpha}_e$ vector in the k-th iteration is chosen such that it minimizes the first sum in (4.2):

$$\boldsymbol{\alpha}_e^k = \underset{\boldsymbol{\alpha}_e}{\operatorname{argmin}} \sum_{i \in \mathcal{C}_e} \boldsymbol{\alpha}_{ei} L(-\mathbf{w}_e^{k\mathsf{T}} \mathbf{d}_{ei}) \tag{4.3}$$

This is done by simply setting $\boldsymbol{\alpha}_e^k$ to 1 for the K smallest values of $L(-\mathbf{w}_e^\mathsf{T} \mathbf{d}_{ei})$, and setting it to zero otherwise. The next step fixes $\boldsymbol{\alpha}_e$ to $\boldsymbol{\alpha}_e^k$ and optimizes (4.2) to yield the new $\mathbf{w}_e^{k+1}$:

$$\mathbf{w}_e^{k+1} = \underset{\mathbf{w}_e}{\operatorname{argmin}} \sum_{i : \in \mathcal{C}_e} \boldsymbol{\alpha}_{ei}^k L(-\mathbf{w}_e^\mathsf{T} \mathbf{d}_{ei}) + \sum_{i \notin \mathcal{C}_e} L(\mathbf{w}_e^\mathsf{T} \mathbf{d}_{ei}) \tag{4.4}$$

When choosing the loss function L to be the squared hinge loss function, this optimization yields standard Support Vector Machine learning [19]. The iterative procedure converges when $\boldsymbol{\alpha}_e^k = \boldsymbol{\alpha}_e^{k+1}$. [95] showed that the learned distance functions provide excellent recognition results for image segments.

4.3 Domain Adaptation

So far, the approach assumes that the exemplars in the training set $\mathcal{E}$ are drawn from the same distribution as the segments on which the approach will be applied. While this worked well in [95], it does not perform well when training and test domain are significantly different. In our scenario, for example, the classification is applied to segments extracted from 3D point clouds, while most of the training data is extracted from the 3D Warehouse data set. As we will show in the experimental results, combining training data from both domains can improve classification over just using data from either domain, but this performance gain cannot be achieved by simply combining data from the two domains into a single training set.

In general, we distinguish between two domains. The first one, the *target domain*, is the domain on which the classifier will be applied after training. The second domain, the *source domain*, differs from the target domain but provides additional data that can help to learn a good classifier for the target domain. In our context, the training data now consists of exemplars chosen from these two domains: $\mathcal{E} = \mathcal{E}^t \cup \mathcal{E}^s$. Here, $\mathcal{E}^t$ contains exemplars from the target domain, that is, labeled segments extracted from the real laser data. $\mathcal{E}^s$ contains segments extracted from the 3D Warehouse. As typical in domain adaptation, we assume that we have substantially more labeled data from the source domain than from the target domain: $|\mathcal{E}^s| \gg |\mathcal{E}^t|$. We now describe two methods for domain adaptation in the context of the exemplar-based learning technique.

4.3.1 Domain Adaptation via Feature Augmentation

[32] introduced feature augmentation as a general approach to domain adaptation. It is extremely easy to implement and has been shown to outperform various other domain adaptation techniques and to perform as well as the thus far most successful approach to domain adaptation [33]. The approach performs adaptation by generating a stacked feature vector from the original features used by the underlying learning technique. Specifically, let $\mathbf{f}_e$ be the feature vector describing exemplar e. The approach in [32] generates a stacked vector $\mathbf{f}_e^*$ as follows:

$$\mathbf{f}_e^* = \begin{pmatrix} \mathbf{f}_e \\ \mathbf{f}_e^s \\ \mathbf{f}_e^t \end{pmatrix} \tag{4.5}$$

Here, $\mathbf{f}_e^s = \mathbf{f}_e$ if e belongs to the source domain, and $\mathbf{f}_e^s = \mathbf{0}$ if it belongs to the target domain. Similarly, $\mathbf{f}_e^t = \mathbf{f}_e$ if e belongs to the target domain, and $\mathbf{f}_e^t = \mathbf{0}$ otherwise. Using the stacked feature vector, it becomes clear that exemplars from the same domain are automatically closer to each other in feature space than exemplars from different domains. [32] argued that this approach works well since data points from the target domain have more influence than source domain points when making predictions about test data.

4.3.2 Domain Adaptation for Exemplar-based Learning

We now present a method for domain adaptation specifically designed for the exemplar-based learning approach. The key difference between our domain adaptation technique and the single domain approach described in Section 4.2 lies in the specification of the binary vector $\boldsymbol{\alpha}_e$. Instead of treating all exemplars in the class of e the same way, we distinguish between exemplars in the source and the target domain. Specifically, we use the binary vectors $\boldsymbol{\alpha}_e^s$ and $\boldsymbol{\alpha}_e^t$ for the exemplars in these two domains.

The domain adaptation objective becomes

$$\{\mathbf{w}_e^*, \boldsymbol{\alpha}_e^{s*}, \boldsymbol{\alpha}_e^{t*}\} = \underset{\mathbf{w}_e, \boldsymbol{\alpha}_e^s, \boldsymbol{\alpha}_e^t}{\operatorname{argmin}} \sum_{i \in \mathcal{C}_e^s} \boldsymbol{\alpha}_{ei}^s L(-\mathbf{w}_e^\mathsf{T} \mathbf{d}_{ei}) + \sum_{i \in \mathcal{C}_e^t} \boldsymbol{\alpha}_{ei}^t L(-\mathbf{w}_e^\mathsf{T} \mathbf{d}_{ei}) + \sum_{i \notin \mathcal{C}_e} L(\mathbf{w}_e^\mathsf{T} \mathbf{d}_{ei}) \ , \quad (4.6)$$

where $\mathcal{C}_e^s$ and $\mathcal{C}_e^t$ are the source and target domain exemplars with the same label as e. The constraints are virtually identical to those for the single domain objective (4.2), with the constraints on the vectors becoming $\sum_i \boldsymbol{\alpha}_{ei}^s = K^s$ and $\sum_i \boldsymbol{\alpha}_{ei}^t = K^t$. The values for K^s and K^t give the number of source and target exemplars that must be considered during the optimization.

The subtle difference between (4.6) and (4.2) has a substantial effect on the learned distance function. To see this, imagine the case where we train the distance function of an exemplar from the source domain. Naturally, this exemplar will be closer to source domain exemplars from the same class than to target domain exemplars from that class. In the extreme case, the vectors determined via (4.3) will contain 1s only for source domain exemplars, while they are zero for all target domain exemplars. The single domain training algorithm will thus not take target domain exemplars into account and learn distance functions for source domain exemplars that are good in classifying source domain data. There is no incentive to make them classify target domain exemplars well. By keeping two different α-vectors, we can force the algorithm to optimize for classification on the target domain as well. The values for K^s and K^t allow us to trade off the impact of target and source domain data. They are determined via grid search using cross-validation, where the values that maximize the area under the precision-recall curve are chosen.

The learning algorithm is very similar to the single domain algorithm. In the k-th iteration, optimization of the α-vectors is done by setting $\boldsymbol{\alpha}_e^{s\,k}$ and $\boldsymbol{\alpha}_e^{t\,k}$ to 1 for the exemplars yielding the K^s and K^t smallest loss values, respectively. Then, the weights $\mathbf{w}_e^{k+1}$ are determined via convex SVM optimization [74] using the most recent α-vectors within (4.6).

4.4 Probabilistic Classification

To determine the class of a new segment, z, we first determine the set of associated exemplars, which are those for which $D_e(z) \leq 0$. This corresponds to all exemplars e for which z fall on e's side of the SVM decision boundary. [95] showed that this threshold is not only natural, but also empirically gave good performance. We found this to be the case as well.

In [95], segment z is labeled with the majority class among the associated exemplars. However, this approach does not model the reliability of individual exemplars and does not lend itself naturally to a probabilistic interpretation. Furthermore, it does not take into account that the target domain is different from the source domain.

To overcome these limitations, we choose the following Naïve Bayes model over exemplars. We define the class-conditional probability for each exemplar e in the training set to be

$$p(e \mid c) := \frac{\mid \{e' \mid D_e(e') \leq 0\} \mid}{N_c}, \tag{4.7}$$

where e' are target domain training exemplars in class c and N_c is the number of target domain training exemplars in class c. (4.7) states that the class-conditional probability is the proportion of exemplars e' in class c that are close to e ($D_e(e') \leq 0$). We use only target domain exemplars because the ultimate goal is to label segments from the target domain only.

Given a set of exemplars $\mathcal{E}$, the class-conditional probability of a test segment z is defined to be

$$p(z \mid c) := \prod_{e \in \mathcal{E} \ \wedge \ D_e(z) \leq 0} p(e \mid c) \ / \ \sum_{c'} \prod_{e \in \mathcal{E} \ \wedge \ D_e(z) \leq 0} p(e \mid c'). \tag{4.8}$$

Here we have assumed independence between the class-conditional probability distributions over exemplars. Intuitively, the class-conditional distribution of z should be similar to that of exemplars that are similar to it and (4.8) captures this. The

denominator is a normalization factor to ensure that we have indeed defined a probability distribution. Applying Bayes' rule, the probability that segment z belongs to class c is simply

$$p(c \mid z) = \frac{p(c)\ p(z \mid c)}{\sum_{c'} p(c')\ p(z \mid c')}. \tag{4.9}$$

The prior $p(c)$ is estimated via class frequencies in the target domain training data.

We can apply the results of the above segment classification to individual points. As described in Section 4.2.1, we extract a soup of segments from a 3D point cloud. Thus, each point may belong to multiple segments. Using the probability distributions over the classes of these segments, the distribution over the class of a single point l is given by

$$p(c \mid l) \ \propto \prod_{z \in Z_l} p(c \mid z), \tag{4.10}$$

where Z_l is the set of segments that contain point l. This combines the class hypotheses from multiple segments in the "soup" in a probabilistic manner to produce the final classification. In our setup, points in a test scene are assigned to the class with the highest probability.

4.5 Experiments

We evaluate the different approaches to 3D point cloud classification mostly in the context of outdoor scenes. The task here is to segment and classify point clouds collected in an urban environment into the following seven classes: cars, people, trees, street signs, fences, buildings, and background. Our experiments demonstrate that our two domain adaptation methods lead to improvements over approaches without domain adaptation and alternatives including LogitBoost [48] and a regular Multi-class SVM [25]. In particular, our exemplar-based domain adaptation approach obtains the best performance.

Our exemplar-based learning code is based on a MATLAB implementation provided by Malisiewicz [95]. The distance function learning takes around 15 minutes. We implemented the classification phase as a single-threaded C++ application. It takes on average 80 seconds to classify a scene. The majority of the time (60 seconds) is consumed by segmentation and feature extraction. Computing the distances between every test segment to every training exemplar currently takes 10 seconds. Much of the procedure, including the distance computation, operates on the different test segments and training exemplars independently; the code is highly parallelizable. Hence, a multi-threaded CPU or GPU implementation should speed this up to near real-time performance.

4.5.1 Urban Driving Data Set

We evaluated our approach using models from the Trimble 3D Warehouse as our source domain set, $\mathcal{E}^s$, and ten labeled street scenes as our target domain set, $\mathcal{E}^t$. The ten scenes, collected by a Velodyne LIDAR mounted on a vehicle navigating through the Boston area, were chosen so that they did not overlap spatially. Each scene is a single rotation of the LIDAR, yielding a cloud of nearly 70,000 points. Scenes may contain objects including, but not limited to, cars, bicycles, buildings, pedestrians and street signs. Camera images taken at one of these scenes are shown in Fig. 4.4. Manual labeling of 3D scans was done by inspecting the camera data collected along with the laser data. We automatically downloaded the first 100 models of each of cars, people, trees, street signs, fences and buildings from the Trimble 3D Warehouse and manually pruned out low quality models, leaving around 50 models for each class. We also included a number of models to serve as the background class, consisting of various other objects that commonly appear in street scenes, such as garbage cans, traffic barrels and fire hydrants. Recall that orientation information is preserved in our feature representation. To account for natural orientations that the objects can take in the environment, we generated 10 simulated laser scans from evenly-spaced

Figure 4.4: A scene from the urban driving data set. Starting with the image in the top left and going clockwise, they are captured with left-, forward-, right-, and rear-facing cameras mounted on the vehicle.

Approach	Training Data		Domain Adaptation			
	3D Warehouse	Real Scans	None	Simple	Stacked	Alpha
3DW	x		x			
Real		x	x			
3DW+Real,Simple	x	x		x		
3DW+Real,Stacked	x	x			x	
3DW+Real,Alpha	x	x				x
Boosting	x	x		x		
Multi-class SVM	x	x		x		

Table 4.1: Table summarizing the training data and domain adaptation methods used in the approaches compared in Section 4.5.2.

viewpoints around each of the downloaded models, giving us a total of around 3,200 exemplars in the source domain set. The ten labeled scans totaled to around 400 exemplars in the six actual object classes. We generate a "soup of segments" from these exemplars, using the data points in real scans not belonging to the six actual classes as candidates for additional background class exemplars. After this process, we obtain a total of 4,900 source domain segments and 2,400 target domain segments.

4.5.2 Comparison with Alternative Approaches

We compare the classification performance of our exemplar-based domain adaptation approach to several approaches, including training the single domain exemplar-based technique only on Warehouse exemplars, training it only on the real scans, and training it on a mix of Warehouse objects and labeled scans. The last combination can be viewed as a naïve form of domain adaptation. We also tested the stacked feature approach to domain adaptation described in Section 4.3.1. The different approaches

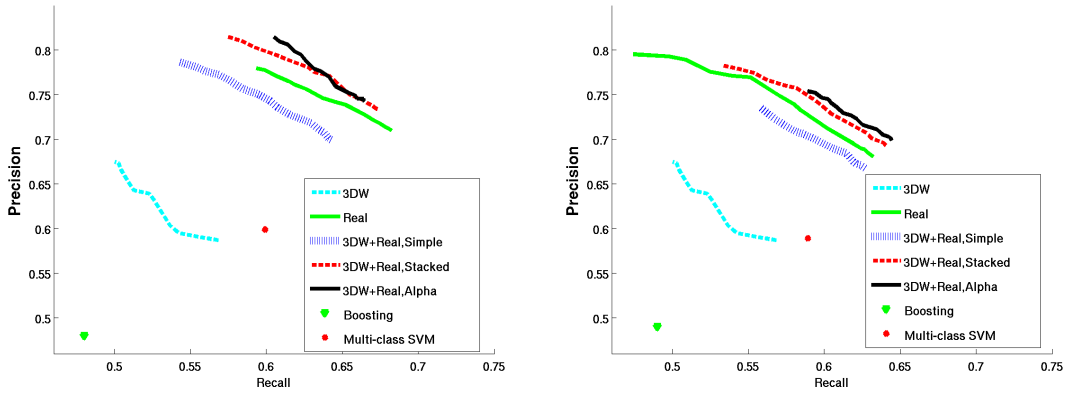

Figure 4.5: Precision-recall curves comparing performance of the various approaches trained using five (left) and three (right) real scans where applicable.

being compared are summarized in Table 4.1. "3DW" stands for exemplars from the 3D Warehouse, and "Real" stands for exemplars extracted from the Velodyne laser scans. Where exemplars from both the 3D Warehouse and real scans are used, we also specify the domain adaptation technique used. By "Simple" we denote the naïve adaptation of only mixing real and Warehouse data. "Stacked" refers to the stacked feature approach, applied to the single domain exemplar technique. Finally, "Alpha" is our exemplar-based domain adaptation technique.

The optimal K values (number of non-zero elements in the $\boldsymbol{\alpha}$ vectors) for each approach were determined separately using grid search and cross validation. Where training involves using real scans, we repeated each experiment 10 times using random train/test splits of the 10 total available scans. Each labeled scan contains around 240 segments on average.

The results are summarized in Fig. 4.5. Here the probabilistic classification described in Section 4.4 was used and the precision-recall curves were generated by varying the probabilistic classification threshold between $[0.5, 1]$. The precision and recall values are calculated on a per-point basis over entire scenes, including all seven

object classes, but omitting the ground plane points. Note that this evaluation criterion is different from the evaluation used in [95], where they considered any correctly labeled segment with an overlap of more than 50% with a ground truth object to be correct. Each curve in Fig. 4.5 corresponds to a different experimental setup. The left plot shows the approaches trained on five real scans, while the right plot shows the approaches trained on three real scans. All approaches are tested only on the remaining real scans that were not seen during training. Note that since the first setup (3DW) does not use real laser scans for training, the curves for this approach on the two plots are identical.

It comes as no surprise that training on Warehouse exemplars only performs worst. This result confirms the fact that the two domains actually have rather different characteristics. For instance, the windshields of cars are invisible to the Velodyne laser, thereby causing a large hole in the object segment. In Warehouse cars, however, the windshields are considered solid, causing a locally very different point cloud. Also, Warehouse models, created largely by casual hobbyists, tend to be composed of simple geometric primitives, while the shape of objects from real data can be both more complex and more noisy.

The naïve approach of training on a mix of both Warehouse and real scans does not perform well. In fact, it leads to worse performance than just training on real scans alone. This shows that domain adaptation is indeed necessary when incorporating training data from multiple domains. Both domain adaptation approaches outperform the approaches without domain adaptation. Our exemplar-based approach marginally outperforms the stacked feature approach when target domain training data is very scarce (when trained with only 3 real scans).

To gauge the overall difficulty of the classification task, we also trained two baseline classifiers, LogitBoost [48] and Multi-class (one-versus-all) SVM [25], on the mix of Warehouse and real scans. We evaluated these two baselines in the same manner as the approaches described above. We used the implementation of LogitBoost in Weka [58]

and the implementation of multi-class SVMs in LibSVM [25]. Parameters were tuned via cross-validation on the training set. The precision-recall values for these two approaches are shown in Fig. 4.5. We do not show the full curves since LogitBoost gave very peaked class distributions and there is no single value to threshold on in a one-versus-all Multi-class SVM.

In an application like autonomously driving vehicles, recall and precision are equally important. The vehicle must detect as many objects on the road as possible (high recall), and try to identify them correctly (high precision). Thus, the F-score is a good measure of the overall capability. The F-score is the harmonic mean between precision and recall: $F = 2 \cdot Precision \cdot Recall/(Precision + Recall)$ [137]. LogitBoost achieved a maximum F-score of 0.48 when trained on five scans, and a maximum F-score of 0.49 when trained on three scans, while the multi-class SVM achieved a maximum F-score of 0.60 when trained on five scans, and 0.59 when trained on three scans. (see Fig. 4.5). As a comparison, our approach achieves an F-score of 0.70 when trained on five scans and 0.67 when trained on three scans. The inferior results achieved by LogitBoost and the multi-class SVM demonstrate that this is not a trivial classification problem and that the exemplar-based approach is an extremely promising technique for 3D point cloud classification. Also, there is no significant degradation in performance between training on five scans and training on three scans.

4.5.3 Urban Data Set Examples

Fig. 4.6 provides examples of exemplars matched to the three laser segments shown in the panels in the left column. The top row gives ordered matches for the car segment on the left, the middle and bottom row show matches for a person and tree segment, respectively. As can be seen, the segments extracted from the real scans are successfully matched against segments from both domains, real and 3D Warehouse. The person is mismatched with one object from the background class

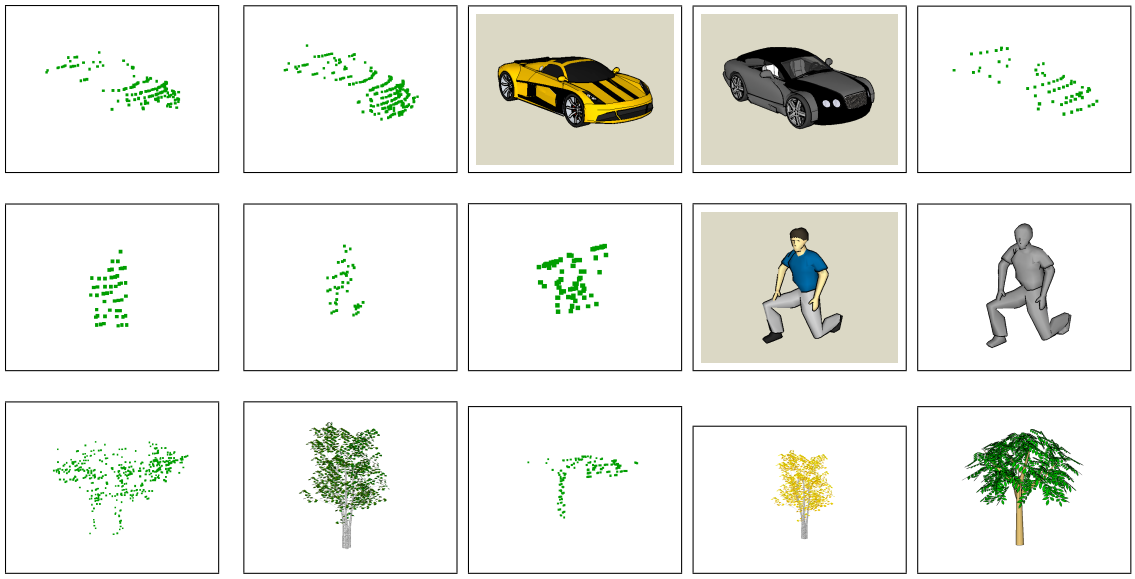

Figure 4.6: Exemplar matches. The leftmost column shows example segments extracted from 3D laser scans: car, person, tree (top to bottom). Second to last columns show exemplars with distance below threshold, closer exemplars are further to the left.

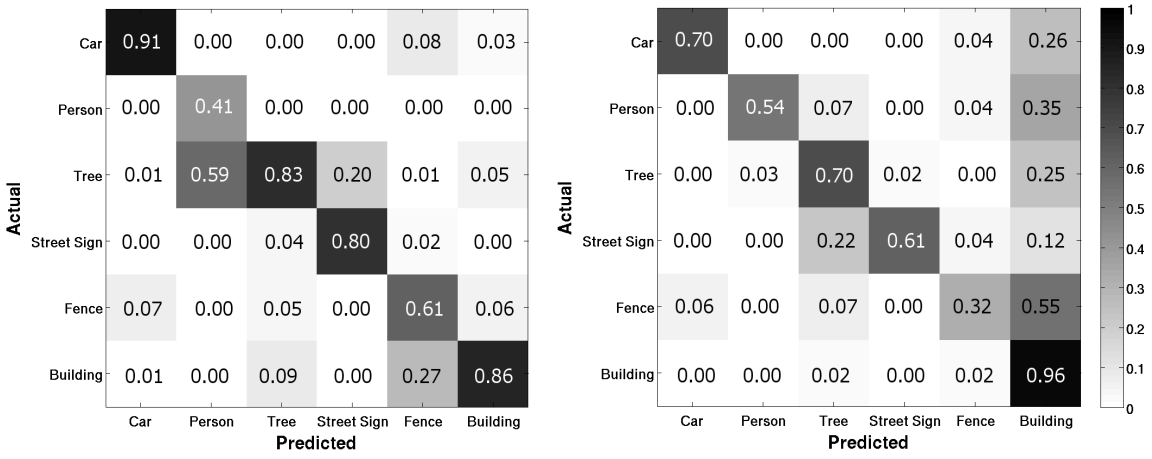

Figure 4.7: Confusion matrices between the six urban object classes. (left) Column-normalized precision matrix, (right) row-normalized recall matrix.

"other" (second row, third column). Part of the laser scan from the scene in Fig. 4.4 and its ground truth labeling is shown in color in Fig. 4.10. These figures also include the labeling achieved by our exemplar-based domain adaptation approach described in Section 4.3.2. Fig. 4.7 presents both the precision (column-normalized) and recall (row-normalized) confusion matrices between the six classes over all 10 scenes.

4.5.4 Feature Evaluation

To verify that all of the selected features contribute to the success of our approach, we also compared the performance of our approach using three different sets of features. We looked at using just bounding box dimensions and the minimum height off the ground (Dims), adding in the original, rotation-invariant spin image signatures as described in [5] (SIS_O + Dims), and adding in our $3 \times 3 \times 3$ grid of Spin Image Signatures (SIS_G + Dims). Fig. 4.8 (left) shows the precision-recall curves obtained by the three sets of features. Once again the precision-recall curves are generated by varying the probabilistic classification threshold between $[0.5, 1]$. Although using just dimensions features or the original spin image signatures can lead to higher precision values, this comes at the cost of much lower recall.

When trained on 3 scans (randomly selected and repeated for 10 trials) using dimensions features only, our approach achieves a maximum F-score of 0.63. Using the original Spin Image Signatures and dimensions features, we achieved an F-score of 0.64. Finally, using our Grid Spin Image Signatures and dimensions features achieved an F-score of 0.67. Due to noise and occlusions in the scans, as well as imperfect segmentation, the classes are not easily separable just based on bounding box dimensions alone. Also, our Grid Spin Image Signature features perform better than the original, rotation-invariant, Spin Image Signatures, justifying our modification to remove their rotation-invariance.

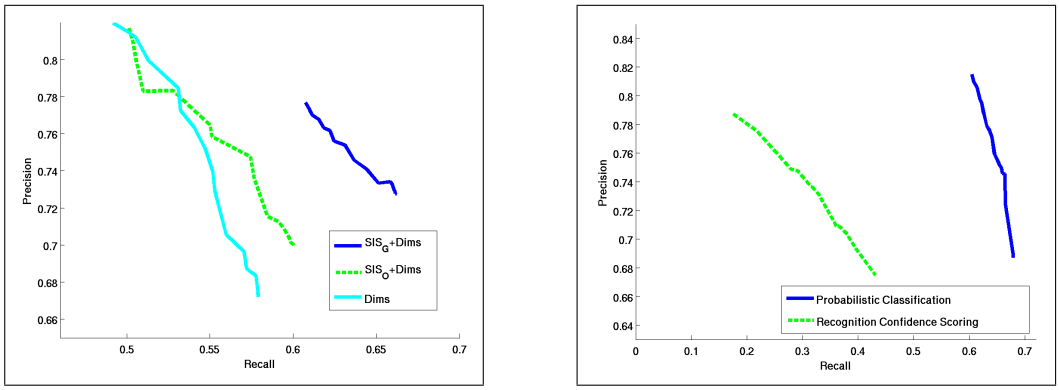

Figure 4.8: Precision-recall curves comparing different approaches: (left) using various sets of features. (right) Using our probabilistic classification and recognition confidence scoring.

4.5.5 Classification Method Comparison

In this experiment, we compared our probabilistic classification approach to the recognition confidence scoring method described in [95]. To use this scoring function, the distances must first be rescaled so that 0 denotes perfect similarity and 1 represents the decision boundary (as opposed to 0 in the standard SVM formulation). Letting E be the list of exemplars associated with segment z (i.e. $E = \{e \mid D_e(z) \leq 1\}$), the recognition confidence is defined as

$$s(z, E) = 1/\sum_{e \in E} \frac{1}{D_e(z)} \tag{4.11}$$

The intuition here is that a lower score is better, and this is attained by having many exemplars with low distances to our segment z. The resulting precision-recall curves are shown in Fig. 4.8 (right). Just as in the previous experiments, the precision-recall curve for our probabilistic classification is generated by varying the probability threshold between $[0, 1]$. The precision-recall curve for recognition confidence scoring is generated by thresholding on the recognition confidence, with the label for each point being the majority label of all segments containing that point. For clarity, only

the result from training on 5 scans (randomly selected and repeated for 10 trials) is shown, but the trend from training on 3 scans is identical. As evident from the plot, the probabilistic classification method attains recalls between 30-50 percentage points above recognition confidence scoring for corresponding precision values.

4.5.6 Effect of Varying the K Parameters

Two important parameters in our domain adaptation approach are K^s and K^t, which control the number of source domain and target domain exemplars to associate with each exemplar during the distance function learning process. The optimal setting for these parameters depends on a number of factors. The absolute number of K^s and K^t depends on the number of training exemplars and the amount of intra-class variation, while the ratio between the two depends on how different the source and target domains are from each other. K^s and K^t are determined via a grid search over possible settings of these two parameters. Cross-validation within the training data is used to select the setting that yields the highest F-score. Fig. 4.9 shows a plot of how the performance of our approach varies with different settings of K^s and K^t. A higher value for K^t than K^s is favored, which is to be expected since target domain exemplars tend to be more useful than source domain exemplars. Although the approach does best when K^s is set to be low, source domain exemplars are not completely ignored. They still play an important role as negative exemplars. Increasing either parameter beyond a certain value leads to degradations in performance. There is a single mode around which the maximum performance is attained. The specific optimum setting was found to be $K^s = 3$, $K^t = 15$.

4.6 Related Work

The problem of object recognition has been studied extensively by the computer vision community. Recently, there has been a focus on using large, web-based data sets for object and scene recognition [90, 95, 112, 134] and scene completion [59]. These

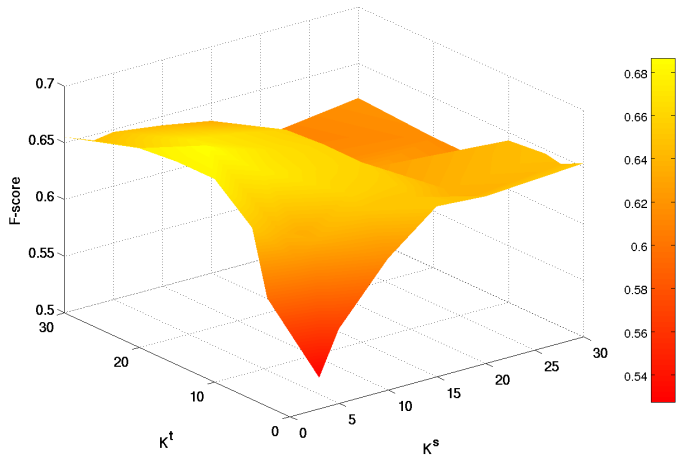

Figure 4.9: F-scores for different values of the K^s and K^t parameters used during training of the exemplar-based domain adaptation approach.

techniques take a radically different approach to the computer vision problem; they tackle the complexity of the visual world by using millions of weakly labeled images along with non-parametric techniques instead of parametric, model-based approaches. The goal of our work is similar to these previous works, but these approaches have been both trained and evaluated on web-based data. In our case, we are applying the learned classifier to shape-based object recognition using data collected from a robot, which can have characteristics very different from the web-based data used to train the system.

The shape retrieval community has designed many 3D shape description features and methods for searching through a database to retrieve objects that match a given query object. Shape retrieval methods have been proposed using a number of features including 3D shape contexts [77], 3D Zernike descriptors [103], and spin image signatures [70, 5]. However, the focus of this line of work is retrieving similar objects rather than the classification of the query object. Our approach uses one particular shape descriptor, spin image signatures, from this community, but with a modification that

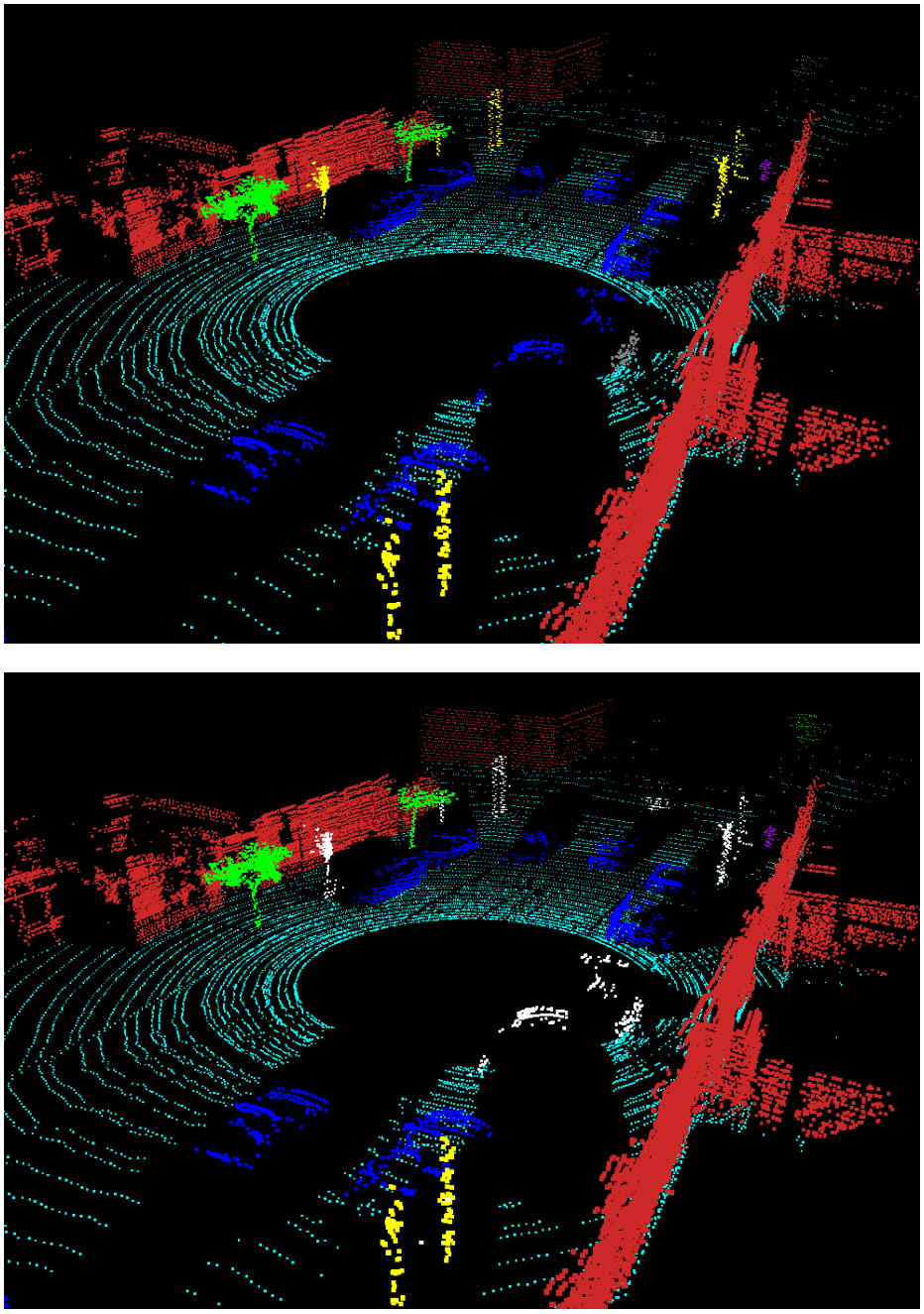

Figure 4.10: (top) Ground truth classification for part of a 3D laser scan. Colors indicate ground plane (cyan) and object types (green: tree, blue: car, yellow: street sign, purple: person, red: building, grey: other, white: not classified). (bottom) Classification achieved by our approach. As can be seen, most of the objects are classified correctly. The street signs in the back and the car near the center are not labeled since they are not close enough to any exemplar.

eliminates its rotation-invariance. Rotation-invariance makes sense in the context of shape retrieval if the query object can be presented in any orientation. Real-world data, however, often appears in very constrained set of orientations and so it can be a very useful cue.

Recently, several robotics research groups have also developed techniques for classification tasks based on visual and laser range information [141, 4, 36, 135, 116]. In robotics, Saxena and colleagues [118] used synthetically generated images of objects to learn grasp points for manipulation. Their system learned good grasp points solely based on synthetic training data. Newman's group has also done classification in maps constructed using laser range and camera data [107]. Their work has thus far been concerned with terrain classification as opposed to the classification and localization of specific objects. [104] is an earlier work on 3D point cloud classification using Gentle AdaBoost. Although they demonstrated good results detecting office chairs in several indoor scenes, our comparison against a LogitBoost classifier suggests that an off-the-shelf boosting algorithm will not perform well on our data set, which contains a lot of variability in objects, orientations, and occlusions.

The problem of combining data from different sources is a major area of research in natural language processing and computer vision [67, 53, 6, 111, 26, 114, 133]. Here, text sources from very different topic domains are often combined to help classification. Several relevant techniques have been developed for transfer learning [24, 30] and, more recently, domain adaptation [26, 69, 33, 32]. In this chapter, we have applied one of the state-of-the-art domain adaptation techniques from the NLP community [33] to the problem of 3D point cloud classification and showed that it can significantly improve performance. In addition, we also presented an alternative domain adaptation technique specific to per-exemplar distance function learning and showed that it attains slightly better performance.

4.7 Summary

The computer vision community has recently shown that using large sets of weakly labeled image data can help tremendously to deal with the complexity of the visual world. When trying to leverage large data sets to help classification tasks in robotics, one main obstacle is that data collected by a mobile robot typically has very different characteristics from data available on the World Wide Web, for example. For instance, our experiments show that simply adding Trimble 3D Warehouse objects to manually labeled 3D point clouds without treating them differently can *decrease* the accuracy of the resulting classifier.

In this chapter we presented a domain adaptation approach that overcomes this problem. Our technique is based on an exemplar learning approach developed in the context of image-based classification [95]. We showed how this approach can be applied to 3D point cloud data and extended it to the domain adaptation setting. For each scene, we generate a "soup of segments" in order to generate multiple possible segmentations of the point cloud. The experimental results show that our domain adaptation improves the classification accuracy of the original exemplar-based approach and clearly outperforms boosting and multi-class SVM classifiers trained on the same data. The approach was additionally evaluated on a data set of indoor objects and achieved very promising results, demonstrating the effectiveness of the approach in a wide range of problem domains.

A key strength of the exemplar-based distance learning approach presented here is that the objective function explicitly learns the weights for combining multiple types of features. However, learning a distance function for every exemplar can be very expensive for a large dataset. In the RGB-D Object Dataset, exemplars are grouped into object instances which have very similar appearance, since they are just the same object viewed from different angles. In such a scenario, it makes sense to learn only a single distance function for each object. In the next chapter, we present a distance

learning approach that learns per-object rather than per-exemplar distances.

Chapter 5

DISTANCE LEARNING FOR RGB-D OBJECT RECOGNITION

5.1 Background

RGB-D cameras output two different streams of data: RGB images and depth images. As seen in Chapter 3, extracting features from both streams and combining them yields better performance than either alone. However, combining features of different dimensionality and range of values is non-trivial. Methods for doing so include feature scaling [18, 46], multiple kernel learning [7, 128], and distance learning (e.g. [144, 140]), in particular local distance learning [121]. Local distance learning has been extensively studied and demonstrated for object recognition, both for color images [49, 50, 95]. In Chapter 4, we presented an exemplar-based distance learning approach for 3D point clouds. A key property of these approaches is that they can model complex decision boundaries by combining elementary distances. Local distance learning, however, is not without issues. For our problem setting, there are two main limitations to overcome: (1) existing formulations of local distance learning do not capture the relations between object categories and specific instances under them; (2) they provide no means for selecting representative views, or example images, of instances and thus become very inefficient if a large number of views are collected for each object.

In this chapter we propose an approach for **I**nstance **D**istance **L**earning (IDL): instead of learning per-view distances, we define and optimize a *per-instance distance* that combines all views of an object instance (see Fig. 5.1). By learning a distance function jointly for all views of a particular object, our approach significantly outperforms view-based distance learning for RGB, Depth, and RGB+Depth recognition.

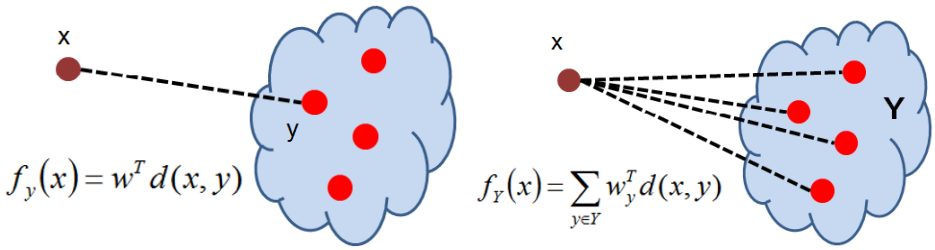

Figure 5.1: Two distance learning approaches. (Left) Local distance learning uses a view-to-view distance, typically followed by a k-nearest neighbor rule. (Right) The proposed instance distance learning, where we use the weighted average distance from a view x to an object instance $\mathbf{Y}$ which consists of a set of views of the same object.

This result can also be motivated as subclass classification [138, 37]. In addition, joint instance distance learning naturally leads to a sparse solution using Group-Lasso regularization, where a sparse set of views of each instance is selected from a large pool of views. Thus, IDL provides a data-driven way to select informative training examples for each object and significantly sparsify the data set, discarding redundant views and speeding up classification. We show that IDL achieves sparse solutions without any decrease in performance.

5.2 Learning Instance Distances

In this section, we describe how to learn instance distance functions for classification tasks in the context of image classification. In image classification, we are given a set of objects Y. The goal is to learn a classifier to predict category and instance labels of images, or views, outside the training set. One of the simplest methods to do this is to find nearest neighbors of the test view and make a prediction based on the labels of these nearest neighbors. In this section, we show how to improve this approach by learning an instance distance function. We start by considering a simple classification rule, the *nearest instance classifier*, which labels incoming test images x using the

label of the nearest instance (an extension to k-nearest instances is straightforward):

$$c_x = \underset{i,j}{\operatorname{argmin}} \frac{1}{|Y_{ij}|} \sum_{y \in Y_{ij}} d(x, y) \tag{5.1}$$

Here, Y_{ij} denotes the set of views taken of the j-th instance of the i-th category. As can be seen, c_x is the object that appears most similar to the test image, averaged over its views. $d(x, y)$ can be any distance function between views x and y. In this work, we use the l_2 distance $d(x, y) = \|x - y\|$. The nearest instance classifier given in (5.1) can be used for both category and instance recognition: The index i provides the category and the index j gives the corresponding instance. Unfortunately, the nearest instance classifier can often perform poorly in practice due to the difficulties of finding a good distance measure.

We now consider a significantly more powerful variant by learning an instance distance function for recognition. In many problems there are multiple features available and the best performance is obtained by using all available information. To do so, we replace the scalar distance $d(x, y)$ between two views x and y by a vector $\mathbf{d}(x, y)$ of separate l_2 feature distances. The corresponding instance distance function between example x and the j-th instance of i-th category Y_{ij} can then be written as

$$f_{ij}(x) = \frac{1}{|Y_{ij}|} \sum_{y \in Y_{ij}} \mathbf{w}_y^\top \mathbf{d}(x, y) + b_{ij} \tag{5.2}$$

where $\mathbf{W}$ is a set of weight vectors $\mathbf{w}_y$ for all $y \in Y_{ij}$. Unlike the *nearest instance classifier*, this significantly more expressive distance function allows the classifier to assign different weights to each feature and for each view, enabling it to adapt to the data. Note that we have added a bias term, b_{ij}, to the instance distance function to allow negative values. The weight vector $\mathbf{w}_y$ is D-dimensional, where D is the number of different features extracted for each view. Note also that each example view has a different weight vector. Due to this, the functions do not define a true distance metric, as they are asymmetric. This is advantageous since different examples

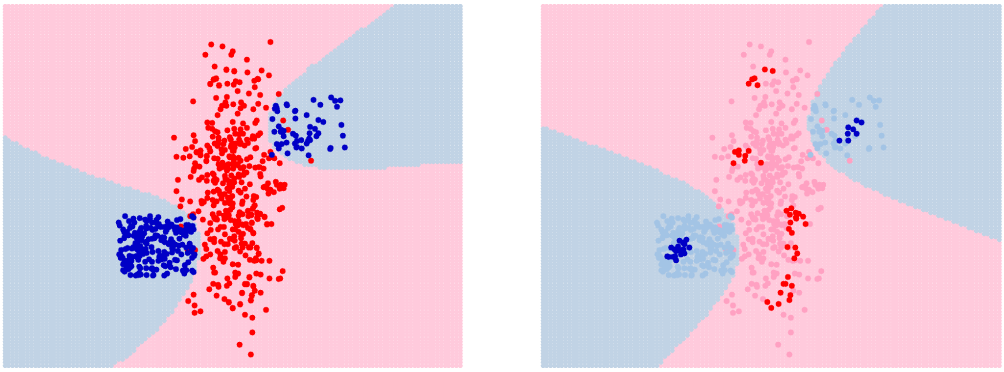

Figure 5.2: Decision boundaries found by two instance distance classifiers on a two-dimensional dataset. (Left) instance distance learning with l_2 regularization. (Right) instance distance learning with data sparsification, which retains only 8% of data (stronger colors) and still has a similar decision boundary.

may have different sets of features that are better for distinguishing them from other examples, or views.

When learning the weight vector for an instance, it is necessary to distinguish between category and instance classification. For *instance recognition*, the weight $\mathbf{W}_{ij}$ defining the distance function for the j-th instance in category i can be learned using the following l_2 regularized loss function:

$$\sum_{x \in Y_{ij}} L(-f_{ij}(x)) + \sum_{x \in Y \setminus Y_{ij}} L(f_{ij}(x)) + \lambda \mathbf{W}_{ij}^{\top} \mathbf{W}_{ij} \tag{5.3}$$

where we have chosen $L(z) = \max(0, 1 - z)^2$, the squared hinge loss. The first term penalizes misclassification of views $x \in Y_{ij}$ that belong to the same instance. The second term similarly penalizes misclassification of negative examples, or views, by incurring a loss when their distance is small. Note that the negative examples also include views of different instances that belong to the same category i. The final term is a standard l_2 regularizer, biasing the system to learn smaller weight vectors. This objective function is convex and can be optimized using standard optimization algo-

rithms. We use an L-BFGS solver in our experiments. Given a test image x, we assign to it the category or instance label of the nearest object using $c_x = \text{argmin}_{i,j} f_{ij}(x)$.

For *category recognition*, we learn the instance distance by minimizing the following l_2 regularized loss:

$$\sum_{x \in Y_i} L(-f_{ij}(x)) + \sum_{x \in Y \setminus Y_i} L(f_{ij}(x)) + \lambda \mathbf{W}_{ij}^\top \mathbf{W}_{ij} \tag{5.4}$$

where $Y_i = \bigcup_{s=1}^{N_i} Y_{is}$ and N_i is the number of instances in the $i-th$ category. The key difference between the instance recognition and the category recognition loss is that in the former, only the views of the same instance are positive examples, whereas in the latter the views of *all* instances in the same category become positive examples.

Fig. 5.2 (left) shows the decision boundary obtained with instance distance learning on a two-dimensional dataset. The dataset contains two classes: red and blue. There are two separate instances in the blue class and they lie on opposite sides of the single red class instance. Instance distance learning is able to find a very good decision boundary separating the two classes.

5.3 Example Selection via Group-Lasso

An important property of the instance distance we defined in Section 5.2 is that it allows for data sparsification. This is achieved by replacing l_2 regularization in (5.3) with Group-Lasso [147, 99], resulting in the following objective function:

$$\sum_{x \in Y_{ij}} L(-f_{ij}(x)) + \sum_{x \in Y \setminus Y_{ij}} L(f_{ij}(x)) + \lambda \sum_{y \in Y_{ij}} \sqrt{\mathbf{w}_y^\top \mathbf{w}_y} \tag{5.5}$$

Here, the first two terms optimize over individual components of the instance weight vector, and the third, Group-Lasso, term drives the weight vectors of individual views toward zero. Group-Lasso achieves this by grouping the weight components of individual views in the penalty term. In contrast to previous work that make use of the Group-Lasso for encouraging feature sparsity, here we use it to encourage data sparsity. In other words, optimizing this objective function yields a supervised method

for choosing a subset of representative examples, or views. If the Group-Lasso drives an entire weight vector $\mathbf{w}_y$ to $\mathbf{0}$, the corresponding example no longer affects the decision boundary and has effectively been removed by the optimization. The degree of sparsity can be tuned by varying the λ parameter. Intuitively, data sparsity is often possible because many examples may lie well within the decision region or are densely packed together. Removing such examples would reduce the magnitude of the regularization term while having little or no effect on the loss terms. Each data point is only one of many that contribute to the instance distance and redundant examples would not significantly influence the decision boundary.

The advantage of data sparsification using the proposed objective function is twofold. As explained above, the proposed technique can remove redundant and uninformative examples. Secondly, removing examples from consideration at test time results in computational cost savings which counteracts the data-size-dependent time complexity of nearest neighbor techniques.

For category level, the group lasso based instance distance learning uses the following objective function

$$\sum_{x \in Y_i} L(-f_{ij}(x)) + \sum_{x \in Y \setminus Y_i} L(f_{ij}(x)) + \lambda \sum_{y \in Y_{ij}} \sqrt{\mathbf{w}_y^\top \mathbf{w}_y} \tag{5.6}$$

Fig. 5.2 (right) shows a data sparsification example using instance distance learning with Group-Lasso. In this two-dimensional dataset, the technique is able to throw away 92% of the data and still obtain decision boundaries that closely match the one learned without data sparsification.

5.4 Experiments

We apply the proposed Instance Distance Learning (IDL) to two related object recognition tasks: category recognition and instance recognition. In category recognition, the system is trained on several objects belonging to each category and the task is to

classify a never-before-seen object into one of the categories. In the instance recognition task, the system is presented with multiple views of each object, and the task is to classify never-before-seen views of these same objects. The experimental results in this section demonstrate that our technique obtains good performance on both recognition tasks, particularly when taking full advantage of both shape and visual information available from the sensor. The technique is able to not only automatically sparsify training data, but it also exceeds the performance of several alternative approaches and baselines, even after sparsification. We also apply the instance distance learning technique to object detection and show that it is able to detect objects in a cluttered scene.

5.4.1 Experimental Setup

We evaluate our technique on the RGB-D Object Dataset (see Chapter 2) consisting of cropped and segmented images of distinct objects spun around on a turntable. We use the same subsampled dataset as in the object recognition experiments in Section 3.2 and extract the same features including EMK spin images, EMK SIFT features, and texton histograms (see Section 3.2).

For EMK spin image features, we once again divide an axis-aligned bounding cube around each view into a $3 \times 3 \times 3$ grid and compute a 1000-dimensional EMK feature in each of the 27 cells separately. We also once again perform principal component analysis (PCA) on the EMK features in each cell and take the first 100 components. Finally, we once again include as shape features the width, depth and height of a 3D bounding box around the view. Instead of concatenating all of these into a 2703-dimensional descriptor as was done in Section 3.2, we treat each of the 27 EMK spin image cells and each of the bounding cube width, depth, and height as separate features. This gives us a total of 30 shape descriptors. For visual features, we once again divide the image into a 2×2 grid and compute EMK SIFT features separately in each cell. We also once again extract texton histograms [88], a color histogram and

the mean and standard deviation of each color channel. There are a total of 13 visual descriptors. Since there are a total of 43 feature descriptors, in these experiments IDL learns distance functions where the weight vector of each view, w_y, is 43-dimensional.

5.4.2 Performance Comparisons

Given the above set of features, we evaluate the category and instance recognition performance of the proposed instance distance learning technique and compare it to a number of alternative state-of-the-art classifiers:

- IDL: Our proposed instance distance learning algorithm with l_2 regularization.
- EB LOCAL: The exemplar-based local distance function learning of Chapter 4 without domain adaptation, i.e. as originally proposed by Malisiewicz et al. [95].
- SVM: linear support vector machine
- RF: random forest classifier [21]

We follow the experimental setup in Section 3.2 to allow for direct comparisons. For category recognition, we randomly leave one object out from each category for testing and train the classifier on all views of the remaining objects. For instance recognition, we divide each video into 3 consecutive sequences of equal length and for each object instance. There are 3 heights (videos) for each object, so this gives 9 video sequences for each instance. We randomly select 7 of these for training and test on the remaining 2.

To verify that our technique is indeed able to take advantage of both shape and visual information available from the RGB-D camera, we evaluated the performance of all the techniques using only shape-based features, only visual-based feature, and using both shape and visual features. Table 5.1 shows the overall classification performance of the different algorithms on both category-level and instance-level recog-

Technique	Classification Accuracy					
	Category			Instance		
	Shape	Vision	All	Shape	Vision	All
EBLocal	58.9 ± 2.1	70.1 ± 3.4	78.4 ± 2.8	41.2 ± 0.6	81.2 ± 0.6	84.5 ± 0.5
LinSVM	53.1 ± 1.7	74.3 ± 3.3	81.9 ± 2.8	32.4 ± 0.5	$\mathbf{90.9 \pm 0.5}$	90.2 ± 0.6
RF	66.8 ± 2.5	74.7 ± 3.6	79.6 ± 4.0	52.7 ± 1.0	90.1 ± 0.8	90.5 ± 0.4
IDL	$\mathbf{70.2 \pm 2.0}$	$\mathbf{78.6 \pm 3.1}$	$\mathbf{85.4 \pm 3.2}$	$\mathbf{54.8 \pm 0.6}$	89.8 ± 0.2	$\mathbf{91.3 \pm 0.3}$

Table 5.1: Classification performance of various techniques on the RGB-D data set. EBLocal is exemplar-based local distance learning, LinSVM is linear SVM, RF is Random Forest, and IDL is the proposed instance distance learning.

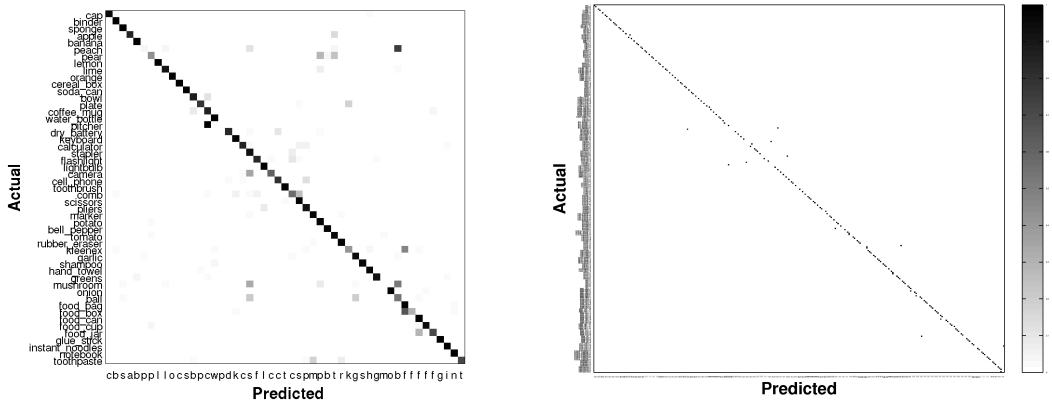

Figure 5.3: Confusion matrices (row-normalized) for *sparse instance distance learning* on (left) category recognition and (right) instance recognition.

nition. As can be seen from the results, our technique substantially improves upon the performance of a competitive exemplar-based local distance method and other state-of-the-art classification techniques in most cases or otherwise gets comparable performance.

Overall, visual features are more useful than shape features for both category level and instance level recognition. However, shape features are relatively more useful in category recognition, while visual features are relatively more effective in instance recognition. This is exactly what we should expect, since a particular object instance has a fairly constant visual appearance across views, while objects in the same category can have different texture and color. On the other hand, shape tends to be stable across a category in many cases, thereby making instance recognition via shape more difficult. The fact that combining both shape and visual features enables our technique to perform better on both tasks demonstrates that our technique can take advantage of both shape and visual features. This same trend was also observed in Section 3.2.

5.4.3 Data Sparsification Results

Fig. 5.4 (left) shows the classification accuracy of two data sparsification techniques at varying levels of data sparsity: 1) running instance distance learning technique on a uniform random downsampling of the training data and 2) our sparse instance distance learning (IDL SPARSE). The curve for IDL SPARSE is generated by varying the regularization tradeoff parameter, λ. The plot shows that IDL SPARSE is able to sparsify the data considerably (up to a factor of $\frac{1}{5}$) without causing any significant loss in accuracy. Note that IDL SPARSE does not necessarily converge to the same accuracy as IDL because the techniques use different regularization. The two techniques are only identical when the regularization tradeoff parameter is set to 0, but this would lead to overfitting.

Although uniform random downsampling is a naïve form of sparsification, it actually works very well on our dataset, since uniform sampling of video frames gives good coverage of object views. Nevertheless, the plot clearly shows that IDL SPARSE obtains higher classification accuracy than random downsampling across all levels of data sparsity. Fig. 5.4 (right) shows some example views retained for several objects.

Fig. 5.3 shows the confusion matrices between the 51 categories for category recognition (left) and the 300 instances for instance recognition (right). In the category recognition run, the sparse instance distance learning obtained an overall accuracy of 83% and retained 15% of the training data. In the instance recognition run, the technique obtained an overall accuracy of 89.7% and retained 19% of the training data.

5.4.4 3D Object Category Dataset

In addition to the novel RGB-D Object Dataset that we collected, we also evaluated instance distance learning (IDL) on a publicly available image-only dataset: the 3D object category dataset presented by Savarese et al. [117]. There are 8 object cat-

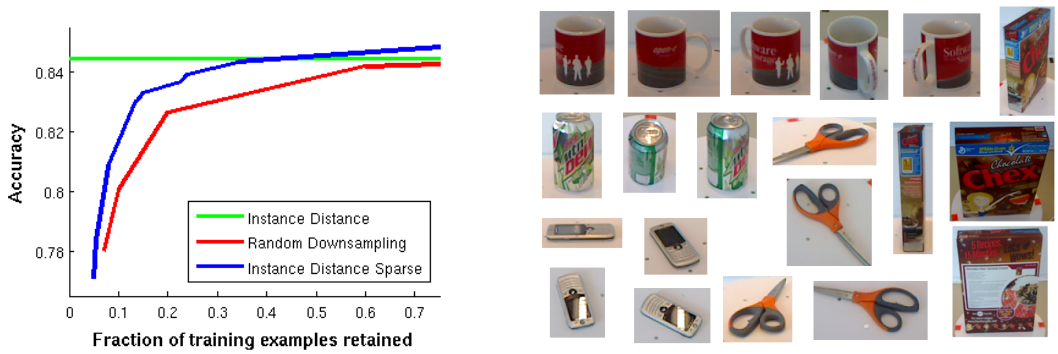

Figure 5.4: (Left) Number of examples retained versus classification accuracy of two example selection techniques: 1) random downsampling and 2) sparse instance distance learning. Accuracy of Instance distance learning is shown for comparison. (Right) Data selection with Group-Lasso: A small set of representative views that were chosen for several objects.

egories in the dataset: bike, shoe, car, iron, mouse, cellphone, stapler, and toaster. For each category, the dataset contains images of 10 individual object instances under 8 viewing angles, 3 heights and 3 scales for a total number of 7000 images that are all roughly 400×300 pixels. We evaluated IDL on category level recognition on this dataset using the same setup as [117]: we randomly select 7 instances per category for training and use the rest for testing. The furthest scale is not considered for testing. IDL obtains substantially higher accuracy (80.1%) than the results reported in [117] (75.7%).

5.4.5 Object Detection

In this section, we apply instance distance learning to the problem of object detection. In object detection, the system is given a fixed set of objects to search for and trains the appropriate detectors beforehand. At test time, the system is presented with a set of images and must identify all objects of interest that are present in the image

by specifying bounding boxes around them. Given the task of identifying a particular set of objects, an instance distance classifier is trained for each instance by using views in the particular instance as positive examples. The set of negative examples is constructed from views of other objects as well as a separate set of background images that do not contain any objects the system is tasked to find. At test time, the system is presented with a video sequence taken from a particular scene, e.g. a kitchen area or an office table. Similar to the approach presented in Section 3.3, the system runs a sliding window detector of a fixed size across each video frame. The window sliding is once again done over an image pyramid to search across 20 scales. As before, we perform non-maximum suppression to remove multiple overlapping detections. The difference is that instead of invoking a linear SVM classifier at each window, an instance distance classifier is used instead. The classifier returns a score, which we threshold to obtain bounding boxes. As in Section 3.3, we use HOG features extracted from the RGB image, HOG features extracted from the depth image, and normalized depth histogram features.

We evaluated the IDL classifier on the object detection task on a video sequence of an office environment from the RGB-D Scenes Dataset. Objects were placed on a table and the system was tasked with finding the soda can, coffee mug, and cap in the video sequence. The cereal box acts as a distractor object and sometimes occludes the objects of interest. Following the PASCAL VOC evaluation metric, a candidate detection is considered correct if the intersection of the predicted bounding box and the ground truth bounding box is more than half of their union. Only one of multiple successful detections for the same ground truth is considered correct and the rest are counted as false positives. Fig. 5.5 (left) shows the precision-recall curves of the individual object detectors as well as the overall precision-recall curve for all the objects. Each precision-recall curve is generated by ranking the resulting detections using scores returned by the classifier and thresholding on them. Each threshold gives a point along the curve. We run only the detector for the particular object to generate

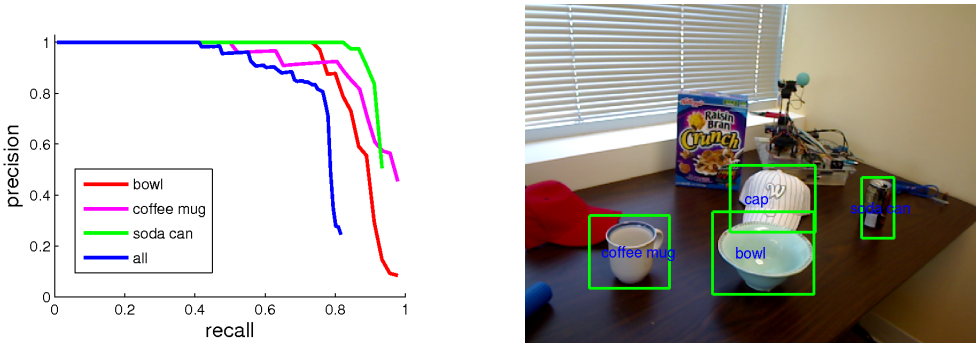

Figure 5.5: Object detection results on a video sequence. (Left) Precision-recall curves of individual bowl, coffee mug, and soda can detectors and aggregated detections. (Right) Example video frame with detection results.

the precision-recall curves for the individual objects. For the multiple-object curve, we run all three object detectors and pool all candidate detections across objects and generate a single precision-recall curve. The precision-recall curves show that IDL attains good performance on the object detection task. Even when searching for three different objects by running multiple detectors in the video sequence, there is only a slight drop in the precision and recall. Fig. 5.5 (right) shows an example multi-object detection. Here the system is able to correctly locate the three objects even though there are other objects and background clutter in the scene.

5.5 Summary

In this chapter we studied both object category and instance recognition using the RGB-D Object Dataset (see Chapter 2), a large RGB-D (color+depth) dataset of everyday objects. Our work is of interest both in terms of algorithm design and of the empirical validations on appearance and depth cues for recognition. Our key insight is that because a category consists of different objects, there is a natural division of a category into subclasses, and this motivates our use of the instance

distance. We show that by jointly learning the weights in this distance function, we outperform alternative state-of-the-art approaches. The proposed instance distance learning provides a distance measure for evaluating the similarity of a view to a known set of objects. This information can be used as input to other robotics tasks, such as grasping. An interesting direction for future work is to treat the training data as an object database where grasping information is stored for each object. When the robot encounters an object in the world, it can use the instance distance classifier to match the object to objects in the database to retrieve potential grasps.

The use of Group-Lasso allows us to find a compact representation of each object instance as a small set of views without compromising accuracy. With the ever increasing size of data sets available on the World Wide Web, *sparsification* of such data will become more important. While the current technique assumes an offline setting, the development of online Group-Lasso style sparsification is an interesting and promising direction for future work.

Finally, we showed that using both shape and visual features achieves higher performance than either set of cues alone for both category and instance recognition. This matches what we saw with existing techniques in Chapter 3, lending further support to the importance of combining RGB images and 3D point cloud data.

While category and instance recognition are both very important capabilities for intelligent systems, in many robotics applications where the system must interact with the environment, pose estimation is also necessary. For example, whether a mug handle is facing left or right can inform a robot planning to grasp the object, and whether a mug upside down confers information about whether it is currently being used. In the next chapter, we present a scalable algorithm that simultaneously estimates the category, instance, and pose of an object.

Chapter 6

JOINT RGB-D OBJECT RECOGNITION AND POSE ESTIMATION

6.1 Background

In the previous chapter, we presented an algorithm for object category and instance recognition. In practice, object recognition has multiple levels of semantics and multiple usage scenarios. When an autonomous robot encounters an object, we would like it to answer any or all of the following questions: *Is this a coffee mug or a plate?* (category recognition); *Is this Alice's coffee mug or Bob's coffee mug?* (instance recognition); *Am I looking at the mug with the handle facing left or right?* (pose recognition or approximate pose estimation). Although it is clear that category, instance, and pose recognition are closely connected and multiple facets of a single object perception problem, they have traditionally been studied in different contexts and solved using different techniques. The computer vision community has focused on category-level recognition and developed sophisticated features [93, 13] and matching techniques [86, 42]. There have been efforts to study the problem of discrete (or rough) pose estimation [117], and it has been recently shown that category recognition and discrete pose recognition can be solved together [56]. The robotics community has typically focused on instance recognition and pose estimation, with techniques using sparse feature matching for textured object labeling [96], or ICP alignment for generic 3D pose estimation [10, 76].

One other prevailing issue and urgent need of practical object recognition research is that of *scalability*. For practical systems, partly because of the need for real-time processing, recognition is traditionally demonstrated on only a handful of objects.

Recent studies have started to push the limit to dozens of object instances [96], but it is still in sharp contrast with the possibly thousands of objects one would encounter in everyday life. In computer vision, there have been ambitious efforts to move up to $100,000$ object classes [34], and the machine learning community has developed scalable tree-based approaches to solve such large-scale problems [9]. A two-stage approach has also been proposed to handle large-scale recognition by filtering the database into small sets [60]. There have been few such studies on scalable object recognition from a practical perspective. For example, in practice there is a strong need for online incremental learning, as an autonomous agent constantly explores and learns about novel objects.

In this chapter, we describe a tree-based approach that simultaneously solves the three object recognition problems: category, instance, and pose [81]. We evaluate the performance of this technique on the RGB-D Object Dataset that was introduced in Chapter 2.

6.2 Object-Pose Tree

In this section, we describe the Object-Pose Tree, which we first proposed in [81]. The Object-Pose Tree is an object recognition framework that simultaneously addresses category, instance, and pose recognition. These three levels of object recognition form a tree as naturally defined by the semantic structures: a category covers multiple instances, an instance covers multiple (discrete) "views", and each view is a collection of (continuous) object poses (Fig. 6.1). We show that all three recognition tasks can be performed robustly and efficiently by traversing the tree. The tree, or hierarchical, view of the three tasks enables us to jointly learn recognition models at all three levels.

Our joint recognition approach directly tackles the problem of scalability. We show that (1) our tree-based approach leads to efficient recognition without losing accuracy; (2) the tree structure allows efficient and joint training of recognition models at all

levels, much faster than standard 1-vs-All training; and (3) online learning, based on stochastic gradient descent, allows us to efficiently update an existing tree model and incorporate novel objects. By solving three recognition tasks together, we are able to perform near real-time recognition over thousands of object poses much more efficiently and accurately than nearest neighbor retrieval or 1-vs-All classifiers.

In the joint category, instance, and pose recognition task, we are given a set of training samples $\{x_i, y_i^C, y_i^I, y_i^V, y_i^P\}_{i=1}^N$ where inputs x_i are cropped and segmented images of objects, the outputs y_i^C,y_i^I, y_i^V, and y_i^P are respectively the categories, instances, views, and poses of the objects, and N is the number of training samples. The goal is to predict the category name, instance name and pose of unseen images. An intermediate view level between the instance and pose levels is added where each view contains a set of nearby poses (see Chapter 2 for more details). To estimate the category, instance, and pose of a given object jointly, we build an Object-Pose Tree (OP-Tree). The OP-Tree is a semantically structured tree of classifiers that consists of four levels: category, instance, view, and pose as illustrated in Fig. 6.1, where one classifier (semantic label) is associated with each tree node.

6.2.1 Joint Category, Instance & Pose Recognition

To learn an OP-Tree, we will minimize the empirical loss on a set of labeled training data. Before defining this loss, we need to specify the classification performed by the OP-Tree. At test-time, the OP-Tree recognizes an object by evaluating the tree of classifiers one level at a time. The category-level classifiers are first evaluated and the object is assigned the category label of the highest scoring classifier

$$F^C(x) = \underset{i}{\operatorname{argmax}}\, f_i^C(x) \;, \tag{6.1}$$

where $f_i^C(x)$ is the output of the binary classifier for category i.

The system then evaluates the set of instance classifiers belonging to the assigned category label and selects the instance label of the highest scoring classifier at the

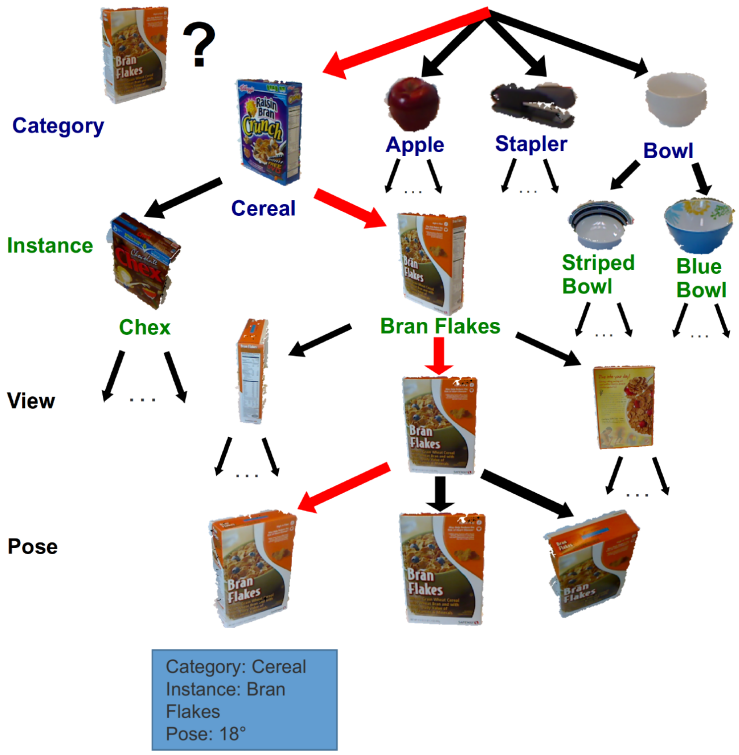

Figure 6.1: Category, instance, and pose recognition of a box of Bran Flakes cereal using the Object-Pose Tree. The system labels the test image by evaluating a set of classifiers arranged in a semantically structured tree, starting with the category-level at the top and traversing down the tree to the instance, view, and finally the pose level at the bottom. The system finds the most similar (but not identical) pose in the training set.

instance level. The system continues down through the view and pose levels to finally assign a pose label to the image. Instance, view, and pose classification are summarized by the following rule:

$$F^s(x) = \underset{i:(F^r(x),i)\in E}{\operatorname{argmax}} f_i^s(x), \tag{6.2}$$
$$(s,r) \in \{(I,C),(V,I),(P,V)\}$$

Here, the set of edges $E = (p_1, c_1), \ldots, (p_{|E|}, c_{|E|})$ contains ordered pairs of parent and child node indices, and $f_i^s(x)$ are the classifiers associated with the nodes i at level s. For example, the *pose* of an input x is predicted as

$$F^P(x) = \underset{i:(F^V(x),i)\in E}{\operatorname{argmax}} f_i^P(x), \tag{6.3}$$

where $F^V(x)$ is the predicted *view*, and the maximization is performed over all pose classifiers $f_i^P(x)$ associated with that specific view.

We are now ready to specify the empirical loss associated with a set of labeled inputs x_i. Note that since the view level is only added for efficiency reasons, the loss is specified only at the category, instance, and pose level:

$$\begin{aligned} R_{emp} &= \frac{1}{N}\sum_{i=1}^{N} \delta(F^C(x_i) \neq y_i^C) \\ &+ \frac{1}{N}\sum_{i=1}^{N} \max_{s\in\{C,I\}} \delta(F^s(x_i) \neq y_i^s) \\ &+ \frac{1}{N}\sum_{i=1}^{N} \max\{ \max_{s\in\{C,I\}} \delta(F^s(x_i) \neq y_i^s), \max_{s\in\{V,P\}} \Delta(F^s(x_i), y_i^s)\} \,. \end{aligned} \tag{6.4}$$

Here, $\delta(\cdot)$ is the indicator function, and $\Delta(\cdot)$ is a continuous loss function normalized into $[0,1]$. The first term measures the error for assigning an incorrect category label. The second and third terms, which account for the errors of instance and pose recognition, respectively, take the maximum over multiple levels of the tree because a mistake at any of the higher levels will also lead to an incorrect prediction. For example, if the tree makes a mistake at the category level for a particular object, it will also fail on both the instance and pose recognition task. In other words, R_{emp} is

a path loss that sums the number of nodes that do not match the true path for every training example.

Unlike category and instance recognition, which involve a discrete set of labels, for pose recognition it is natural to consider a continuous loss function. In our application, we represent views and poses as angles in $[0, 2\pi]$ and use the following continuous loss to measure the difference between two views or pose labels:

$$\Delta(y_i, y_j) = \frac{\min\{|y_i - y_j|, 2\pi - |y_i - y_j|\}}{\pi} \tag{6.5}$$

This function converts angle differences into a value in $[0, 1]$ to make the loss comparable with the 0-1 loss metric used at the category and instance levels.

Although the empirical loss described above is ideal, using it would yield a combinatorial optimization problem. We approximate this non-differentiable empirical loss with the hinge loss (used in support vector machines) to get a convex optimization problem. For efficiency we use linear classifiers $f_i^s(x) = (w_i^s)^\top x + b_i^s$. At the category level, this hinge loss exactly corresponds to the loss function for a joint multi-class support vector machine [29]:

$$\begin{aligned} R_{svm}^C &= \frac{1}{N}\sum_{i=1}^{N} \xi_i^C \\ s.t.\quad & f_{y_i^C}^C(x_i) - f_y^C(x_i) \geq 1 - \xi_i^C, \forall y \in L^C \\ & \xi_i^C \geq 0, i = 1, \ldots, N \end{aligned} \tag{6.6}$$

where L^C is the category label set.

We approximate the empirical loss over multiple levels of the tree using the approach proposed in [9], which takes the largest loss from the current level and the levels above (the ancestral levels). The hinge loss over the instance and pose levels

are given by

$$R^I_{svm} = \frac{1}{N}\sum_{i=1}^{N}\max\{\xi^C_i, \xi^I_i\} \tag{6.7}$$

$$\begin{aligned} s.t.\quad & f^C_{y^C_i}(x_i) - f^C_y(x_i) \geq 1 - \xi^C_i, \forall y \in L^C \\ & f^I_{y^I_i}(x_i) - f^I_y(x_i) \geq 1 - \xi^I_i, \forall y : (y^C_i, y) \in E \\ & \xi^C_i, \xi^I_i \geq 0, i = 1, \ldots, N \end{aligned}$$

and

$$R^P_{svm} = \frac{1}{N}\sum_{i=1}^{N}\max\{\xi^C_i, \xi^I_i, \xi^V_i, \xi^P_i\} \tag{6.8}$$

$$\begin{aligned} s.t.\quad & f^C_{y^C_i}(x_i) - f^C_y(x_i) \geq 1 - \xi^C_i, \forall y \in L^C \\ & f^I_{y^I_i}(x_i) - f^I_y(x_i) \geq 1 - \xi^I_i, \forall y : (y^C_i, y) \in E \\ & f^V_{y^V_i}(x_i) - f^V_y(x_i) \geq \Delta(y^V_i, y) - \xi^V_i, \forall y : (y^I_i, y) \in E \\ & f^P_{y^P_i}(x_i) - f^P_y(x_i) \geq \Delta(y^P_i, y) - \xi^P_i, \forall y : (y^V_i, y) \in E \\ & \xi^C_i, \xi^I_i, \xi^V_i, \xi^P_i \geq 0, i = 1, \ldots, N \end{aligned}$$

where $y : (y^C_i, y) \in E$ is the set of child nodes of the node y^C_i and similarly for y^I_i, y^V_i, and y^P_i.

We learn the weights of classifiers by optimizing the overall convex loss function, the summation of three above convex loss functions (Eq. 6.6), (Eq. 6.7), and (Eq. 6.8) and a convex term:

$$W^* = \underset{W}{\operatorname{argmin}}\{R_{svm} = \frac{\lambda}{2}W^\top W + R^C_{svm} + R^I_{svm} + R^P_{svm}\} \tag{6.9}$$

where the first term is the l_2-norm regularizer, λ is the trade-off parameter, and W is the concatenation of all weight vectors in the tree.

6.2.2 Object-Pose Tree Learning

We optimize eq. 6.9 using stochastic gradient descent (SGD), which is suitable to problems where the full gradients decompose as a sum of individual gradients of the

training samples. Unlike batch methods that estimate gradients using the full set of training samples, SGD approximates gradients using only a subset of training samples of fixed size and thus makes the cost of each iteration constant. This makes SGD a practical choice for large datasets. We use the SGD algorithm proposed by [122] which iteratively updates model parameters in two steps. The first step is

$$W_{t+\frac{1}{2}} = W_t + \frac{1}{\lambda t}\frac{\partial R^t_{svm}}{\partial W_t} \tag{6.10}$$

where R^t_{svm} is the hinge loss over a randomly chosen subset of training samples at step t, and the learning rate is set to be $\frac{1}{\lambda t}$. In the second step, we set W_t to be the projection of $W_{t+\frac{1}{2}}$ onto the set $\{B = W : \|W\| \leq \frac{1}{\sqrt{\lambda}}\}$

$$W_{t+1} = \min\{1, \frac{1}{\sqrt{\lambda}\|W_{t+\frac{1}{2}}\|}\}W_{t+\frac{1}{2}} \tag{6.11}$$

[122] present theoretical guarantees and empirical results to demonstrate that this stochastic gradient descent algorithm converges in reasonable time. As our experiments will demonstrate, our experience also confirms the usefulness of this optimization technique.

The ability to update the classification model in an incremental fashion is important for certain applications. Consider the scenario where there is an ever-growing database of objects shared by a number of robots for doing object recognition, where objects may be added to the database at any time. In such a scenario we want the model to be updated quickly so that robots can begin recognizing the newly added objects.

An important advantage of the proposed OP-Tree learning technique is that the tree can be quickly updated in an incremental fashion. Using stochastic gradient descent to optimize the tree parameters allows incremental updating of the tree as the system encounters new objects. This is accomplished simply by continuously running SGD as objects are added to the tree. The parameters of each new classifier is initialized using parameters from a randomly chosen sibling, while existing classifiers

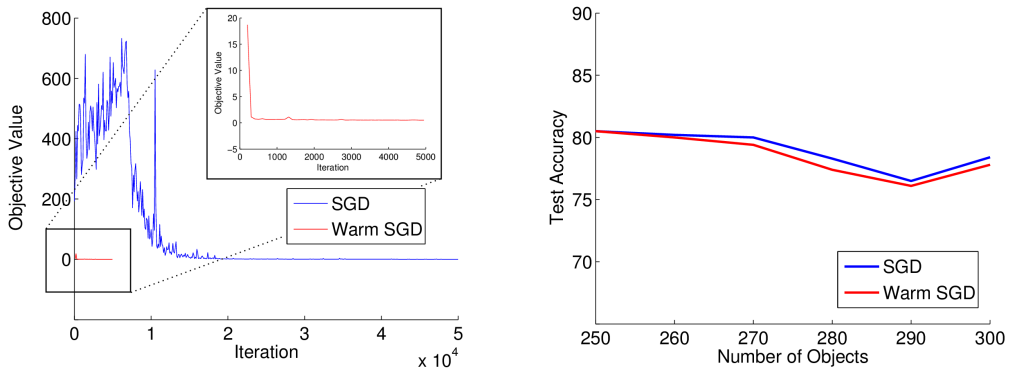

Figure 6.2: (Left) Optimization convergence comparison of stochastic gradient descent from scratch (SGD) and stochastic gradient descent starting with parameters learned for 290 objects and adding 10 more (Warm SGD). (Right) Instance recognition test accuracy of stochastic gradient descent from scratch (SGD) and stochastic gradient descent with pre-initialized parameters (Warm SGD), starting from 250 objects and adding 10 each round for 5 rounds.

are initialized with their current parameters. The incremental learning samples from both existing images as well as images of the new objects.

To verify this claim, we conducted an experiment to compare the efficiency of running SGD from scratch versus initializing parameters with previously learned weights. We first trained a tree from scratch using 290 objects from the RGB-D Object Dataset (see Chapter 2). We then consider two methods for updating the tree after adding ten objects to the classifier: 1) run SGD from scratch on the complete set of 300 objects (SGD), 2) run SGD with the parameters initialized as described in the previous paragraph (Warm SGD).

Fig. 6.2 (left) shows how the objective function value changes as the two optimization algorithms proceed. Although the objective value for SGD appears to have stabilized after 30,000 iterations, we found that test accuracy continued to improve until 50,000 iterations, using 200 random samples per iteration. In contrast, Warm

SGD only took 5000 iterations, also 200 samples per iteration, to converge and achieve the same test accuracy, yielding a 10x speed-up.

We also conducted an experiment where we start with an OP-Tree for 250 objects and then incrementally add 10 objects at a time for 5 rounds. Fig. 6.2 (right) shows that Warm SGD achieves comparable instance recognition test accuracies with SGD from scratch, even though only 5000 iterations were run on each round of Warm SGD while SGD from scratch had to be run for 50,000 iterations each round to reach comparable accuracy.

Finally, we verified that the Warm SGD optimization algorithm can learn an entire OP-Tree from scratch. We started an OP-Tree with no objects and added the entire dataset of 300 objects, 10 at a time, for 30 rounds. Due to time constraints this experiment was conducted using only the category and instance levels of the tree. We found that an OP-Tree learned this way, with 5000 iterations per round, achieved category and instance recognition accuracy within 1% of running SGD on the full dataset for 50,000 iterations.

6.3 Experiments

We compared OP-Trees to other approaches using the RGB-D Object Dataset, which contains images of 300 objects in 51 categories taken from multiple views with a Kinect-style 3D camera. 640×480 RGB and depth image pairs of each object are taken with the camera mounted at three different heights corresponding to angles of 30°, 45° and 60° with the horizon. Each image pair is annotated with ground truth category and instance labels, as well as pose angles in $[0, 2\pi]$. We divide the range of pose angles into 8 slices each covering 45° to form nodes in the view layer of the OP-Tree. Pose angles are aligned across all video sequences of instances from the same category. Following the *Leave-Sequence-Out* evaluation procedure in Section 3.2, we uniformly sample 48 images per video and use the 30° and 60° sequences as training data and the 45° sequence for evaluation. This yields around 28,000 RGB + Depth

image pairs for training and 14,000 for testing. Fig. 2.5 shows some example objects from the dataset.

6.3.1 Experimental Setup

To represent each RGB+Depth image pair, we first extract gradient and shape kernel descriptors [13] over a dense 8×8 grid from the RGB and depth images separately. We use Efficient Match Kernels (EMK) described in [17] to generate image-level features separately for each set of local kernel descriptors. This combination of kernel descriptors and EMK was demonstrated in [13] to outperform alternative state-of-the-art techniques on publicly available computer vision datasets like Caltech-101 and CIFAR-10. Our previous work [12] demonstrated that these features outperform the set of shape and visual features used in [80].

In our experiments we consider three broad approaches: 1) Nearest Neighbor classification, 2) One-versus-all linear classifier, and 3) the proposed Object-Pose Tree.

A nearest neighbor classifier can solve these 3 tasks jointly by labeling each test image using the category, instance, and pose labels of the k nearest training images in feature space. We tried different values of k and found that $k = 1$ gives the best results on the RGB-D Object Dataset.

Another approach is to train one-versus-all linear support vector machines separately for each recognition task. However, on our dataset a one-vs-all SVM for pose recognition involves training 28,000 binary classifiers, which did not finish in a reasonable amount of time. Hence, one must combine the use of one-vs-all classifiers with an alternative approach for pose estimation. One way is to use the nearest neighbor training image within the instance predicted using a one-vs-all classifier. Another way is to fit a ridge regression model for each instance and use the regression model for the predicted instance to predict the pose angle.

The OP-Tree technique presents a unified framework for jointly addressing category, instance, and pose recognition. The regularization trade-off parameter was

chosen using cross validation on the training set. To demonstrate the merits of this approach on all three recognition tasks, we also consider alternative methods of doing pose recognition given the category and instance output of the OP-Tree. As in one-versus-all, we consider using a nearest neighbor classifier or using ridge regression within the predicted instance for doing pose recognition.

6.3.2 Evaluation Criteria

All of the described approaches are evaluated on three recognition tasks: category, instance, and pose. We report standard category and instance recognition accuracies. As mentioned in Section 6.2, pose estimation is naturally a continuous problem, so in our evaluation we use one minus the continuous loss function (Eq. 6.5) as the pose accuracy.

6.3.3 Experimental Results

Table 6.1 presents a comparison of the various techniques described above. The results are grouped into three overall approaches: nearest neighbor (NN), one-versus-all (1vsA) and Object-Pose Tree (OPTree). Since the different approaches perform differently on instance recognition, the pose accuracies given correct instance predictions are only comparable within an approach but not across approaches. Test times are given for performing all three recognition tasks on a single test image. Feature extraction takes around one second regardless of the technique and is not included in the reported running times.

We report pose accuracies under three different scenarios. We chose to report both median and mean pose accuracies because the distribution across objects is skewed (see Fig. 6.4). For Avg & Med Pose, pose accuracies are computed on the entire test set, but images that were assigned an incorrect category or instance label have a pose accuracy of 0. Avg & Med Pose (C) are computed only on test images that were

Technique	Category	Instance	Avg Pose	Med Pose	Avg Pose (C)	Med Pose (C)	Avg Pose (I)	Med Pose (I)	Test Time (s)
NN	86.8	60.3	39.1	20.0	45.1	41.6	65.2	81.4	54.76
FLANN	84.6	55.9	38.2	10.9	42.5	33.4	64.3	78.4	0.21
1vsA	93.5	75.7	n/a	n/a	n/a	n/a	n/a	n/a	n/a
1vsA+NN			48.4	51.3	53.2	61.3	63.9	77.7	1.99
1vsA+RR			44.0	47.7	48.3	52.9	58.0	61.2	1.65
Indep Tree	92.0	77.4	50.4	59.3	54.8	65.6	65.0	75.2	0.33
OPTree	**94.3**	**78.4**	**53.5**	**65.2**	**56.8**	**71.4**	**68.3**	**83.2**	**0.33**
OPTree+NN			50.3	55.4	53.3	61.7	64.2	78.4	0.53
OPTree+RR			45.5	49.6	48.2	52.9	58.0	61.2	0.30

Table 6.1: Category, instance, and pose recognition accuracy and running time comparison of several techniques. *NN* is exact nearest neighbor classification, *FLANN* is an approximate nearest neighbor classification, *1vsA* is one-vs-all linear SVM, *1vsA+NN* is one-vs-all linear SVM for category and instance recognition, followed by nearest neighbor for pose estimation, *1vsA+RR* is one-vs-all linear SVM for category and instance recognition, followed by ridge regression for pose estimation, *Indep Tree* is a tree of classifiers where each level is trained as an independent linear SVM. *OPTree* is the Object-Pose Tree technique described in Section 6.2, *OPTree+NN* is an Object Tree for category and instance recognition, followed by nearest neighbor for pose estimation, *OPTree+RR* is an Object Tree for category and instance recognition, followed by ridge regression for pose estimation. *1vsA* for pose recognition cannot be trained in a reasonable amount of time and is omitted.

assigned the correct category by the system. Avg & Med Pose (I), are computed only on test images that were assigned the correct instance by the system.

Compared to alternative approaches, exact nearest neighbor classification (NN) is extremely slow and gives poor results. Methods for speeding up nearest neighbors include using a kd-tree and doing approximate nearest neighbors (FLANN) [100]. Due to the high dimensionality of our features, using a kd-tree for computing exact nearest neighbors is actually slower than exhaustive search. FLANN with automatically tuned parameters for 95% NN search accuracy gives running time comparable to one-versus-all and OP-Tree approaches, but leads to even lower recognition performance.

One-versus-all approaches (1vsA) improve over the accuracy and running time of nearest neighbors significantly. However, one-versus-all cannot be used for pose recognition because training a classifier for each image is not scalable. At test-time the system must also evaluate all classifiers, which scales linearly with the number of objects and poses. This means pose recognition must be addressed using a separate technique like nearest neighbors (1vsA+NN) or ridge regression (1vsA+RR).

Our Object-Pose Tree (OPTree) exceeds the accuracy and running time performance of nearest neighbor classification, one-versus-all classifiers, as well as an identically structured tree of classifiers where each set of nodes sharing a common parent is level of the tree is independently trained as a multi-class linear SVM (Indep Tree). The fact that OPTrees jointly address all three tasks enables the formulation of an objective function that takes advantage of the structure of the problem by using a continuous loss function at the view and pose levels.

Fig. 6.3 shows recognition results on two images using the OPTree. The image shown for the top 5 matching instances of each test image is the best matching pose from that instance according to the OPTree, by traversing the tree down to the pose layer regardless of whether the instance is the right one. These images, as well as the pose accuracies given correct category and correct instance reported in Fig. 6.4, show that the OPTree can give good pose estimates even when the instance classification

Figure 6.3: Recognition results from the Object-Pose Tree for two objects: *Red Mug* (top left), and *Ultrabrite Toothpaste* (bottom left). (Top) From left to right, the top five objects with the highest classifier response at the instance level and at the pose level for *Red Mug*. (Bottom) Instance and pose classifier responses for *Ultrabrite toothpaste*.

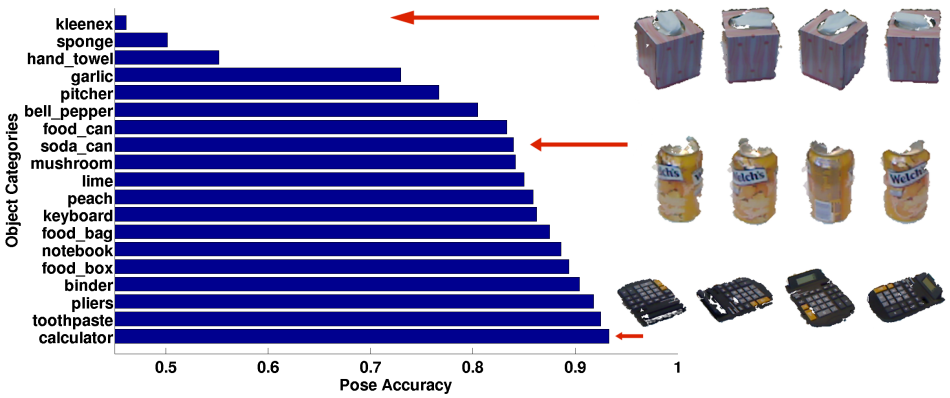

Figure 6.4: Median pose recognition accuracies from an Object-Pose Tree, given correct instance prediction, for a uniformly sampled subset of object categories. Object categories are sorted in increasing accuracies. Different views of a tissue box, a soda can, and a calculator illustrate the varying difficulty of pose estimation for objects in the RGB-D Object Dataset.

is wrong.

The experiments also demonstrate that nearest neighbor and ridge regression within the predicted instance (OPTree+NN, OPTree+RR) are both inferior to the OP-Tree in accuracy and running time. Nearest neighbor pose estimation can attain good accuracy, but its running time scales linearly with the number of training images of each object, while the OP-Tree scales logarithmically. Linear ridge regression offers constant-time performance, but gives much worse results. While the use of kernel ridge regression may improve accuracy, like nearest neighbors its computational cost also scales linearly with the number of poses.

Fig. 6.4 shows the median pose recognition accuracies from an OP-Tree, given correct instance predictions, for a uniformly sampled subset of object categories. The diversity of shape and appearance of objects in the RGB-D Object Dataset means that the difficulty of pose recognition can vary significantly, ranging from textureless and

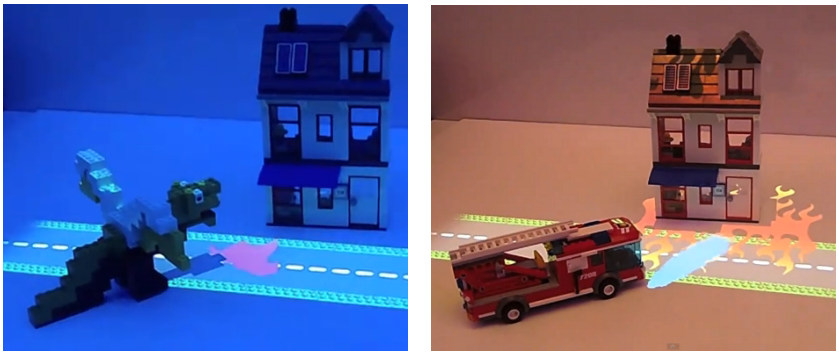

Figure 6.5: An interactive LEGO playing scenario in which an Object-Pose Tree is used to recognize several LEGO objects and estimate their pose. Recognizing the front of the house enables projection of a street. Detecting which direction the dragon is facing enables projection of a fire breathing animation (left), while detecting the fire truck enables projection of a fire extinguishing animation (right).

symmetric objects like balls and bowls where it is extremely difficult to distinguish the different poses, to calculators and flashlights where it is easy to tell based on appearance and shape. The OP-Tree achieves $> 83\%$ accuracy on 37 of 51 categories and 185 of the 300 objects. This level of accuracy corresponds to a pose estimation error of $< 30°$, obtained by inverting the continuous loss function (Eq. 6.5). A small subset of categories with symmetry and/or little texture like kleenex, sponge and hand towel has large pose accuracies of 90° or 180°.

6.3.4 Object-Aware Situated Interactive System

In collaboration with colleagues at the University of Washington and Intel Labs Seattle, we experimented with using our OP-Tree as the core object recognition algorithm in OASIS, an Object-Aware Situated Interactive System. [148] introduced an interactive LEGO toy playing scenario (LEGO OASIS) where objects placed on a table can be augmented with projected animations and interact with other physical and virtual

objects. Objects are associated with different animations and interact with each other in different ways depending on both their identities and poses. The OP-Tree plays an important role in LEGO OASIS by providing real-time object recognition and pose estimation (100ms per image), which enables responsive object-centric animations and interactions.

In LEGO OASIS, an RGB-D camera and a projector are both mounted above a table and pointed downwards. Since the table is at a fixed position relative to the camera, an object placed on the table can be easily detected and segmented via depth thresholding. The cropped and segmented image is presented to the proposed OP-Tree for determining the object's identity and pose (one of 12 discrete clock directions). Using this result, the system projects the appropriate animations for the identified object. For example, by recognizing a LEGO dragon and the direction at which its head is pointed, the system can project a fire breathing animation. Multiple objects can be recognized in the same play area, allowing interactions such as a dragon setting fire to a house, and a fire truck subsequently putting out the fire (Fig. 6.5). Four different objects were included in LEGO OASIS: a dragon, a house, a fire truck, and a train. The OP-tree, trained using around 200 images per object taken from various poses, is able to determine the identity and pose of these four objects at $> 95\%$ accuracy. The system was deployed during the 2011 Consumer Electronics Show (CES), where it was seen by more than 10,000 attendees (see http://www.cs.washington.edu/rgbd-dataset/demos.html).

6.4 Summary

In this chapter, we introduced Object-Pose Trees, a scalable approach for joint object category, instance, and pose recognition. In applications where it is necessary to solve large object recognition problems involving many categories and object instances at multiple levels of granularity, the Object-Pose Tree provides a computationally efficient framework. We provide extensive experimental results on the RGB-D Object

Dataset (see Chapter 2), which covers 300 object instances captured in both color and depth using RGB-D cameras [75]. Our tree-based system is able to search the entire object database within 0.33 seconds to identify an object's category, instance, and pose. The Object-Pose Tree trains and utilizes $28,000$ classifiers at the leaf level, which are too many to train via standard 1-vs-All classification. Online stochastic gradient descent for parameter optimization allows us to achieve a 10x speed-up when adding new objects to the database. We achieve $< 30^\circ$ pose estimation error on a wide range of household objects, often exhibiting symmetry and lacking distinctive shape or texture.

Thus far in this thesis, we have used features that have been specifically designed for a particular sensor modality and for capturing a specific set of attributes (e.g. color, gradient, shape, and surface normal orientation). For example, SIFT and kernel descriptor features are designed to capture image gradient information, while spin images are designed to capture surface normal and shape information. While such hand-designed features can be very effective, they are difficult to apply to different sensing modalities. Also, a new feature must be designed for each set of attributes that we wish to capture. In the next chapter, we introduce an unsupervised feature learning technique that can learn features from RGB-D data. Unlike existing feature learning techniques that learn features over the 2D structure of images, our approach learns features over 3D point clouds constructed from RGB-D videos. The approach can be applied to different attributes without modification, yielding features that are sensitive to different characteristics in the data.

Chapter 7

UNSUPERVISED FEATURE LEARNING FOR 3D POINT CLOUDS

7.1 Background

The main objective of research on feature representations is designing features that incorporate cues that are useful for the task at hand. Raw sensor data is often high-dimensional and noisy, and thus cannot be used directly. Over the years, the research community has designed features that use cues such as image gradients([93, 31]), color [52], shape [8], and surface normal orientation [70, 113]. In many of these papers, the researcher designs a specific algorithm for processing each cue, and perhaps combines several of them, controlling the algorithm by tuning a set of parameters. These approaches have been wildly successful in many applications including object recognition and detection in web images [146, 42], pedestrian detection [35], and object recognition and scene labeling in 3D point clouds [85, 101, 2]. As we showed in Section 3.2 and Section 3.3, combining such hand-designed features can lead to good object recognition and detection performance on RGB-D data as well.

Recently, researchers have increasingly studied unsupervised learning of feature representations directly from sensor data. While all feature representations are implicitly limited by the design choices and trade-offs made by their creators, unsupervised feature learning algorithms are more data-driven than hand-designed features. Instead of having a domain expert encode his or her knowledge into the feature representation through manually designing it, unsupervised feature learning approaches use a large dataset of sensor data to learn the feature representation using modern machine learning algorithms. This can lead to domain-independent feature learning

frameworks that can capture statistical patterns in data from any sensor modality (e.g. RGB images, depth images, 3D point clouds, speech). These more flexible representations have translated into better performance on a number of object recognition and scene labeling tasks: sparse coding and convolutional neural networks have shown promising results on image classification [87, 78, 16], object detection [110], and scene labeling [127].

One particular unsupervised learning algorithm is Hierarchical Matching Pursuit (HMP). HMP learns multiple levels of feature representations in a bottom-up fashion and has been applied to RGB-D images. It achieves state-of-the-art performance on object recognition [15] and attribute learning [131].

In this chapter, we introduce HMP3D, a novel hierarchical sparse coding technique for learning features over 3D point cloud data. HMP3D adapts the hierarchical matching pursuit (HMP) algorithm [15], a state-of-the-art sparse coding technique for image data, for use in learning features for 3D point clouds. HMP was originally proposed for learning features on grayscale images [14]. It has also been extended to learning features on RGB images, depth images, and surface normal images [15]. A commonality among these existing works is that HMP is used to learn features and perform spatial pooling over the 2D grid structure of images. As such, none of these methods use the 3D structure of objects and scenes available from RGB-D (Kinect-like) cameras. In contrast, HMP3D is a fully 3D feature learning approach that learns features and performs spatial pooling in 3D. We will show in Chapter 8 that this is beneficial for scene labeling tasks.

7.2 Feature Learning for 3D Point Cloud Data

Given a 3D point cloud of an object, we first discretize the point cloud to create a voxel representation. Each voxel is described by a set of attributes computed from the 3D points falling inside the voxel, such as RGB value and surface normal vector. In the context of scene labeling, we wish to learn a feature representation over a dataset

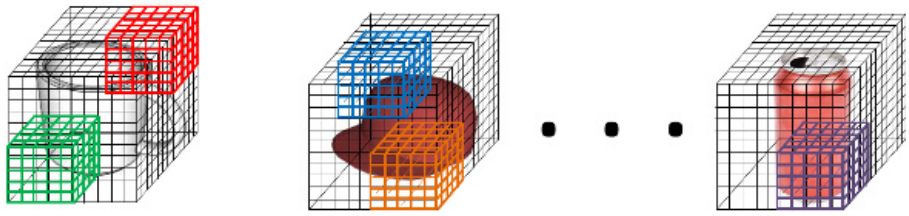

Figure 7.1: HMP over voxel data: To learn the first-layer dictionary, $5 \times 5 \times 5$ voxel patches are randomly sampled from the entire dataset of objects.

of objects stored in this voxel representation such that the features can be used for effective object recognition.

7.2.1 Dictionary Learning

The goal of dictionary learning is to compute a set of vectors (codes) such that the data can be represented using the linear combination of a sparse set of codes. As an example, consider the case where the 3D voxel data consists of the surface normal vectors at each voxel. A dictionary could be learned by sampling $5 \times 5 \times 5$ voxels (i.e. $5 \times 5 \times 5 \times 3 = 375$-dimensional patches) to create a set of code words that can accurately reconstruct local geometry 5 voxels across each dimension (see Fig. 7.1).

We learn dictionaries using K-SVD [1], a learning approach that generalizes K-Means. It learns an M-word dictionary $D = [d_1, \cdots, d_M]$ and the associated N sparse codes $X = [x_1, \cdots, x_N]$ from a matrix Y of observed data (the set of patch samples in our case) by minimizing the reconstruction error

$$\begin{aligned} &\min_{D,X} \|Y - DX\|_F^2 \\ &s.t. \ \ \forall m, \ \ \|d_m\|_2 = 1 \ \text{ and } \ \forall n, \ \ \|x_n\|_0 \leq K \end{aligned} \tag{7.1}$$

where $\|A\|_F$ is the Frobenius norm, and K controls the number of non-zero entries. K-means is a special case of K-SVD when the sparsity level K is set to be 1 and the sparse code matrix is binary.

K-SVD solves the optimization problem of eq. 7.1 by alternating between two steps at each iteration:

- The current dictionary D is used to encode the data Y by computing the sparse code matrix X.

- The codewords of the dictionary are then updated one at a time to create a new dictionary.

In the first step, given a dictionary the sparse codes are computed using batch tree orthogonal matching pursuit. Orthogonal Matching Pursuit [105] is a greedy algorithm for expressing a patch as a linear combination of a small set of code words. The algorithm first selects the best matching code word, subtracts its contribution, and then selects the next best matching code word that minimizes the residual error. This process is repeated K times to yield a K-sparse representation of the patch, where $K << M$. Despite its greedy nature, the algorithm often achieves near-optimal results. In the second step, a dictionary code is updated so as to minimize the reconstruction error, which is done via singular value decomposition (SVD). Consult [1] and [14] for technical details of the dictionary learning algorithm.

7.2.2 HMP3D: Hierarchical Matching Pursuit for 3D voxel data

HMP3D uses the hierarchical matching pursuit algorithm to build a two-layer feature hierarchy by learning dictionaries and applying the orthogonal matching pursuit encoder recursively (Fig. 7.2). The procedure is analogous to the original HMP features over images, except that patches are groups of voxels instead of pixels, and spatial pyramid max pooling is performed in 3D over voxels instead of over the 2D image plane of pixels.

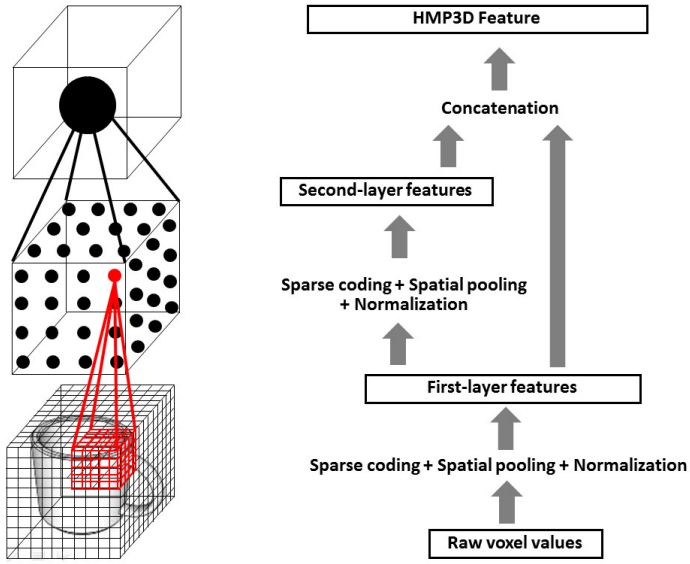

Figure 7.2: HMP3D: Hierarchical matching pursuit for 3D voxel data. In the first layer, sparse codes are learned over $5 \times 5 \times 5$ patches of voxel attributes. These sparse codes are pooled into features representing $20 \times 20 \times 20$ patches using spatial pyramid max pooling. A second layer of sparse codes are learned for encoding these features using a dictionary from sampled first-layer features. The HMP3D feature representing the whole object is a concatenation of the spatial pyramid pooled and contrast normalized sparse codes from both the first and second layers.

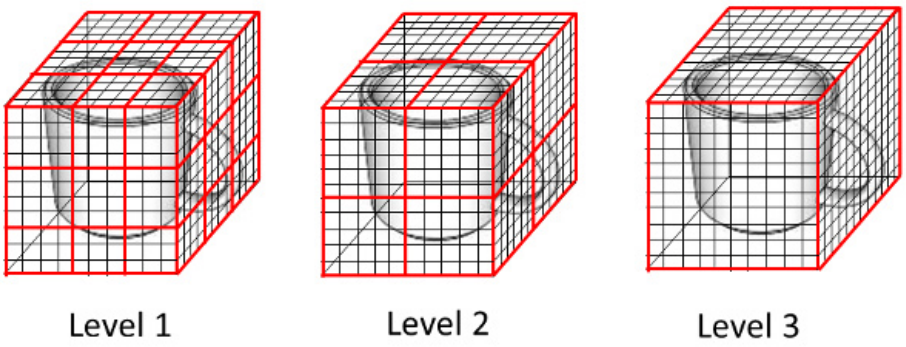

Figure 7.3: 3D spatial pyramid max pooling divides voxels into several levels of cells. Sparse codes within each cell are combined by taking component-wise maximum.

First Layer

The first layer generates features for $5 \times 5 \times 5$ voxel patches. Orthogonal matching pursuit is used to compute M-dimensional sparse codes representing each patch. Spatial pyramid max pooling is then applied to these sparse codes to generate patch level features. The voxel representation of the object is partitioned into several levels of spatial cells. The component-wise maximum over all sparse codes in a cell is taken to generate the sparse code of that cell. For example, consider a 3 level spatial pyramid pooling where the first layer divides the voxel grid into $3 \times 3 \times 3$ cells, the second layer into $2 \times 2 \times 2$ cells, and the final layer into 1 cell (see Fig. 7.3). The resulting feature vector describing the whole object would be $(27 + 8 + 1)M$-dimensional, where M is the size of the dictionary. Spatial pyramid pooling adds multiple levels of invariance to local deformation. The spatial nature of the pooling process increases the discriminative power of the resulting feature. Finally, feature vectors are normalized by their L_2 norm. This makes the features robust to changes in magnitudes, which can happen, for example, due to changes in illumination in the case of grayscale intensity and RGB values.

Second Layer

The second layer of HMP3D computes a higher-level representation over larger regions of the object. The dictionary for the second layer is learned by sampling patches over the first-layer features. For example, when sampling $4 \times 4 \times 4$ first-layer patches, each of which is a M-dimensional spatially pooled sparse code vector, the second layer dictionary would be learned over regions spanning $20 \times 20 \times 20$ voxels and the sampled patches would be $4 \times 4 \times 4 \times M$-dimensional. Similar to the first layer, spatial pyramid pooling and contrast normalization is used to generate a fixed-length feature vector describing the entire object.

The final feature representation of the object is the concatenation of the spatial

pyramid pooled features from the first and second layers. Note that Hierarchical Matching Pursuit is an unsupervised learning algorithm. These feature vectors are used in conjunction with class labels to train linear Support Vector Machine (SVM) classifiers.

7.3 Experiments

In this section, we evaluate the HMP3D feature representation on object recognition. We investigate how the features perform in both 3D object recognition on complete object models, and RGB-D object recognition on incomplete object models. Often, a complete object model cannot be obtained by capturing data from a single viewpoint due to self occlusion. When there are gaps in the object model because it has not been observed from enough viewpoints, the resulting model is incomplete. In this section, we evaluate the performance of HMP3D features on complete object models using the Princeton Shape Benchmark, and incomplete object models using the RGB-D Object Dataset.

7.3.1 3D Object Recognition on the Princeton Shape Benchmark

One large and commonly used database for 3D object recognition is the Princeton Shape Benchmark (PSB) [124]. The PSB contains $1,814$ 3D polygonal models of objects collected from the World Wide Web. The models are split into training and testing databases of 907 models each. We evaluate the usefulness of HMP3D in distinguishing between the 92 base categories in the PSB testing database. Following the procedure of [39], we learn HMP3D features using all 907 models in the training database, and then use the learned dictionaries to compute features for the 907 models in the testing database. We perform leave-one-out linear SVM classification and report overall accuracy averaged over 907 runs, each run using a different object model as test data and the other 906 as training data.

We first convert each 3D polygonal model into a 3D point cloud by randomly

Technique	SHD (NN)	SHD (SVM)	HSH (NN)	HSH (SVM)	HMP3D (NN)	HMP3D (SVM)
Accuracy (%)	55.6 [39]	55.4 [39]	59.8 [39]	68.4 [39]	65.4	**71.0**

Table 7.1: Comparison of HMP3D with alternative shape descriptors on base category classification on the Princeton Shape Benchmark testing database. Each feature is paired with either a nearest neighbor (NN) classifier or a support vector machine (SVM) classifier.

sampling points from each polygonal surface with constant density. The resulting point cloud is rescaled and voxelized so that the longest edge has unit length and a resolution of 80 voxels. A two-layer feature hierarchy is learned over surface normals as the voxel attribute. The first-layer dictionary is learned over $5 \times 5 \times 5$ voxel patches and contains 144 codewords. The second-layer dictionary is learned over $4 \times 4 \times 4$ pooled first-layer patches and contains 2000 codewords. Three-level spatial pyramid max pooling is used with 1, 8, and 27 cells in each level respectively.

We compare the performance of HMP3D with the spherical harmonic descriptor (SHD) [73] and Harmonic Shape Histograms (HSH) [39]. Fig. 7.1 shows the classification accuracy on the Princeton Shape Benchmark testing database of the various approaches. Each feature is paired with either a nearest neighbor (NN) classifier or a support vector machine (SVM) classifier. The results show that HMP3D is very powerful feature for 3D data, outperforming the alternative 3D shape descriptors when paired with a linear SVM classifier. Since HMP3D is a very high dimensional feature ($403,776$ dimensions in this experiment), it is not surprising that it perform less well when paired with a nearest neighbor classifier. Nevertheless, HMP3D still outperforms SHD and HSH when these features are also paired with a nearest neighbor classifier.

7.3.2 RGB-D Object Recognition

HMP3D is designed for 3D point clouds of objects generated from multiple views, while many RGB-D object recognition datasets are captured using a Kinect from a single view[80, 22]. In such data, the point cloud may not capture the entire object because it is constructed from only one image. HMP3D is intended for learning feature representations on point clouds created from RGB-D videos, where there is coverage of objects from multiple viewpoints so that the object model is relatively complete. Nevertheless, we wanted to evaluate HMP3D on an existing, extensive, object recognition dataset to see how HMP3D will perform single-view RGB-D images, even though that is not its intended use case. For this, we conducted a category recognition experiment on the RGB-D Object Dataset described in Chapter 2, which contains 300 objects in 51 categories recorded using a Kinect-style RGB-D camera. Each object was placed on a turntable and an RGB-D video sequence of one 360° rotation is recorded. This process is repeated with the camera mounted at approximately 30°, 45°, and 60° with the horizon. The full dataset contains 250,000 RGB and depth image pairs and we use only every 5th video frame as was done in [80]. We use the same evaluation set as in Section 3.2, produced by sampling every 5th video frame from the dataset. We also use the same 10 train/test split available on the dataset website, which selects one object instance per category on each split as test data.

We compared HMP over voxel data (HMP3D) and HMP over RGB-D images (HMP2D). Both approaches use the same pixel/voxel attributes: grayscale and RGB value from RGB images, and depth and surface normals from depth images. Two-layer dictionaries were learned on each attribute independently. HMP2D used a patch size of 5×5 pixels in the first layer, and 4×4 pooled first-layer patches in the second layer. HMP3D used a patch size of $5 \times 5 \times 5$ voxels in the first layer, and $4 \times 4 \times 4$ pooled first-layer patches in the second layer. Both techniques used a first-layer dictionary

size of 150, second-layer dictionary size of 1000. HMP2D used three level spatial pyramid max pooling with 1, 2, and 4 cells at each level respectively, while HMP3D used three level spatial pyramid max pooling with 1, 8, and 27 cells at each level respectively. The difference is that HMP2D sparse codes are pooled over cells on an image plane, while HMP3D pools sparse codes over cells in 3D space.

The results in Table 7.2 show that both HMP2D and HMP3D achieve good performance on the RGB-D Object Dataset. HMP3D is able to match the performance of existing image and point cloud based features when using linear SVMs as the classifier (82.1% for HMP3D vs. 81.9 for existing features, cf. Section 3.2). However, HMP2D outperforms HMP3D on this dataset. This is due to several factors. First, the RGB-D Object Dataset contains many objects that are dark, shiny, and/or thin (e.g. stapler, battery, cell phone). These objects have lots of missing depth values. While HMP2D can still extract features over RGB images in such cases, HMP3D must operate on a point cloud that does not fully capture the object. Even when using RGB and grayscale attributes, they are only available to HMP3D for pixels that have valid depth values. Secondly, the depth image contains large depth discontinuities between the object and the background, and HMP2D is able to capture the object's silhouette, which is a very informative cue. On the other hand, HMP3D works only on the points on the object itself, so it does not capture information about the shape of the object against the background. HMP3D has performed well on this experiment considering that it was designed for 3D point clouds created from RGB-D videos rather than single RGB-D images where partial views and missing depth values is a much bigger problem. In the next chapter, we demonstrate that HMP3D is much better suited to scene labeling tasks where complete objects are visible in the scene due to the use of RGB-D videos instead of single RGB-D images. Nevertheless, HMP3D performed better than the combination of SIFT, color histograms, texton histograms, spin images, and bounding box size in Section 3.2 (Comb). This, along with the fact that HMP2D performs even better, makes a strong case for using features learned directly

Technique	Category		
	RGB only	Depth only	RGB-D
Comb	53.1 ± 1.7	74.3 ± 3.3	81.9 ± 2.8
HMP2D [15]	82.4 ± 3.1	81.2 ± 2.3	87.5 ± 2.9
HMP3D	77.8 ± 2.5	76.5 ± 2.2	82.1 ± 3.5

Table 7.2: RGB-D Object Dataset category recognition accuracy of various techniques. Comb is the combination of SIFT, color histograms, texton histograms, spin images, and bounding box size described in Section 3.2, HMP2D is HMP over RGB-D images, and HMP3D is HMP over 3D voxels. In both HMP2D and HMP3D, features are learned over grayscale intensities and RGB values for RGB-only, depth values and surface normal vectors for Depth-only, and all four attributes for RGB-D.

from data rather than hand-designed features that were designed for different sensors (regular cameras and laser rangefinders).

7.4 Summary

In this chapter, we introduced HMP3D, a hierarchical sparse coding technique for learning features from 3D point cloud data. The original HMP algorithm learns features and performs spatial pooling over the 2D grid structure of images. In contrast, HMP3D learns features and performs spatial pooling over the 3D structure of objects made available by RGB-D (Kinect-like) cameras. HMP3D achieves outstanding performance on the Princeton Shape Benchmark containing 1814 3D polygonal models of objects, outperforming alternative state-of-the-art 3D shape descriptors. HMP3D also achieves good category recognition results on the RGB-D Object Dataset even though this is a dataset of incomplete point clouds obtained from single-view RGB-D images. In the next chapter, we demonstrate how HMP3D can be used in conjunction with sliding window detectors over RGB-D images to perform semantic scene labeling.

Chapter 8

RGB-D SCENE LABELING

8.1 Background

As robots move from the factory floor into our homes and offices, they will need increasingly sophisticated tools for analyzing and understanding the environment around them. Semantic scene labeling is one important capability that robots must master if they are to execute commands like "fetch me the can of soda from the coffee table". In such applications, the robot must not only be able to recognize soda cans and coffee tables, but also to accurately infer their positions and spatial extent in 3D space. RGB-D cameras like the Microsoft Kinect have enabled rapid progress in robot perception. In this chapter, we study how to use RGB-D videos to enable semantic scene labeling.

Scene labeling, also known as semantic scene understanding, is an extensively studied problem for images [89, 55, 129, 97, 98, 23]. There is also a large body of literature on scene understanding in 3D point clouds [135, 109, 57, 54, 27]. Like approaches designed for outdoor environments ([85, 66]), many of these operate on a 3D point cloud, often obtained using a laser rangefinder. 3D point clouds contain very important shape and spatial information, which allows for models that take advantage of context and spatial relationships between objects [145, 2]. However, using an RGB-D camera it is possible to also collect data from a completely different modality. Not only can we now get the color of each 3D point, but we also have access to a set of RGB and depth image pairs. Existing works have shown that RGB-D images can be used to recognize objects to a high degree of accuracy [80, 82, 11, 15].

In this chapter, we present a technique for 3D scene labeling of objects using

RGB-D videos as input data. A probabilistic Markov Random Field (MRF) over a voxel representation of the 3D scene is used to integrate classifier responses from both sliding window detectors evaluated on the constituent RGB-D video frames and also responses from a classifier using features extracted from the 3D scene point cloud itself. We use the Hierarchical Mathching Pursuit algorithm to learn sparse coding features for RGB-D image detectors [110] and for 3D point clouds (see Chapter 7). The resulting system learns feature representations on both RGB-D images and 3D point clouds, and does not use any hand-designed features. HMP3D is trained using data generated from synthetic 3D scenes created using CAD models from an online database. We show excellent results on the RGB-D Scenes Dataset of 22 indoor scenes containing tabletop objects and large furniture pieces.

8.2 3D Scene Labeling from RGB-D Videos

Given a 3D point cloud of a scene, the scene labeling task is to assign a semantic label to every point. We tackle the problem of labeling point clouds created by aligning a contiguous sequence of images from RGB-D videos, i.e. the output of 3D reconstruction techniques like KinectFusion [102] and RGB-D Mapping [63, 62]. Fig. 8.1 presents an overview of the proposed system. The system learns and extracts features on both the constituent RGB-D video frames, and on a voxel representation of the scene. The classifier responses from these two feature representations are combined using an MRF to produce a semantic labeling of the scene. Images, being two-dimensional, capture objects situated in front of a background, which often provides useful context. On the other hand, 3D point clouds provide spatial information that is possible, but difficult, to estimate from images [119]. As we will demonstrate, it can be beneficial to learn and extract features from both. Another advantage of our system is that it does not require ground truth annotations of every point/pixel in many scenes for training purposes, which is expensive to do. Instead, the proposed approach only requires images containing views of each object in isolation and synthetic 3D models

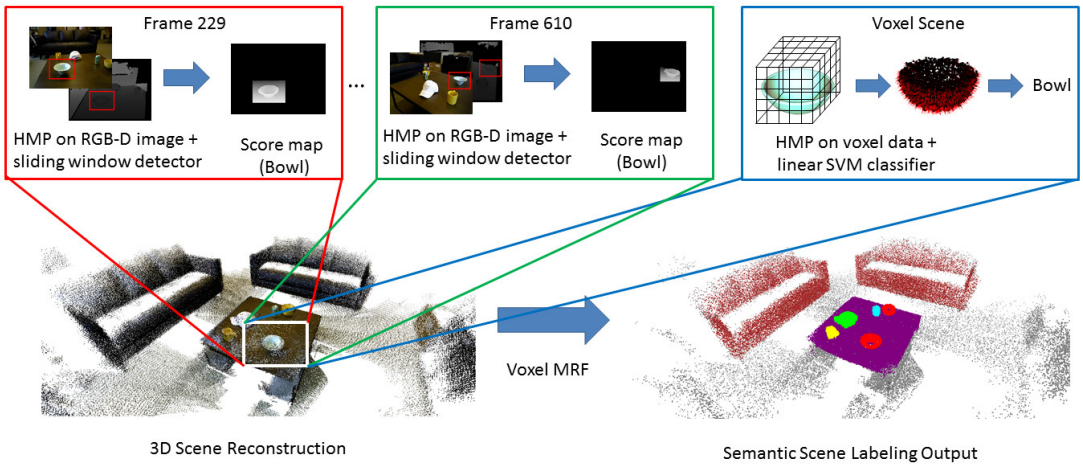

Figure 8.1: System Overview: Given a scene (left), the system extracts learned features from its constituent RGB-D video frames and the voxel representation obtained from the 3D scene point cloud. Classifiers are evaluated separately on these sensor modalities, and their responses are combined using a Markov Random Field to produce the semantic scene labeling (right; an actual result from the system).

downloaded from an online database on the Web.

We use the Patch Volumes 3D reconstruction algorithm of Henry et al. [62] to estimate the six-dimensional camera poses of each RGB-D video frame and create the scene reconstruction. The system creates a voxel representation of the resulting 3D point cloud by discretizing it into a regular grid structure. The individual cells in this grid structure are called a voxel. Let each voxel v be associated with a label $y_v \in \{1, ..., C, c_B\}$, where $1, ..., C$ are object classes and c_B is the background class. We model the joint distribution of voxel labels using a Markov Random Field with pairwise interactions. We define the optimal labeling of the scene to be one which minimizes the following energy:

$$E(y_1, \cdots, y_{|\mathcal{V}|}) = \sum_{v \in \mathcal{V}} \psi_v(y_v) + \sum_{\{i,j\} \in \mathcal{N}} \phi_{i,j}(y_i, y_j) \tag{8.1}$$

where $\mathcal{N}$ is the set of all pairs of neighboring voxels. We connect each voxel to its 26 adjacent voxels (9 below, 8 at the same height, 9 above). The data term (*first sum*) measures how well the assigned label fits the observed data and the pairwise term (*second sum*) models interactions between adjacent voxels. We first proposed this model for scene labeling in RGB-D videos in [83].

MRF-based techniques have been used for many labeling tasks, providing a unified framework for combining local evidence, such as appearance and shape, with dependencies across regions like label smoothness. A popular approach is to define the data term using local features on over-segmentations (e.g. [85, 145, 3]). These segmentation-based approaches deal with imperfect segmentation by generating a "soup of segments" [94] by running multiple segmentation algorithms on the same scene and aggregating classification results, hoping that at least some of segments will be distinctive enough to be recognized.

However, bottom-up segmentation that does not use any object label information is unreliable and can fail. Hence, in this work we instead use a training dataset of objects to estimate their true size in 3D and use that to inform our model. We train

both voxel classifiers and view-based object detectors that run on individual RGB-D frames. The responses of voxel classifiers and object detectors are both incorporated into the data term.

8.2.1 Voxel Classification

The voxel classifiers assign a probability distribution over object classes to each voxel in the scene (see top right of Fig. 8.1). There is a binary voxel classifier for each object class we wish to label. Let x denote a 3D point, v denote a voxel, and Ω_v denote the set of 3D points inside voxel v. The voxel classifiers are evaluated on every voxel in the scene, and the resulting class probability distribution for a voxel v is assigned to every 3D point x inside v, i.e. $p_{vox}(y_v|x), \forall x \in \Omega_v$.

The voxel classifiers extract features from a rectangular prism shaped region around that voxel based on the expected true size of the object, which is estimated from training data. For example, in our experiments a coffee mug voxel classifier extracts features from a 16.2cm×16.8cm×14.4cm window. This is in contrast to existing segmentation-based approaches like [3], which use bottom-up segmentation algorithms that only look at local appearance and shape and do not make use of information derived from object class. We use the HMP3D features described in Section 7.2 learned on a dataset of synthetic object models downloaded from the web. The dataset and training procedure are described in Section 8.4.

8.2.2 RGB-D Object Detection

Sliding window based approaches learn templates from object views that have been annotated with bounding boxes encompassing the entire object. An object is localized in an image by scanning over multiple positions and scales. View-based object detection has been extensively studied [31, 43, 61]. Sliding window detectors assign scores to a grid of locations in the image on which the detector window has been centered, often across multiple image scales.

The standard object detection framework uses this score map pyramid to select a subset of high scoring detector bounding boxes to label as object candidates. For scene labeling, we instead want to assign a label to every 3D point in the scene. For scenes reconstructed from RGB-D videos, each point is a pixel in some RGB-D frame and the system can remember this one-to-one mapping. RGB-D object detectors are trained using the RGB-D Object Dataset with the hard negative example mining procedure described in Section 3.3. The detectors are run on individual RGB-D frames and instead of using the score map pyramid to select bounding box candidates as in standard object detection, their responses are used to compute a score map over all pixels (see top left of Fig. 8.1). Combining the score maps from all object detectors, we get an object class probability for every pixel in every frame and, via one-to-one mapping, every 3D point in the scene.

Scoring 3D points using object detection. When computing features for a 3D point x, the system looks up the corresponding pixel and computes them using the source RGB-D frame. Let $f_c^h(x)$ be the feature vector of x for object detector c at scale h. We use features extracted from both the RGB and depth images (see Section 8.3). An object can appear in different sizes depending on its distance from the camera, so it is necessary to run sliding window detectors on multiple scaled versions of the image. While RGB-D data makes it plausible to select the "proper" scale based on the physical sizes of objects, we choose not to enforce any hard scale constraint and instead use scale as an input feature to the detector, as described in Section 3.3. Hence, the maximum detector response across all scales is the best estimate of the true object scale at a particular image position and we obtain the score map of a linear SVM object detector c:

$$s_c(x) = \max_h \{w_c^\top f_c^h(x) + b_c\} \tag{8.2}$$

where w_c and b_c is the weight vector and bias of the object detector c, learned from training data. We train one detector for each object class, so $1 \leq c \leq C$. We convert

these linear score maps into probability maps that define the probability of point x belonging to class c, using Platt scaling [106] (Algorithm 1, line 5):

$$p(c|x) = \frac{1}{1 + \exp\{us_c(x) + v\}} \tag{8.3}$$

where u and v are the parameters of the sigmoid and can be found by minimizing negative log likelihood of the training or validation set. This is not a probability over all classes, but instead a probability obtained from the binary classifier between class c and background [115]. The advantage of training C binary classifiers instead of one multi-class classifier is that we can train each detector with a different template window size and aspect ratio, using a different training dataset.

Background probability. A point belongs to the background class if it does not lie on any of the objects. Detectors provide evidence for the background class when they do not fire at a location. The object detectors are trained and evaluated independently, each seeing other objects as negative examples. To obtain a background probability, we observe that when there are C foreground classes, the following holds for the probability of point x being background:

$$p(c_B|x) \leq 1 - p(c|x), \ \forall c \in \{1, ..., C\} \tag{8.4}$$

Hence, we can use the smallest probability of the negative classes as the upper bound for that of the background class. In practice, we use the discounted value

$$p(c_B|x) = \beta \min_{1 \leq c \leq C} \{1 - p(c|x)\} \tag{8.5}$$

where $0 < \beta \leq 1$ controls the looseness of the upper bound.

The resulting probabilistic score map assigns a probability distribution $p_{im}(y_v|x)$ for every pixel in every frame, where a higher probability means it is more likely that an object of that class is present at that pixel location.

8.2.3 Integrating RGB-D image and Voxel Evidence

The data term $\psi_v(y_v)$ in Eq.8.1 is computed as

$$\psi_v(y_v) = -\ln p(y_v|\Omega_v) = \\ -\frac{1}{|\Omega_v|}\sum_{x\in\Omega_v} \alpha \ln p_{vox}(y_v|x) + (1-\alpha)\ln p_{im}(y_v|x) \tag{8.6}$$

In other words, the log likelihood of a voxel v is the arithmetic mean of the log likelihoods of its constituent points Ω_v. The probabilities from voxel classifiers and image classifiers are multiplied (summed in log-space), with $0 \leq \alpha \leq 1$ being the tradeoff parameter for balancing their respective influence.

8.2.4 Label Consistency and Geometric Information

The pairwise term $\phi_{i,j}(y_i, y_j)$ in Eq.8.1 encodes interactions between nearby voxels. The simplest pairwise interaction is the Potts model [20]

$$\lambda \cdot \mathbf{1}_{y_i \neq y_j} \tag{8.7}$$

where $\mathbf{1}_{y_i \neq y_j}$ evaluates to 1 when $y_i \neq y_j$ and 0 otherwise. This adds a constant penalty λ when adjacent elements do not share the same label. This is based on the assumption that the world is "smooth" in the sense that nearby elements tend to share the same label. However, in reality abrupt label changes do occur at object boundaries.

In image labeling, researchers have used the contrast-dependent smoothness prior [91],

$$\lambda \cdot \mathbf{1}_{y_i \neq y_j} \exp(-\theta \|x_i - x_j\|^2). \tag{8.8}$$

Here x_i and x_j are the intensities/colors of pixels i and j, so label changes between dissimilar pixels are penalized less, with θ controlling the sensitivity. This captures the intuition that label changes tend to occur at sharp edges, where adjacent pixels have very different intensities/colors. However, intensity/color changes in an image do

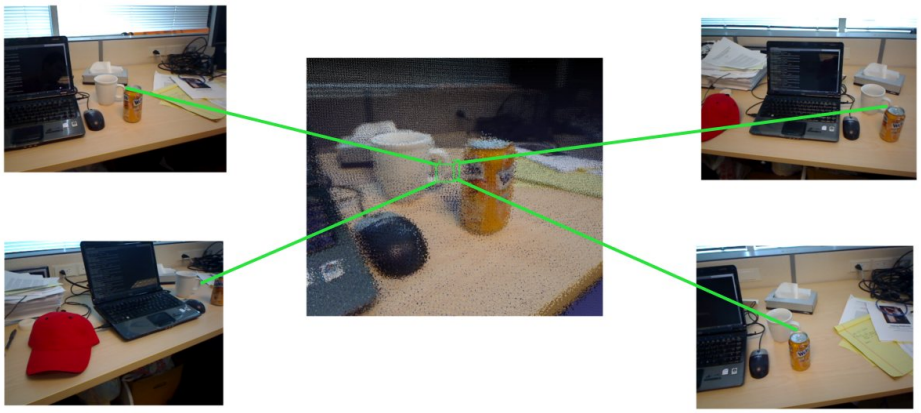

Figure 8.2: Each voxel in the scene (center) contains 3D points projected from multiple RGB-D frames. The points and their projections are known, so we can compute average likelihoods for each voxel based on the likelihoods of constituent points. Combining detections from multiple frames greatly improves the robustness of object detection.

not necessarily correspond to object boundaries, since objects can often have internal edges (e.g. logos on a soda can). Furthermore, the boundary between two similarly colored objects may not have strong image gradients.

For labeling RGB-D scenes, 3D shape information can be used as a more reliable cue for object boundaries. We incorporate this information into our model by defining

$$\phi_{i,j}(y_i, y_j) = \lambda \cdot \frac{\mathbf{1}_{y_i \neq y_j}}{d(n_i, n_j)} \left(\mathbf{I}(n_i, n_j) + \epsilon\right) \tag{8.9}$$

where λ and ϵ are balancing parameters and $\mathbf{1}_{y_i \neq y_j}$ evaluates to 1 when $y_i \neq y_j$ and 0 otherwise.

As in the Potts model [20], the pairwise term is non-zero only when $y_i \neq y_j$. $d(n_i, n_j)$ measures the difference between surface normals n_i and n_j of, respectively, voxels i and j; if $d(\cdot)$ is small, the normals are similar, i and j are on a smooth surface and the cost is high to assign them different labels. We use the L_2-distance plus a small constant as our distance metric between surface normals. $\mathbf{I}(n_i, n_j)$ is an

Figure 8.3: The scene labeling MRF's pairwise term captures the intuition that (left) object boundaries tend to have large changes in surface orientation, and that (right) objects tend to be supported by flat surfaces, leading to concave surface transitions.

indicator variable expressing whether i and j are part of a convex surface (such as the top of a cereal box) or a concave one (where a cereal box touches its supporting surface). Since objects tend to have convex shapes and concave surface transitions are more likely to occur at object boundaries, the cost is lower to cut the scene at a concave shape than a convex one, with the parameter ϵ controlling the trade-off.

This pairwise term captures the intuition that object boundaries tend to have large changes in surface orientation, and that objects tend to be supported by flat surfaces, leading to concave surface transitions (see Fig. 8.3). As was done in [3], we use

$$\mathbf{I}(n_i, n_j) = [(n_i - n_j) \cdot (i - j) > 0] \tag{8.10}$$

to indicate whether the surface transition between voxels i and j is convex. To compute this we need to ensure that all surface normals point outward and not into the object. This can be done because the camera pose of the video frame from which each point originates is known, and the surface normal should form a sharp ($> 90°$)

angle with the camera view vector.

8.2.5 Optimization

The data term $\psi_v(v)$ and the pairwise term $\phi_{i,j}(y_i, y_j)$ together define a multi-class pairwise MRF, whose energy is quickly minimized using graph cuts [20]. For the RGB-D Scenes Dataset, which contains scenes that are around 5m×5m×5m in size, the inference procedure takes less than a second. There are four free parameters in our model (α,β,λ,ϵ) and they are easy to set by hand. We leave learning the parameters to future work.

8.3 Object Detection for Scene Labeling

In this section, we describe how we use RGB-D sliding window detectors, presented in Section 3.3, to generate probabilistic score maps that are used in the scene labeling MRF described in Section 8.2. Note that the proposed MRF-based scene labeling approach only uses the probability maps. If desired, the features and detectors described in this section can be substituted with another image classification technique without changes to the rest of the framework. As an example, one simple substitution would be to use the deformable parts-based model of Felzenszwalb et al. [42].

While the HOG based sliding window classifier is among the most successful object detection techniques [31, 43], recent work [110] has shown that HMP features learned over gradient images can be used instead of HOG features to boost performance. In Section 8.5, we will compare the performance of HOG descriptors with using HMP features learned over RGB-D images (HMP2D).

We use the same HOG descriptors here as in Section 3.3 for RGB-D object detection. For HMP2D features, we learn a single-layer dictionary by sampling 5×5 patches from grayscale and depth images and extract sparse coding features [14]. These features are then used to train object detectors [110]. For our experiments we only use a root template so we do not need to use HOG features to first learn a

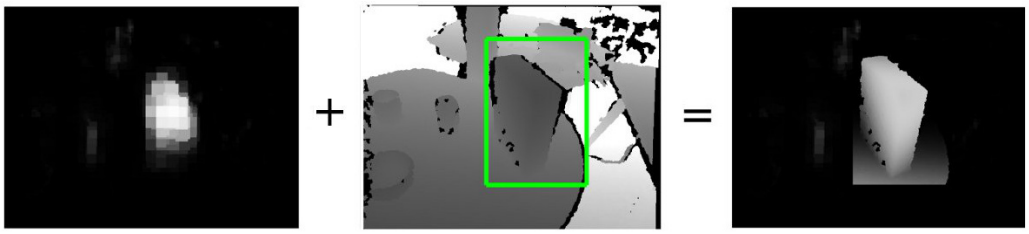

Figure 8.4: An example of using RGB-D object detectors to obtain a probability map defined on all pixels. We first run the cereal box detector to obtain a class probability at each pixel (left). We then use high scoring bounding box candidates (middle) to perform depth-based refinement and obtain the final probability map that better fits the shape of the object (right).

deformable parts structure.

The performance of a classifier heavily depends on its ability to exploit negative data. In a typical image or depth map captured by Kinect, there are on the order of 10^5 potential negative examples, which makes it impractical to use all negative examples. We use a bootstrapping procedure to mine the hard negative examples. The initial negative examples consists of cropped rectangles from background videos and objects from other categories. After training a classifier, the resulting classifier is used to search background images and select the false positives with the highest scores (hard negatives). These hard negatives are added as negative examples and the classifier is retrained. This procedure is repeated 10 times to obtain the final classifier.

When using RGB-D object detectors, we found it helpful to perform a depth-based refinement to improve the probability map defined in eq.8.9. In Fig. 8.4, we show the probability map of a cereal box sliding window detector. In this example the detector correctly finds the location of the cereal box; however, strong signals are only localized around the center of the object. The probabilities quickly decrease

near the edges of the cereal box. This is not surprising since bounding boxes centered near the edge contain a large amount of background and look very different from the cereal box itself. To better capture the spatial extents of detected objects, we refine the probability map using high scoring bounding boxes found by our object detectors. For each bounding box above a detector score threshold of -0.6, we set

$$p(c|x) = p(c|x_0) \exp(-\gamma \|x - x_0\|^2), x \in \mathcal{B} \tag{8.11}$$

where $\mathcal{B}$ is the set of 3D points in a bounding box, x_0 is the center of the corresponding bounding box, and the parameter γ controls how quickly the probability decreases with increasing depth difference. This expansion is analogous to bilateral filtering using depth. From the example in Fig. 8.4, we can see that many more pixels inside the cereal box now have a strong signal. We note that this procedure can falsely introduce some strong signals on non-object pixels (the table in this case) into the refined probability map. While a more expensive approach using more cues may yield better refinement, we found that our simple and quick approach is sufficient, as the false signals can be cleaned up by the MRF.

8.4 Synthetic Data Generation for Voxel Classification

Given a large set of real scenes with ground truth labels of every point/pixel, HMP3D features and classifiers can be learned by sampling from the dataset. However, ground truth real data is very expensive and time consuming to obtain, especially for scene labeling where every 3D point must be annotated with its label. Instead, we explore the use of synthetic data to learn HMP3D features and train voxel classifiers that we then apply to real data.

One key difficulty of using synthetic data is ensuring that classifiers learned using it remains effective when applied to real sensor data. Domain adaptation [32, 85] addresses this by training classifiers using smart methods of combining a large synthetic dataset with a small set of real data, with the aim of optimizing performance

on real data. Domain adaptation is necessary when the source and domain data is significantly different. This was the case in Chapter 4 when the target domain is velodyne LIDAR data and the source domain is Trimble 3D Warehouse CAD models. The LIDAR data is very sparse, especially for far away objects. There are also characteristic differences like windshields being transparent in real-life, but opaque in the CAD models. However, for RGB-D data the difference is less pronounced. RGB-D cameras can capture very dense and accurate point clouds and we also do not consider transparent objects in our experiments. Furthermore, the success of virtual training data synthesis from depth images for human pose estimation has been demonstrated experimentally [125] and from widespread real-world use in the Microsoft Kinect. Inspired by this, we generate a large synthetic training dataset using 3D models downloaded from Trimble 3D Warehouse [136], a large Internet repository of CAD models created by hobbyists and professionals. As our experiments demonstrate, we found empirically that this yielded excellent performance without having to mix in real data and use domain adaptation.

To closely model real-world environments as seen from videos recorded using an RGB-D camera, synthetic object models are placed in automatically generated virtual scenes where they all lie on the same plane and may be very close to and/or occlude each other. Training data is generated by sampling rectangular prism shaped regions centered on objects in these virtual scenes.

For each sampled object, raycasting is used to generate a 3D point cloud of the object aggregated from a randomly selected number of viewpoints between three to eight. Gaussian noise is added to simulate sensor noise in real data. The sampled object is first scaled to be unit length in its longest dimension. It is then rotated by a random amount. In our experiments, we only rotate about the vertical axis as we assume that objects will lie on flat surfaces and the ground plane orientation can be estimated from data. Finally, the object is scaled by a factor uniformly sampled from an interval ($[0.85, 1]$ in our experiments). A voxel representation for the object

is obtained such that a unit length contains a constant number of voxels (80 in our experiments). This yields a scale-invariant representation that captures object shape at roughly the same granularity regardless of its true size. The voxel representation thus contains the same number of voxels in each dimension for every sample of a particular object class. For each voxel it is possible to compute attributes that describe it. In our experiments, we use grayscale intensity, RGB value, binary occupancy (does at least one point fall within the voxel), and surface normal vector. This procedure is repeated for another randomly selected object until the desired number of training set size is reached.

This procedure is used to generate training data for the voxel classifiers. Separate datasets are generated for each object class because the number of voxels along each dimension is dependent on the true size of the object. Positive examples are generated by sampling objects of the target class from the virtual scenes. Negative examples are generated by sampling objects from other classes, as well as regions that contain objects from the target class but the object is positioned off-center. As our experiments will demonstrate, the resulting training data sufficiently captures variance that may be encountered in real-world data: intra-class appearance and shape variation is captured by differences in the object models downloaded from Trimble 3D Warehouse, while scale, orientation, clutter, and occlusion variation is captured via the virtual environment generation and object sampling procedure.

8.5 Experiments

In this section, we evaluate the overall system on a 3D scene labeling task and the HMP3D representation on an RGB-D object recognition task.

To investigate the proposed scene labeling approach, we evaluate it on the task of detecting small tabletop objects and large furniture in typical home and office environments. The RGB-D Scenes Dataset (see Chapter 2) contains 22 such scenes. Eight of the scenes are annotated with only tabletop objects, containing between

three to twelve objects placed on a flat surface, including bowls, caps, cereal boxes, coffee mugs, and soda cans. Fourteen of the scenes are annotated with both tabletop objects and furniture. These 14 scenes may contain furniture including chairs, coffee tables, sofas, and tables and also three to five objects from the same 5 categories placed on a coffee table or table.

8.5.1 Experiment Setup

To train voxel classifiers, we downloaded models of objects from all nine classes from the Trimble 3D Warehouse. In addition, we also downloaded models of other objects commonly found in office and home environments for use as negative data. We then use the procedure for generating synthetic training data described in Section 8.4 to generate approximately $100,000$ training examples. In this procedure, each virtual scene contains only either tabletop objects or furniture. The real-world dimensions of the voxel representations of each object class were computed from training data to be, in cm: bowl ($11.9 \times 11.9 \times 6.36$), cap ($14.9 \times 15.9 \times 9.6$), cereal box ($27.4 \times 26.7 \times 33.7$), coffee mug ($16.2 \times 16.8 \times 14.4$), coffee table ($81.8 \times 81.1 \times 38.5$), chair ($84.7 \times 84.7 \times 96.8$), soda ($12.2 \times 12 \times 14.9$), sofa ($120.0 \times 118.8 \times 72.5$), and table ($110.2 \times 110.2 \times 77.8$). The estimated size is always bigger than the actual object because the voxel representation does not tightly bound the object, depending on variation of scale and orientation in the training data. The voxel grid resolution is set to be 80 along the longest dimension, i.e. the longest edge of each object class's "bounding cube" contains around 80 voxels. Images from the RGB-D Object Dataset (see Chapter 2) are used to train RGB-D image sliding window detectors for the tabletop objects.

For many robotics applications, it is not necessary to exhaustively evaluate every voxel classifier on every voxel. Often a robot can accurately estimate its orientation and the position of the ground plane. Also, objects are often placed on top of furniture. Taking advantage of such prior knowledge about the environment often improves both labeling accuracy and computational efficiency. In our experiments, we assume that

furniture is always placed on the ground and tabletop objects are always placed on tables and the ground plane is known, which is true for the RGB-D Scenes Dataset. The ground plane is removed and a region growing technique is used to segment out isolated blobs of voxels. The system then applies furniture classifiers on each blob and classifier responses are applied to every voxel in the blob. When the system detects a coffee table or a table, it uses the same region growing algorithm to segment isolated blobs on top of the table and evaluates the tabletop object voxel classifiers on them.

We compare the performance of using HOG over RGB-D images with the scene labeling MRF (Det3DMRF), using HMP over RGB-D images with the scene labeling MRF (HMP2D), using HMP3D with the scene labeling MRF (HMP3D), and the combination of HMP2D and HMP3D as proposed in Section 8.2 (HMP2D+3D). HMP2D is very similar to Det3DMRF, except that Histogram of Oriented Gradients (HOG) features have been replaced by sparse coding features learned from RGB and depth image data, an idea first proposed in [110].

For HMP2D, we use a one-layer architecture with a dictionary of size $M = 100$ for both grayscale and depth attributes and is learned from 5×5 image patches. For HMP3D, we use $M = 150$ for the first layer dictionary and $M = 1000$ for the second layer. The first-layer patch size is $5 \times 5 \times 5$ and for second-layer it is $4 \times 4 \times 4$ first-layer patches. The three-level spatial pyramid pooling in Fig. 7.3 is used. SVM training was performed using LIBLINEAR [38] and the outputs are transformed by a sigmoid to yield probabilities for the scene labeling MRF. The MRF parameters were $\alpha = 1/6$, $\beta = 1/10$, $\epsilon = 1/4$, and $\lambda = 50$.

8.5.2 Scene Labeling Results

Table 8.1 shows the precision and recall of scene labeling for each object category as well as the overall performance, with each category given equal weight. The precision and recall is computed on a per-point basis. For example the recall of bowl is the proportion of 3D points actually belonging to the bowl class successfully labeled by

Technique	Precision / Recall										
	Bowl	Cap	Cereal Box	Coffee Mug	Soda Can	Coffee Table	Chair	Sofa	Table	Background	Overall
Det3DMRF	65.4 / 84.7	62.9 / 98.7	64.2 / 96.3	48.1 / 84.7	61.1 / 95.1	12.7 / 15.8	18.4 / 20.8	25.1 / 30.9	14.6 / 22.1	43.9 / 99.5	41.6 / 64.9
HMP2D	74.4 / 85.0	74.9 / 98.6	79.9 / 98.6	64.4 / 87.8	78.2 / **98.1**	11.9 / 17.9	17.7 / 17.2	29.3 / 39.8	16.4 / 23.3	42.9 / **99.6**	49.0 / 66.6
HMP3D	**100.0** / **96.2**	**91.3** / 98.2	85.1 / **100.0**	**90.0** / **93.9**	**100.0** / 81.9	96.1 / **100.0**	57.6 / **100.0**	82.7 / **100.0**	95.3 / **98.7**	**99.9** / 80.0	89.8 / 94.9
HMP2D+3D	97.0 / 89.1	82.7 / **99.0**	**96.2** / 99.3	81.0 / 92.6	97.7 / 98.0	**98.7** / 98.0	**89.7** / 94.5	**92.5** / 92.0	**97.6** / 96.0	95.8 / 95.0	**92.8** / **95.3**

Table 8.1: Per-category and overall (macro-averaged across categories) precision and recall on the fourteen scenes in the RGB-D Scenes Dataset containing furniture.

the system as bowl, while its precision is the proportion of 3D points labeled as bowl that actually belong in the bowl class.

We found that RGB-D image detectors did not perform well on furniture. While there are many images of such objects on the Internet, RGB-D data from many different viewpoints is scarce. Also, furniture is categorized by function rather than form, so there can be a lot of intraclass variation in shape and appearance. Finally, furniture pieces are large objects which are often only partially visible due to occlusion or from being cut off by the camera view cone. This makes detecting them using RGB-D image detectors difficult. On the other hand, HMP3D can handle objects of any size since it operates on the 3D scene reconstruction built from aggregating many video frames. Det3DMRF and HMP2D both achieve similar accuracy. As was the case in [110], using sparse coding features learned from image data is usually slightly better than using HOG features. The fourteen scenes containing furniture are more challenging than the eight scenes that only contain tabletop objects because in the former each video is only one rotation around the table, while in the latter objects are often seen from similar viewpoints multiple times. The shorter videos make it harder to prune out false positives, as objects are not detected repeatedly as often. HMP3D performs well on most objects, but sometimes confuses bowls as being coffee mugs. Often HMP2D and HMP3D do not make mistakes on the same objects, and hence the combination of the two yields the best scene labeling accuracy.

Fig. 8.5 shows scene labeling results from the eight scenes in the RGB-D Scenes Dataset containing no annotated furniture, and Fig. 8.6 shows scene labeling results from the fourteen scenes in the RGB-D Scenes Dataset that contain annotated furniture. Objects are colored by their category: bowl (red), cap (green), cereal box (blue), coffee mug (yellow), soda can (cyan), chair (pink), coffee table (purple), sofa (brown), table (navy blue). The figures show that the proposed approach is able to correctly identify most tabletop and furniture objects. The object boundaries are also very clean. In particular, notice that the system is able to correctly distinguish

between coffee tables and tables. These two object categories are similar in shape and appearance, but are different in height. Since our approach estimates the true size of objects from training data, the voxel classifiers use features computed from different sized regions for each class. The region used for the coffee table classifier is shorter, and thus only the legs of a table would be visible. On the other hand, a coffee table would only occupy the lower portion of the region used by a table classifier. Hence, the features after 3D spatial pooling look very different for coffee table and table. This ability is unique to HMP over voxel data and cannot be achieved by previous feature learning methods applied to RGB-D images, which perform spatial pooling over the 2D grid structure of images as opposed to pooling in 3D space.

There were a few failure cases on this dataset. For example, the system confused two computer workstations placed side by side underneath an office desk as being a coffee table (bottom right of Fig. 8.6). A more comprehensive training dataset may have prevented this. When the region-growing segmentation algorithm fails, the system will also fail to segment out the exact extents of large furniture objects, such as the sofa in the coffee room (fifth and sixth row of Fig. 8.6). Finally, when there are errors in the 3D scene reconstruction, the system may also fail to correctly classify an object (e.g. bowl in the first scene in the third row of Fig. 8.6).

8.5.3 Variants of the Scene Labeling MRF

The scene labeling MRF framework described in Section 8.2 can be divided into two parts: the data term which incorporates classifiers that make decisions based on local appearance and shape, and the pairwise term which enforces global smoothness and prior information about the geometric structure of objects and scenes. In this section, we perform experiments to investigate the contribution of these two parts. We consider variants of our scene labeling MRF involving (a) only the data term in our model (*DetOnly*); (b) incorporating the standard Potts smoothness term, Eq.8.7 (*PottsMRF*); (c) incorporating the contrast-dependent smoothness term, Eq.8.8, over

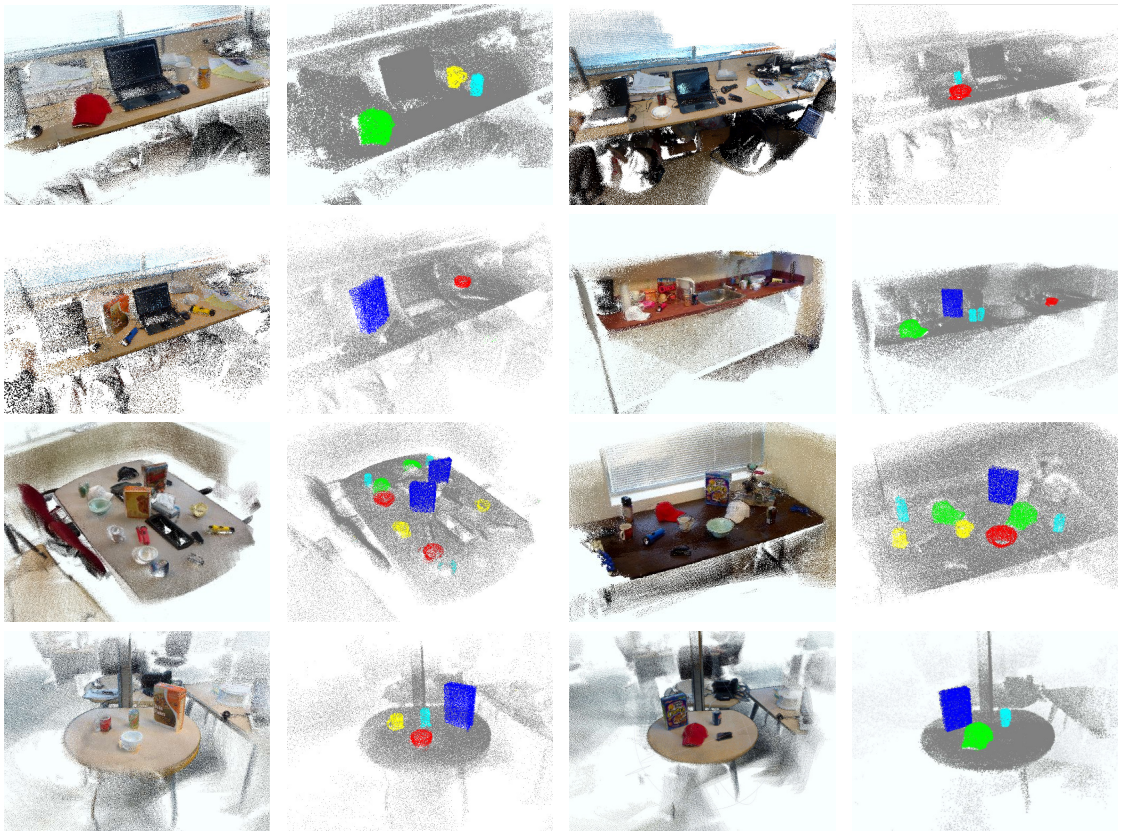

Figure 8.5: 3D scene labeling results for the eight scenes in the RGB-D Scenes Dataset with no annotated furniture. For each scene, the 3D reconstruction and results using a combination of HMP on RGB-D images and HMP over voxel data is shown. Objects are colored by their category: bowl (red), cap (green), cereal box (blue), coffee mug (yellow), soda can (cyan), chair (pink), coffee table (purple), sofa (brown), table (navy blue). Best viewed in color.

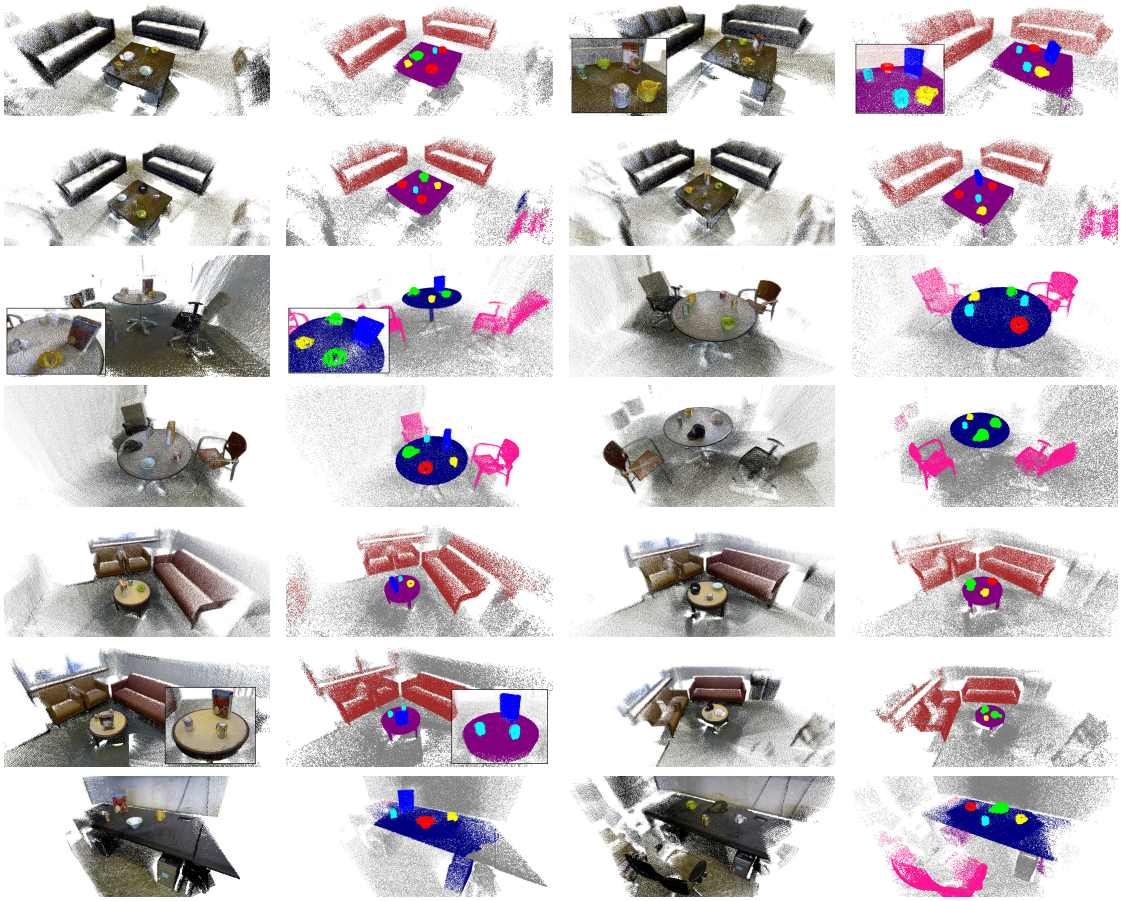

Figure 8.6: 3D scene labeling results for the fourteen scenes in the RGB-D Scenes Dataset containing furniture. For each scene, the 3D reconstruction and results using a combination of HMP on RGB-D images and HMP over voxel data is shown. Objects are colored by their category: bowl (red), cap (green), cereal box (blue), coffee mug (yellow), soda can (cyan), chair (pink), coffee table (purple), sofa (brown), table (navy blue). Best viewed in color.

Technique	Precision	Recall	Micro F-score	Macro F-score
Random	16.7	16.7	16.7	7.4
AllBG	87.5	87.5	87.5	16.7
DetOnly	61.7	87.9	72.5	71.3
PottsMRF	86.9	88.4	87.7	87.4
ColMRF	89.4	87.8	88.6	88.5
Det3DMRF	91.0	88.8	89.9	89.8

Table 8.2: Precision, Recall, Micro F-score and Macro F-score for two naïve algorithms (random and labeling everything as background) and variants of the scene labeling MRF.

voxels that take on the mean color of its constituent points (*ColMRF*). Each variant uses only HOG-based object detectors for the data term and is evaluated on the eight scenes in the RGB-D Scenes Dataset involving only tabletop objects and no furniture.

Table 8.2 compares the precision, recall, micro F-score and macro F-score of variants of the scene labeling MRF, as well as the naïve approaches of randomly labeling each point (*Random*) and labeling everything as background (*AllBG*). The Micro F-score is computed from the overall precision and recall of the scene labeling, while Macro F-score is the average of the F-scores of the five categories, each computed separately. Random is obviously terrible regardless of the performance metric used. Since most points are background, AllBG appears to performs well in terms of precision, recall, and micro F-score. However, it fails to detect any of the objects and hence performs poorly when each category is given equal weight in the macro F-score.

Using the scene labeling MRF with only the data term leads to low precision due to the high number of false positives. PottsMRF improves upon this by significantly increasing precision while also yielding modest gains in recall. Det3DMRF boosts precision and recall further; table points that are mislabeled as objects because they

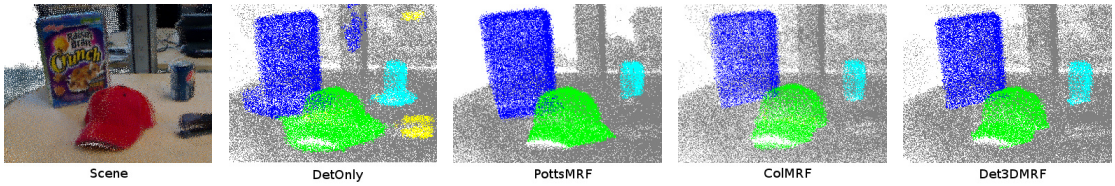

Figure 8.7: Close-up of a scene containing a cap (green), a cereal box (blue), and a soda can (cyan). From left to right, the 3D scene; Detection-only leaves patches of false positives; Potts MRF removes isolated patches but cannot cleanly segment objects from the table; Color MRF includes part of the table with the cap because it is similar in color due to shadows; the proposed detection-based scene labeling obtains clean segmentations. Best viewed in color.

often lie inside high scoring detector bounding boxes can be removed because there is often a sharp concave normal transition between the table and the object. ColMRF can also robustly segment objects if the color is significantly different, but overall does not perform as well as Det3DMRF on the RGB-D Scenes Dataset. Although the gains from Det3DMRF seem modest when looking at the numbers, qualitatively segmentation is improved substantially as points on the table are now almost always excluded from the objects (see Fig. 8.7).

Running time. The Patch Volumes mapping algorithm [62] used for scene reconstruction runs in real-time for our RGB-D videos, which were collected at 15-20 Hz. We evaluate object detectors on every 10th frame, or every 0.5 seconds. Our current single-threaded MATLAB implementation is not yet real-time, requiring 4 seconds to process each frame. The HMP3D feature extraction and voxel classification requires 10 seconds to run, and are run only once at the end for each video. Voxelization and graph cut inference take negligible time. The overwhelming majority of computation is spent on feature extraction from RGB-D video frames and from the voxelized 3D scene. Given that these procedures are highly parallelizable, we believe that a more

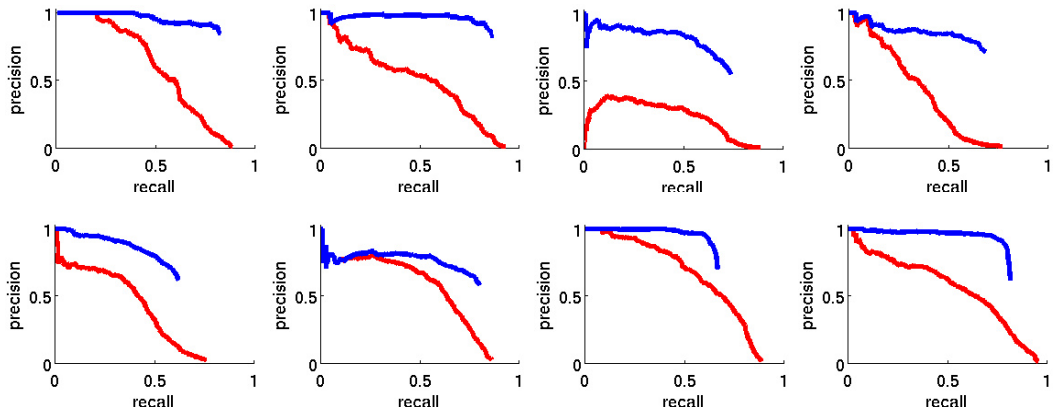

Figure 8.8: Precision-recall curves comparing the performance of labeling images with bounding boxes of detected objects. Each plot shows results on one of the eight video sequences in the RGB-D Scenes Dataset containing only tabletop objects, aggregated over all five category detectors. Frame-by-frame object detection is drawn in red, while 3D scene labeling (our approach) is drawn in blue.

optimized, multi-threaded implementation will be able to run sliding window detectors on every 10th frame, while the voxel classifiers can be run in several seconds per scene.

8.5.4 Refining Image-based Object Detection

While the main focus of our scene labeling approach is assigning a semantic label to every 3D point in the scene, it can also be used to perform object detection where each object is labeled with a bounding box, as is commonly done in the computer vision community (e.g. PASCAL VOC challenge). To do this, we use the labeled 3D scene to validate bounding box proposals from the constituent video frames. We run the object detectors with a low threshold and prune out bounding box candidates whose labels do not agree with the majority label of points in a central region of the bounding box. Fig. 8.8 shows precision-recall curves obtained from both the

individual frame-by-frame object detections (red) and detections validated by our 3D scene labeling (blue) on the eight scenes in the RGB-D Scenes Dataset containing only tabletop objects. Each point along the curve is generated by ranking detections from all five category detectors together and thresholding on the detection score. It is clear that 3D scene labeling can significantly reduce false positives by aggregating evidence across the entire video sequence. While the precision of the frame-by-frame detection approach rapidly decreases beyond 60% recall for all eight scenes, using 3D scene labeling it is possible to obtain 80% recall and 80% precision in a majority of them.

8.6 Summary

This chapter presented an approach for semantic labeling of 3D scenes. HMP3D classifiers are trained using a synthetic dataset of virtual scenes generated using CAD models from an online database. Our scene labeling system combines features learned from RGB-D images and 3D point clouds to assign a semantic label to every 3D point in the scene. Experiments on the RGB-D Scenes Dataset demonstrate that the proposed approach can be used to label indoor scenes containing both small tabletop objects and large furniture pieces. In particular, the incorporation of geometric cues including surface smoothness and convexity via a pairwise potential in the scene labeling MRF is critical for obtaining good segmentation at the object boundaries. We also observed that the scene labeling approach can also be used to refine bounding box object detections in individual RGB-D images, leading to drastically improved precision at no cost to recall.

The experiments in this chapter demonstrated that learning features on both the RGB-D video frames and from the voxelized 3D scene leads to better performance than hand-designed features, and that combining these learned features from both modalities leads to even better performance. Since the scene labeling approach proposed in this chapter uses features learned from sensor data rather than features

hand-designed for each sensor modality, it will be able to take advantage of new sensor technology and additional sensor modalities with little modification.

Chapter 9

CONCLUSION

This thesis has demonstrated that the combination of RGB and depth (including 3D point cloud data) at high frame rates, made possible by RGB-D cameras, is helpful for various recognition tasks including object recognition, object detection, and semantic scene labeling. We explored existing features and algorithms designed for RGB images and 3D point cloud data, and also presented new features and algorithms designed for high frame rate RGB-D data. During this investigation, we discovered a number of key insights regarding RGB-D data.

In Chapter 2, we introduced the RGB-D Object Dataset, a large dataset of 250,000 RGB-D images of 300 objects in 51 categories, and 22 RGB-D videos of a subset of these objects in indoor home and office environments. The dataset is annotated with the ground truth labels of objects and furniture pieces. We demonstrated that by taking advantage of depth data, it is possible to obtain cropped and segmented RGB-D images and 3D point clouds of objects. We also showed that RGB-D videos can easily be annotated by constructing 3D scene point clouds, labeling those, and projecting the labels back into the original RGB-D video frames. The dataset is publicly available at http://www.cs.washington.edu/rgbd-dataset.

RGB images and depth images provide complementary information: RGB images contain texture and color information, while the depth channel gives shape information and is useful for object segmentation. Hence, the combination of the two yields better object recognition and object detection results than either one alone. This was demonstrated in Chapter 3, in which we presented techniques that use existing state-of-the-art features and classifiers for object recognition in RGB-D images that

perform category and instance recognition independently. We also presented a sliding window based approach to object detection in RGB-D images. We evaluated these techniques on the RGB-D Object Dataset to establish a baseline performance. The results show that combining color and shape information helps improve classification accuracy over using either sensor modality alone.

The combination of features from different sensor modalities requires ensuring that each feature will be weighed appropriately by the classifier. Distance learning is one widely used method in the literature for doing this. In Chapter 4, we presented an exemplar-based distance learning technique for 3D point cloud data that explicitly learns how to combine different features and weigh their respective influence so as to maximize discrimination between object classes. We evaluated this technique on an urban driving dataset containing ten scenes. We then refined the distance learning approach by learning per-object distances instead of per-exemplar distances in Chapter 5. The Instance Distance Learning approach retains the property of being able to combine multiple features, which we used to combine features extracted from RGB-D data. The approach was also able to select informative views from the training data and eliminate uninformative ones, leading to faster distance function evaluation at test-time.

The granularity of object recognition (category vs instance vs pose) required depends on the task to be accomplished. When the system needs to distinguish between many classes, computation can be expensive. Recognition tasks of different granularity can be arranged into a hierarchy so that a tree-based classification scheme can be used for efficient, scalable object recognition. In Chapter 6, we presented Object-Pose Trees, a tree-based approach that simultaneously performs object recognition at the category, instance, and pose levels. The Object-Pose Tree also scales logarithmically with the number of objects that need to be distinguished. Experiments on the RGB-D Object Dataset demonstrate that the Object-Pose Tree is able to obtain excellent category, instance, and pose recognition accuracy while being more computationally

efficient than solving each of these three tasks separately.

RGB-D data contains multiple sensor modalities, which opens the possibility of extracting features from many attributes such as color, occupancy, surface normal orientation. Instead of designing new features for every attribute, unsupervised feature learning enables feature extraction from any attribute without having to design a new feature each time. Such learned features can even outperform existing features designed for RGB image and 3D point cloud data. In Chapter 7, we presented HMP3D, an unsupervised feature learning approach for 3D point cloud data. We demonstrated that HMP3D can be used to learn hierarchies of features from different attributes including color, gradient, shape, and surface normal orientation. Experiments on the RGB-D Object Dataset demonstrate that they work well for RGB-D image classification even though they are designed for 3D point clouds created from RGB-D videos rather than individual RGB-D images, where only partial views are available due to self-occlusion.

Finally, the high frame rate of RGB-D cameras allows integrating information across a video stream containing multiple views of a scene, which can be used to perform semantic scene labeling and decrease false positives in object detection. In Chapter 8 we presented a scene labeling approach for scenes constructed from RGB-D videos. The scene labeling MRF leverages both image-based object detection techniques that run on a single RGB-D image and 3D-based object recognition techniques that run on the scene constructed from the entire video sequence. The MRF also incorporates surface smoothness and convexity into the model, which significantly improves the quality of object segmentation.

This dissertation has demonstrated the potential of RGB-D cameras for improving performance in a range of perception problems including object recognition and object detection at the category, instance, and pose level, as well as on the problem of semantic scene labeling. We first presented approaches based on existing image and point cloud based features and classification techniques which work well on RGB-D

data. Then we demonstrated that it is possible to do even better by designing new features and algorithms that specifically utilize RGB-D cameras' advantages over traditional cameras and range sensors, namely high resolution, high frame rate, and easy alignment of color image and 3D shape.

9.1 Limitations and Future Work

By making use of all forms of data from RGB-D cameras, including color images, depth images, and 3D point clouds, we have achieved excellent performance on various semantic perception tasks. However, there are still a number of limitations in this dissertation, leaving room for future work.

While the RGB-D Object Dataset is the largest dataset of its kind, RGB-D data is still scarce, particularly when it comes to data with pixel-level ground truth annotation. Evaluation of the proposed approaches in this dissertation on larger and more datasets could yield further insights into their strengths and weaknesses. For object recognition, other RGB-D datasets include 3D-Net [142] and the Willow Garage Solutions in Perception Challenge dataset [51]. One can also evaluate the semantic scene labeling system on the NYU Depth Dataset [126] and the Berkeley 3-D Object Dataset [68]. However, both of these datasets only contain several views of each scene as opposed to dense video coverage as in the RGB-D Scenes Dataset. This could lead to weaker results, as the strength of the proposed scene labeling MRF of Chapter 8 is its ability to perform spatial-temporal integration across many views in the presence of noisy object detectors.

With enough data, scaling up of the algorithms presented in this thesis to many objects and many categories can be done using Object-Pose Trees. Object-Pose Trees can incorporate levels in the tree above the category-level, for example by using WordNet hypernym-hyponym relations as in ImageNet [34] or by learning it from data [41, 9]. Object detection can be sped up by arranging sliding window detectors into a tree of classifiers as in the Object-Pose Tree. Classifiers at higher levels of

the tree can be used to limit the search space for classifiers at the lower levels. For example, if a coffee mug detector has a strong detection in a certain part of the image at a certain scale, we can choose to only evaluate the coffee mug instance detectors within a subimage and a narrow range of scales centered around the category detector's signal. This would lead to computational savings, and possibly accuracy improvements as well. This idea can also be applied to the sliding window detectors and voxel classifiers in the scene labeling system and is left for future work.

We demonstrated in Chapter 6 that the Object-Pose Tree is fast enough and accurate enough to be deployed in a real-world application: the Object-Aware Situated Interactive System [148]. While we also demonstrated very promising results on semantic scene labeling, our current MATLAB implementation of the scene labeling MRF is too slow to be used in a real system. The main bottleneck is feature extraction and evaluation of sliding window detectors. A more optimized implementation running on both the GPU and CPU could be deployed on a robot. Other techniques presented in this thesis can also be used as part of a larger system, for example a smart web lab system that can automatically track and log experimenters' action. In such an application, object recognition can be used to recognize and track objects being manipulated by the experimenter, such as beakers and flasks of chemicals. Such an experiment would validate the usefulness of the techniques presented in this dissertation in an integrated system.

Feature representation is one of the most important and challenging problems in designing recognition systems. In Chapter 7, we demonstrated that unsupervised feature learning can yield a strong feature for 3D point cloud object recognition and is a promising avenue of further research. Hierarchical matching pursuit learns a hierarchy of feature representations for 3D point cloud data over different attributes, such as color, voxel occupancy, and surface normal orientation, without modification to the algorithm. There are many other properties that could also potentially be useful for recognition, such as grayscale value, reflectance, which have not been explored in

this dissertation. The advantage of unsupervised feature learning is that the same technique can be applied to these properties without modification to the underlying learning algorithm. Addition experiments on different recognition tasks, such as material recognition, and activity recognition, would yield further insights into the strengths and weaknesses of learning feature representations. The question of what is the optimal feature representation for different recognition tasks remains open.

BIBLIOGRAPHY

[1] M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006.

[2] A. Anand, H. S. Koppula, T. Joachims, and A. Saxena. Contextually guided semantic labeling and search for three-dimensional point clouds. *IJRR*, 32(1):19–34, 2013.

[3] A. Anand, H.S. Koppula, T. Joachims, and A. Saxena. Semantic labeling of 3D point clouds for indoor scenes. In *NIPS*, 2011.

[4] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D Gupta, G. Heitz, and A. Ng. Discriminative learning of Markov random fields for segmentation of 3D scan data. In *Proc. of CVPR*, 2005.

[5] J. Assfalg, M. Bertini, A. Del Bimbo, and P. Pala. Content-based retrieval of 3-D objects using spin image signatures. *IEEE Transactions on Multimedia*, 9(3), 2007.

[6] M. Bacchiani and B. Roark. Unsupervised language model adaptation. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 2003.

[7] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.

[8] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, pages 509–522, 2002.

[9] S. Bengio, J. Weston, and D. Grangier. Label Embedding Trees for Large Multi-Class Tasks. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[10] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2), 1992.

[11] M. Blum, J. T. Springenberg, J. Wulfing, and M. Riedmiller. A learned feature descriptor for object recognition in rgb-d data. In *ICRA*, pages 1298–1303, 2012.

[12] L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In *Proc. of CVPR*, 2011.

[13] L. Bo, X. Ren, and D. Fox. Kernel Descriptors for Visual Recognition. In *Advances in Neural Information Processing Systems (NIPS)*, December 2010.

[14] L. Bo, X. Ren, and D. Fox. Hierarchical Matching Pursuit for Image Classification: Architecture and Fast Algorithms. In *NIPS*, 2011.

[15] L. Bo, X. Ren, and D. Fox. Unsupervised Feature Learning for RGB-D Based Object Recognition. In *ISER*, 2012.

[16] L. Bo, X. Ren, and D. Fox. Multipath sparse coding using hierarchical matching pursuit. In *CVPR*, June 2013.

[17] L. Bo and C. Sminchisescu. Efficient Match Kernel between Sets of Features for Visual Recognition. In *Advances in Neural Information Processing Systems (NIPS)*, December 2009.

[18] L. Bo, L. Wang, and L. Jiao. Feature scaling for kernel fisher discriminant analysis using leave-one-out cross validation. *Neural Computation*, 18(4):961–978, 2006.

[19] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM.

[20] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE PAMI*, 23(11):1222–1239, 2001.

[21] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[22] B. Browatzki, J. Fischer, B. Graf, H. H. Bulthoff, and C. Wallraven. Going into depth: Evaluating 2d and 3d cues for object classification on a new, large-scale object dataset. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1189–1195. IEEE, 2011.

[23] J. Carreira, F. Li, and C. Sminchisescu. Object recognition by sequential figure-ground ranking. *International journal of computer vision*, 98(3):243–262, 2012.

[24] R. Caruana. Multitask learning: A knowledge-based source of inductive bias. *Machine Learning*, 28, 1997.

[25] C-C. Chang and C-J. Lin. *LIBSVM: a library for support vector machines*, 2001.

[26] C. Chelba and A. Acero. Adaptation of maximum entropy capitalizer: Little data can help a lot. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 285–292, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[27] A. Collet, M. Martinez, and S. Srinivasa. Object recognition and full pose registration from a single image for robotic manipulation. *IJRR*, 30(10), 2011.

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

[29] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR*, 2:265–292, 2001.

[30] W. Dai, Q. Yang, G. Xue, and Y. Yu. Boosting for transfer learning. In *International Conference on Machine Learning (ICML)*, 2007.

[31] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. of CVPR*, 2005.

[32] H. Daumé III. Frustratingly easy domain adaptation. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2007.

[33] H. Daumé III and D. Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research (JAIR)*, 26, 2006.

[34] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proc. of CVPR*, 2009.

[35] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 304–311. IEEE, 2009.

[36] B. Douillard, D. Fox, and F. Ramos. Laser and vision based outdoor object mapping. In *Proc. of RSS*, 2008.

[37] S. Escalera, D. M. J. Tax, O. Pujol, P. Radeva, and R. P. W. Duin. Subclass problem-dependent design for error-correcting output codes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(6):1041–1054, 2008.

[38] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research (JMLR)*, 9:1871–1874, 2008.

[39] J. Fehr and H. Burkhardt. Harmonic shape histograms for 3d shape classification and retrieval. In *IAPR Conference on Machine Vision Applications*, pages 384–387, 2007.

[40] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(4):594–611, 2006.

[41] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 524–531. IEEE, 2005.

[42] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:1627–1645, 2010.

[43] P. F. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Proc. of CVPR*, 2008.

[44] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[45] S. Foix, G. Alenya, and C. Torras. Lock-in time-of-flight (tof) cameras: a survey. *Sensors Journal, IEEE*, 11(9):1917–1926, 2011.

[46] G. Forman, M. Scholz, and S. Rajaram. Feature shaping for linear svm classifiers. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 299–308. ACM, 2009.

[47] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning (ICML)*, pages 148–156, 1996.

[48] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2), 2000.

[49] A. Frome, Y. Singer, and J. Malik. Image retrieval and classification using local distance functions. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[50] A. Frome, Y. Singer, F. Sha, and J. Malik. Learning globally-consistent local distance functions for shape-based image retrieval and classication. In *IEEE International Conference on Computer Vision (ICCV)*, 2007.

[51] Willow Garage. Solutions in perception challenge dataset. http://vault.willowgarage.com/wgdata1/vol1/solutions_in_perception/, 2011.

[52] T. Gevers and A. W. M. Smeulders. Color-based object recognition. *Pattern recognition*, 32(3):453–464, 1999.

[53] D. Gildea. Corpus variation and parser performance. In L. Lee and D. Harman, editors, *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pages 167–202, 2001.

[54] J. Glover, G. Bradski, and R.B. Rusu. Monte Carlo pose estimation with quaternion kernels and the bingham distribution. In *Proc. of RSS*, 2011.

[55] J. M. Gonfaus, X. Boix, J. Van De Weijer, A. D. Bagdanov, J. Serrat, and J. Gonzalez. Harmony potentials for joint classification and segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3280–3287. IEEE, 2010.

[56] C. Gu and X. Ren. Discriminative Mixture-of-Templates for Viewpoint Classification. In *Proc. of European Conference on Computer Vision (ECCV)*, pages 408–421, 2010.

[57] G.D. Hager and B. Wegbreit. Scene parsing using a prior world model. *IJRR*, 2011. To appear.

[58] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[59] J. Hays and A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 26(3), 2007.

[60] R. He, B.G. Hu, W.S. Zheng, and Y.Q. Guo. Two-Stage Sparse Representation for Robust Recognition on Large-Scale Database. In *AAAI-10*, 2010.

[61] S. Helmer and D. G. Lowe. Using stereo for object recognition. In *Proc. of ICRA*, pages 3121–3127, 2010.

[62] P. Henry, D. Fox, A. Bhowmik, and R. Mongia. Patch Volumes: Segmentation-based Consistent Mapping with RGB-D Cameras. In *International Conference on 3D Vision (3DV)*, 2013.

[63] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D Mapping: Using depth cameras for dense 3D modeling of indoor environments. In *the 12th International Symposium on Experimental Robotics (ISER)*, December 2010.

[64] S. Hinterstoisser, V. Lepetit, S. Ilic, P. Fua, and N. Navab. Dominant orientation templates for real-time detection of texture-less objects. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2257–2264. IEEE, 2010.

[65] A. Howard and N. Roy. The robotics data set repository, 2003.

[66] H. Hu, D. Munoz, J. A. Bagnell, and M. Hebert. Efficient 3-d scene analysis from streaming data. In *ICRA*, May 2013.

[67] R. Hwa. Supervised grammar induction using training data with limited constituent information. In *In Proceedings of the 37th Annual Meeting of the ACL*, pages 73–79, 1999.

[68] A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell. A category-level 3d object dataset: Putting the kinect to work. In *Consumer Depth Cameras for Computer Vision*, pages 141–165. Springer, 2013.

[69] J. Jiang and C. Zhai. Instance weighting for domain adaptation in NLP. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2007.

[70] A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 21(5), 1999.

[71] P. Kaewtrakulpong and R. Bowden. An improved adaptive background mixture model for realtime tracking with shadow detection. In *European Workshop on Advanced Video Based Surveillance Systems*, 2001.

[72] S. Kane, D. Avrahami, J. Wobbrock, B. Harrison, A. Rea, M. Philipose, and A. LaMarca. Bonfire: A nomadic system for hybrid laptop-tabletop interaction. In *Proc. of UIST*, 2009.

[73] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 156–164. Eurographics Association, 2003.

[74] S. S. Keerthi, O. Chapelle, and D. DeCoste. Building support vector machines with reduced classifier complexity. *J. Mach. Learn. Res.*, 7:1493–1515, 2006.

[75] Microsoft Kinect. `http://www.xbox.com/en-us/kinect`.

[76] U. Klank, M.Z. Zia, and M. Beetz. 3d model selection from an internet database for robotic vision. In *Proc. of ICRA*, pages 2406–2411, 2009.

[77] M. Körtgen, G. . Park, M. Novotni, and R. Klein. 3d shape matching with 3d shape contexts. In *The 7th Central European Seminar on Computer Graphics*, April 2003.

[78] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012.

[79] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[80] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *ICRA*, 2011.

[81] K. Lai, L. Bo, X. Ren, and D. Fox. A scalable tree-based approach for joint object and pose recognition. In *Twenty-Fifth Conference on Artificial Intelligence (AAAI)*, August 2011.

[82] K. Lai, L. Bo, X. Ren, and D. Fox. Sparse distance learning for object recognition combining rgb and depth information. In *ICRA*, 2011.

[83] K. Lai, L. Bo, X. Ren, and D. Fox. Detection-based object labeling in 3d scenes. In *ICRA*, 2012.

[84] K. Lai and D. Fox. 3d laser scan classification using web data and domain adaptation. In *Robotics: Science and Systems (RSS)*, 2009.

[85] K. Lai and D. Fox. Object Recognition in 3D Point Clouds Using Web Data and Domain Adaptation. *The International Journal of Robotics Research*, 2010.

[86] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proc. of CVPR*, volume 2, pages 2169–2178, 2006.

[87] H. Lee, R. Grosse, R. Ranganath, and A. Ng. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations. In *ICML*, 2009.

[88] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *Int. J. Comput. Vision*, 43(1):29–44, June 2001.

[89] L-J Li, R. Socher, and L. Fei-Fei. Towards total scene understanding: Classification, annotation and segmentation in an automatic framework. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2036–2043. IEEE, 2009.

[90] L.-J. Li, G. Wang, and L. Fei-Fei. Optimol: automatic object picture collection via incremental model learning. In *IEEE Computer Vision and Pattern Recognition (CVPR).*, 2007.

[91] B. Liu, S. Gould, and D. Koller. Single image depth estimation from predicted semantic labels. In *Proc. of CVPR*, 2010.

[92] D. G. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV)*, 1999.

[93] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[94] T. Malisiewicz and A. Efros. Improving spatial support for objects via multiple segmentations. In *Proc. of the British Machine Vision Conference*, 2007.

[95] T. Malisiewicz and A. Efros. Recognition by association via learning per-examplar distances. In *Proc. of CVPR*, 2008.

[96] M. Martinez, A. Collet, and S.S. Srinivasa. MOPED: A scalable and low latency object recognition and pose estimation system. In *Proc. of ICRA*, pages 2043–2049, 2010.

[97] D. Meger and J. J. Little. Mobile 3d object detection in clutter. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4885–4892. IEEE, 2011.

[98] D. Meger, C. Wojek, J. J. Little, and B. Schiele. Explicit occlusion reasoning for 3d object detection. In *BMVC*, 2011.

[99] L. Meier, S. Van De Geer, and P. Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

[100] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISSAPP*, pages 331–340. INSTICC Press, 2009.

[101] M. Muja, R. B. Rusu, G. Bradski, and D. G. Lowe. Rein-a fast, robust, scalable recognition infrastructure. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2939–2946. IEEE, 2011.

[102] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136, 2011.

[103] Ma. Novotni and R. Klein. 3d zernike descriptors for content based shape retrieval. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 216–225, New York, NY, USA, 2003. ACM.

[104] A. Nüchter, H. Surmann, and J. Hertzberg. Automatic classification of objects in 3d laser range scans. In *In Proc. 8th Conf. on Intelligent Autonomous Systems, pages 963 970*, pages 963–970. IOS Press, 2004.

[105] Y. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition. In *The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, pages 40–44, 1993.

[106] J. Platt. Probabilistic outputs for support vectormachines and comparisons to regularized likelihood methods. In *Advances in large margin classifiers*. Cambridge: MIT Press, 1999.

[107] I. Posner, M. Cummins, and P. Newman. Fast probabilistic labeling of city maps. In *Proc. of RSS*, 2008.

[108] PrimeSense. `http://www.primesense.com/`.

[109] M. Quigley, S. Batra, S. Gould, E. Klingbeil, Q. Le, A. Wellman, and A.Y. Ng. High-accuracy 3d sensing for mobile manipulation: Improving object detection and door opening. In *Proc. of ICRA*, 2009.

[110] X. Ren and D. Ramanan. Histograms of sparse codes for object detection. In *CVPR*, June 2013.

[111] B. Roark and M. Bacchiani. Supervised and unsupervised pcfg adaptation to novel domains. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 126–133, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[112] B. Russell, A. Torralba, K. Murphy, and W. Freeman. Labelme: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1-3), 2008.

[113] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2155–2162. IEEE, 2010.

[114] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *Computer Vision–ECCV 2010*, pages 213–226. Springer, 2010.

[115] R. Salakhutdinov, A. Torralba, and J. Tenenbaum. Learning to share visual appearance for multiclass object detection. In *Proc. of CVPR*, 2011.

[116] B. Sapp, A. Saxena, and A. Ng. A fast data collection and augmentation procedure for object recognition. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2008.

[117] S. Savarese and Li Fei-Fei. 3d generic object categorization, localization and pose estimation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, 2007.

[118] A. Saxena, J. Driemeyer, and A. Ng. Robotic grasping of novel objects using vision. *IJRR*, 27(2), 2008.

[119] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE PAMI*, 31(5):824–840, 2009.

[120] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[121] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *Advances in Neural Information Processing Systems (NIPS)*, page 41. The MIT Press, 2003.

[122] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated subgradient solver for SVM. In *International Conference on Machine Learning (ICML)*, pages 807–814, 2007.

[123] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8), 2000.

[124] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. The princeton shape benchmark. In *Shape Modeling International*, 2004.

[125] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[126] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *Computer Vision–ECCV 2012*, pages 746–760. Springer, 2012.

[127] R. Socher, C. C. Lin, A. Ng, and C. Manning. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, pages 129–136, 2011.

[128] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1531–1565, 2006.

[129] M. Stark, J. Krause, B. Pepik, D. Meger, J. J. Little, B. Schiele, and D. Koller. Fine-grained categorization for 3d scene understanding. *International Journal of Robotics Research*, 30(13):1543–1552, 2011.

[130] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. Narf: 3d range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 44, 2010.

[131] Y. Sun, L. Bo, and D. Fox. Attribute Based Object Identification. In *ICRA*, 2013.

[132] A. Teichman and S. Thrun. Tracking-based semi-supervised learning. *The International Journal of Robotics Research*, 31(7):804–818, 2012.

[133] A. Torralba and A. A. Efros. Unbiased look at dataset bias. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1521–1528. IEEE, 2011.

[134] A. Torralba, R. Fergus, and W. Freeman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(11), 2008.

[135] R. Triebel, R. Schmidt, O. Martinez Mozos, and W. Burgard. Instance-based amn classification for improved object recognition in 2d and 3d laser range data. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[136] Trimble. 3d warehouse. http://sketchup.google.com/3dwarehouse/, 2008.

[137] C.J. van Rijsbergen. *Information Retrieval, 2nd ed.* Butterworths, London, 1979.

[138] C. J. Veenman and M. J. T. Reinders. The nearest subclass classifier: A compromise between the nearest mean and nearest neighbor classifier. *IEEE Trans on Pattern Analysis and Machine Intelligence*, 27:1417–1429, 2005.

[139] C. Vondrick, D. Ramanan, and D. Patterson. Efficiently scaling up video annotation with crowdsourced marketplaces. In *European Conference on Computer Vision (ECCV)*, 2010.

[140] K.Q. Weinberger and L.K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research (JMLR)*, 10:207–244, 2009.

[141] C. Wellington, A. Courville, and T. Stentz. Interacting Markov random fields for simultaneous terrain modeling and obstacle detection. In *Proc. of RSS*, 2005.

[142] W. Wohlkinger, A. Aldoma, R. B. Rusu, and M. Vincze. 3dnet: Large-scale object class recognition from cad models. In *ICRA*, pages 5384–5391, 2012.

[143] K. M. Wurm, H. Kretzschmar, R. Kümmerle, C. Stachniss, and W. Burgard. Identifying vegetation from laser data in structured outdoor environments. *Robotics and Autonomous Systems*, 2012.

[144] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning with application to clustering with side-information. *Advances in Neural Information Processing Systems (NIPS)*, pages 521–528, 2003.

[145] X. Xiong, D. Munoz, J. Bagnell, and M. Hebert. 3-D scene analysis via sequenced predictions over points and regions. In *Proc. of ICRA*, 2011.

[146] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear Spatial Pyramid Matching using Sparse Coding for Image Classification. In *CVPR*, 2009.

[147] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49, 2006.

[148] R. Ziola, S. Grampurohit, N. Landes, J. Fogarty, and B. Harrison. Examining Interaction with General-Purpose Object Recognition in OASIS. In *University of Washington Technical Report UW-CSE-11-05-01*, 2011.

VITA

Kevin Kar Wai Lai obtained his Bachelor of Science degree in Computer Science at the University of British Columbia in 2008. He obtained his M.S. and Ph.D. degrees in Computer Science at the University of Washington in 2010 and 2013 respectively. Kevin has published a number of papers in refereed conferences and journals, including "Sparse Distance Learning for Object Recognition Combining RGB and Depth Information," which won the Best Vision Paper award at the IEEE International Conference on Robotics and Automation (ICRA) in 2011. He is also a recipient of the doctoral-level Postgraduate Scholarship from the Natural Sciences and Engineering Research Council (NSERC) of Canada.