Representations for Visually Guided Actions

Saurabh Gupta

Representations for Visually Guided Actions

by

Saurabh Gupta

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jitendra Malik, Chair
Professor Trevor Darrell
Professor Jack L. Gallant

Summer 2018

Representations for Visually Guided Actions

Abstract

Representations for Visually Guided Actions

by

Saurabh Gupta

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Jitendra Malik, Chair

In recent times, computer vision has made great leaps towards 2D understanding of sparse visual snapshots of the world. This is insufficient for robots that need to exist and act in the 3D world around them based on a continuous stream of multi-modal inputs. In this work, we present efforts towards bridging this gap between computer vision and robotics. We show how thinking about computer vision and robotics together brings out limitations of current computer vision tasks and techniques, and motivates joint study of perception and action. We present some initial efforts towards this and investigate a) how we can move from 2D understanding of images to 3D understanding of the underlying scene, b) how recent advances in representation learning for images can be extended to obtain representations for varied sensing modalities useful in robotics, and c) how thinking about vision and action together can lead to more effective solutions for the classical problem of visual navigation.

To Baba and Daadi

Contents

Acknowledgments

I am extremely lucky to have had Jitendra as my advisor. As I started my PhD, this didn't immediately occur to me. But over the last seven years, with every passing milestone in graduate school, I have realized how awesome an advisor, and even more, how amazing a person he is. Jitendra has taught me how to be friends with every pixel as well as how to look at the big picture. Jitendra's energy, enthusiasm and passion for science is infectious. Numerous moments of self-doubt and dejections from paper rejections, were cured by Jitendra's optimism and positivity. Even seven years weren't enough to learn everything from Jitendra. I will consider myself lucky, if over time I am able to know as much as he does, give as evocative a talk or come up with as pithy a title. Jitendra, thanks for taking me on as a student and grooming me as a researcher.

Being at Berkeley also gave me the opportunity to learn through close interactions with a number of leaders in the field. Paper reading groups with Alyosha, Trevor and Jitendra exposed me to different points of view and enabled me to develop a holistic taste. Alyosha's comments (though almost always about nearest neighbors and the meaninglessness of categories) and Trevor's succinct remarks were deeply profound and thought-provoking. Though I never directly worked with Alyosha, informal interactions with him have shaped my thoughts in a big way and I am really glad that he came to Berkeley. I did have the fortune of directly collaborating with Trevor and Sergey, and their ideas and style of thinking provided a much needed counter-weight to Jitendra's views. I also got a chance to interact with Pieter through his robotics class as well as when I served as a GSI for his AI class. Pieter's dedication to teaching and research is truly inspiring. Also, a special thanks to Trevor, Jack, Ruzena, and Pieter for taking out time to serve on various committees through my PhD, and asking hard questions and providing insightful feedback.

It would have been impossible to get started in research, if it were not for Pablo and Ross, who hand-held me through my first projects. Pablo and Ross were extremely patient and encouraging in training me. They taught me the basics of science: how to design experiments, how to conduct controls and ablations, how to pick strong baselines, and above all how to maintain the highest scientific standards. Their selfless devotion to science and research will always inspire me.

I also had the fortune of spending time at Microsoft and Google research labs, and learned a lot from everyone I met there: Piotr Dollár, Larry Zitnick, John Platt, Meg Mitchell, Rahul Sukthankar and James Davidson. Thank you for helping me broaden my horizons.

I will also like to thank Manik Varma and Prateek Jain for nurturing me as a researcher early on as an undergraduate, and instilling in me the curiosity to pursue a PhD.

Graduate school would have been hard to survive without fellow graduate students and post-docs. Bharath, thanks for explaining the hardest papers, for your suggestions on improving the experiments, and for keeping the lab atmosphere as relaxed as it was. Georgia, thanks for always questioning why, for helping distill ideas to their essence, and for showing how to balance research and life. Shubham, thanks for your insightful and constructive criticism, for finding the fatal flaw before an experiment was run, and for lending laptops in dire circumstances. And thanks Varun and Ashish for your enthusiasm and energy that kept me going during my elongated senioritis.

I can't thank enough all the amazing people I met in the vision group at Berkeley: Chunhui, Jeannette, Jon, Pulkit, Abhishek, Panna, Somil, Philipp, João, Katerina, Weicheng, Ke, Andrew, Christian, David, Angjoo, Ashish, Zhe, Allan, Jasmine, Judy, Jeff, Ning, Sergey K, Yangqing, Evan, Lisa, Deepak, Sergio, Richard, Tinghui, Jun-Yan, Shiry, Phillip, and all the people in the BAIR community.

And a special thanks to Angie Abbatecola, for organizing all the fun activities (BAIR retreats, happy hours and conference parties) and greatly simplifying all the paperwork.

Life over the last seven years, would not have been nearly as much fun had it not been for friends outside of work: Varun Jog, Nitesh, Shashank, Nikunj, Sreeta, TD, Moitrayee, Anurag, Shromona, Gautam, Neeraja, Vivek, Aishwarya, Rishabh, Amin, Aida, Kush.

The biggest thanks goes to my family. Thanks for always sending me off with smiles and laughter, and for supporting me in all my pursuits, whether reasonable or unreasonable. I was also fortunate to have some of you in the bay area: Gaurav, Neha, Divya and Rahul. While nothing beats being at home, it was extremely comforting to have you all around. Thanks for all the fun times we had over the last years.

And finally to Radhika, for everything.

Chapter 1

Introduction

Computer vision started out as a field to enable robots to understand the world. Early work in computer vision was readily applicable to robotic applications, be it modeling images with 3D CAD models [134], or inferring the 3D structure of the world given multiple images [85, 122]. Not only this, but the first book in computer vision by Berthold Horn was in fact called *Robot Vision* [84].

But in the 90s, computer vision researchers branched out and the focus was on problems that could enable computer graphics applications [148]. Researchers designed techniques to stitch images together into panoramas [166] and developed algorithms to render people and places from novel viewpoints [149, 30].

And in the 2000s, computer vision found yet another buddy, this time in machine learning and big data. Computer vision labs moved from stocking cameras to stocking hard-disks, so that they could download big data from the Internet. Computer vision at this point has transformed into something that is call *Internet Computer Vision*, where the dominant research paradigm is to download and label large-scale image datasets over the Internet, analyze them using sophisticated machine learning techniques to study image labeling problems. And this has been a successful endeavor, leading up to algorithms such as Mask R-CNN [73] that can label complex real world images with objects that are present in them.

This naturally presents us with the question: *is the current paradigm of Internet Computer Vision sufficient for Berthold Horn's robot to successfully perceive the world?*

To answer this question, let us examine what Internet Computer Vision can do for us. Given an input image such as in Figure 1.1, Internet Computer Vision can label it with the objects in the image. It can also tell us where they are, and what are the precise pixels that belong to the different objects. While this is great, unfortunately, all of this understanding only exists in the 2D image space. It tells us nothing about the 3D structure of the world. For example, it does not tell us that the road extends behind these objects. It does not answer, if we can walk into the scene or not. It does not even tell us, where would these objects be if we were to observe the scene from a slightly different viewpoint. These are just some of the questions about the world, that a robot will need to have answers of, to successfully act in the world.

Moreover, Internet Computer Vision is limited to understanding the world through RGB images

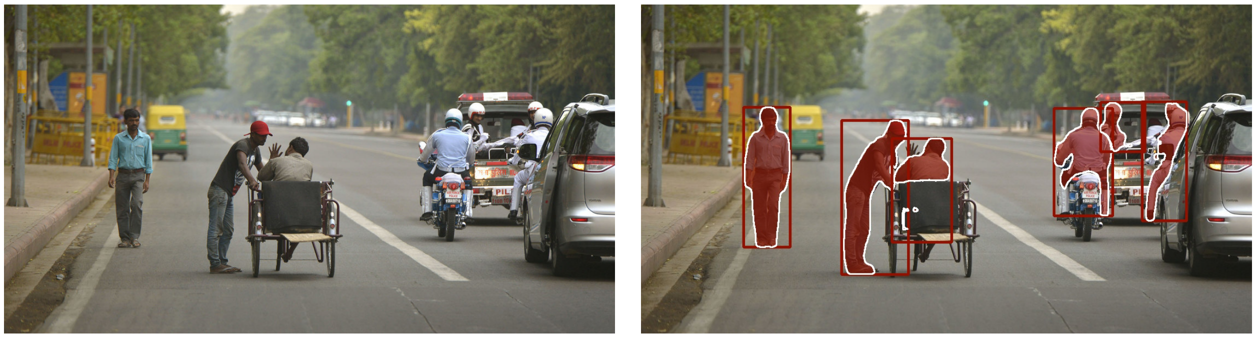


Figure 1.1: Understanding an image, so as to take actions in the world, goes beyond merely detecting and delineating objects of interest in the 2D image space. Image source: https://www.flickr.com/photos/secdef/17790319373.

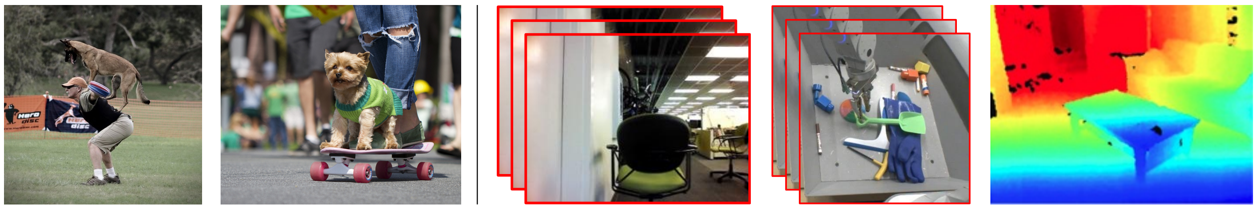


Figure 1.2: Internet Computer Vision *vs.* Robot Vision. Left shows sample images from the MS COCO dataset, while right shows images in context of robotics. Image credits: Lin *et al.* [119], Finn *et al.* [43] and Silberman *et al.* [156].

uploaded by people onto the Internet. For example, Figure 1.2 (left) visualizes images from the MS-COCO dataset [119], a popular dataset in computer vision (and in fact the one that was used to design and develop currently popular R-CNN line of object detectors). A day in the life of these object detectors thus comprises of anal zing such interesting and photogenic images from Flickr. On the other hand, Figure 1.2 (right) shows a typical day in the life of a robot: a mobile robot that is moving around in indoor office environments and an armed robot that is learning to grasp objects in clutter. Not only do these images look different from images on the Internet, these are in fact not images but continuous streams of data. Streams of data that are determined by the robot itself through its actions.

Furthermore, because we determine what sensors we put onto our robots, robots are not constrained to learn about the world only through RGB image observations. They can sense the world through a plethora of different sensing modalities, such as through GelSight images from a haptic sensor, or through point cloud data from a LiDAR scanner.

Thus, the current paradigm of Internet Computer Vision only tackles the narrow problem of 2D understanding of images on the Internet, which forms only a small part of the perception problem faced by a robot. In this work, we focus on bridging this gap between perception and action. We investigate how to build actionable representations of the unstructured real world from raw sensory inputs from different modalities. We demonstrate the utility of these representations for *perceiving the world in 3D* and for *acting in the 3D world to move around*.

This thesis starts by studying the problem of 3D scene understanding in context of range sensors (such as Microsoft Kinect). Given observations from such sensors, we first study central computer vision problems that of perceptual organization and recognition. We then go beyond these standard tasks, that only make inferences about what is explicitly visible, to make predictions for parts of the scene that aren't directly visible. We do this for scene surfaces (via reasoning about their amodal extent behind objects) and objects (by corresponding them with full 3D CAD models). This work was presented in [58, 64, 63, 60], and is summarized in Chapter 2.

We then study how such 3D scene understanding can be used in context of the robotic tasks of navigation. Most existing work in this area falls in one of two paradigms: classical robotics and learning based robotics. Models in classical robotics use explicit hand-crafted geometric representations that capture the problem structure but ignore semantics and can't be learned from data. In contrast, models in learning based robotics can learn good representations in context of the end task, but ignore the structure present in the problem. Thus, results have mostly been confined to simple tasks often in 2D worlds, and extremely specialized behavior that does not generalize to new environments. To address this divide we formulate a new paradigm, that instantiates insights about problem structure (from classical methods) into learning formalisms (from learning-based robotics) through use of specialized end-to-end trainable policy architectures that jointly map and plan. Thus, our resulting policies not only benefit from learned task-driven representations, but also leverage the problem structure appropriately, leading to effective performance in novel environments. This work was presented in [61] and is summarized in Chapter 3.

While working on these problems, we saw improvements in performance when using information from additional sensing modalities such as range scans. While, on the one hand, using custom sensors simplified the task, the dominant paradigm for learning representation relies on supervised training using large amounts of labeled data. This reliance on labeled data for learning representations severely limits the information that can be derived from additional sensors. This led to the study of how supervision could be transferred from well-labeled modalities (such as Internet images) to such impoverished modalities. The was presented in [59] and is summarized in Chapter 4.

Chapter 2

3D Scene Understanding

Truly understanding a scene involves reasoning not just about what is visible but also about what is not visible. Consider for example the images in Figure 2.1. After we recognize an object as a chair, we have a pretty good sense of how far it extends in depth and what it might look like from another viewpoint. One way of achieving this kind of understanding in a computer vision system would be by 'replacing in-place' the chair pixels by the rendering of a 3D CAD model of the chair. This explicit correspondence to a 3D CAD model leads to a richer representation than output from traditional computer vision algorithms like object detection, semantic or instance segmentation, fine-grained categorization and pose estimation. Each of these tasks by themselves is insufficient from a robotics perspective for tasks like trajectory optimization, motion planning or grasp estimation. Our proposed system starts from a single RGB-D image of a cluttered indoor scene and produces the output visualized in Figure 2.1. Our approach is able to successfully retrieve and align relevant models with objects in real-world cluttered scenes. We believe our rich output representation will facilitate use of perception in robotics.

Figure 2.2 shows an overview of our approach. We approach the problem by first studying bottom-up grouping, where we combine inputs from RGB and depth images to get improved RGB-D contours. We then use these RGB-D contours to obtain bottom-up candidates for objects. We then study the problem of feature learning for RGB-D images using convolutional neural networks and classify the obtained bottom-up candidates to detect objects. We then analyze object detections and reason about their pixel support, pose and sub-type, to obtain candidate 3D models which are aligned and placed in the virtual scene. This work can also be seen as unifying three independent strands of research in computer vision: perceptual grouping (chunking raw pixels in an image into discrete entities), recognition (associating semantics with these groups of pixels), and 3D shape inference (inferring the 3D shape of these objects). In the following sections, we describe our approaches to each of these problems.

This chapter is based on work done with Pablo Arbeláez, Ross Girshick and Jitendra Malik, and is presented here primarily as it appeared in CVPR 2013 [58], ECCV 2014 [64], IJCV 2015 [63], and CVPR 2015 [60]. Statements about past work should be read with this context in mind.

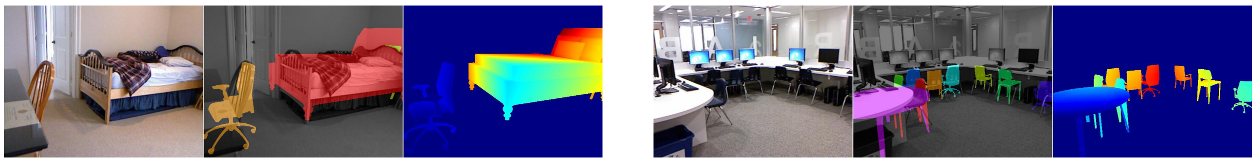


Figure 2.1: Output of our system: Starting from an RGB-D image, we produce a 3D scene where objects have been replaced by corresponding 3D models.

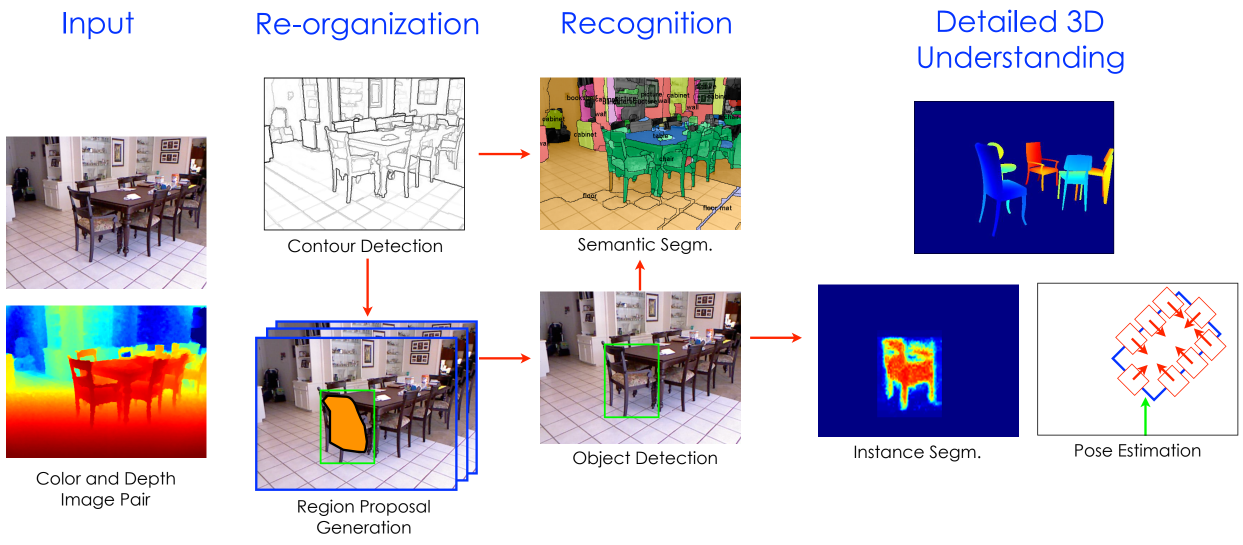


Figure 2.2: Overview: From an RGB and depth image pair, our system detects contours, generates 2.5D region proposals, classifies them into object categories, and then infers segmentation masks for instances of "thing"-like objects, as well as labels for pixels belonging to "stuff"-like categories. We then analyze the detected objects and reason about their pixel support, and pose to finally replace it with a corresponding 3D model to obtain a "virtual" scene.

2.1 Background

A large body of work in computer vision has focused on the problem of object detection, where the final output is a bounding box around the object, [27, 42, 49, 137, 175]. There has also been substantial work on labeling each pixel in the image with a semantic label *e.g.* [7, 21]. Recent work from Hariharan *et al.* [70], Tighe *et al.* [170] brings these two lines of research together by inferring the pixel support of object instances.

There have been corresponding works for RGB-D images studying the problems of object detection [17, 90, 108, 109, 110, 118, 163, 97, 168], semantic segmentation [13, 104, 133, 156, 155], and more recently instance segmentation [155]. Since our approach builds on an object detection system, we discuss this body of research in more detail. Modifications to deformable part models [42] for RGB-D images were proposed in [90, 97, 168]. [102, 118] operate in a similar paradigm of reasoning with bottom-up region proposals, but focus on modeling object-

object, object-scene, and image-text relationships.

We note that, although all of these outputs are useful representations, each of them is far from an understanding of the world that would enable a robot to interact with it.

We are of course not the first ones to raise this argument. There is a lot of research on 3D scene understanding from a single RGB image [75, 141], and 3D object analysis [9, 77, 116, 147, 184]. Given the challenging nature of the problem, most of these works are restricted to unoccluded clean instances and fail under clutter. In this paper, we study the problem in the context of the challenging NYUD2 dataset and analyze how RGB-D data can be effectively leveraged for this task.

The most relevant research to our work comes from Song and Xiao [163] and Guo and Hoiem [56]. Song and Xiao [163] reason in 3D, train exemplar SVMs using synthetic data, and slide these exemplars in 3D space to search for objects, thus naturally dealing with occlusion. Their approach is inspiring, but computationally expensive (25 minutes per image per category). They also show examples where their model is able to place a good fitting exemplar to data, but they do not address the problem of estimating good 3D models that fit the data. We differ from their philosophy and propose to reason on the problem in 2D to effectively prune large parts of the search space, and then do detailed 3D reasoning with the top few winning candidates. As a result, our final system is significantly faster (taking about two minutes per image). We also show that lifting from a 2D representation to a 3D representation is possible and show that naively fitting a box around the detected region outperforms the model from [163].

Guo and Hoeim [56] start with a bottom-up segmentation, retrieve nearest neighbors from the training set, and align the retrieved candidate with the data. In contrast, we use category knowledge in the form of top-down object detectors and inform the search procedure about the orientation of the object. Moreover, our algorithm does not rely on detailed annotations (which take about 5 minutes for each scene) [57] of the form used in [56]. We also propose a category-level metric to evaluate the rich and detailed output from such algorithms.

Finally, [140, 152], among many others, study the same problem but either consider known instances of objects, or rely on user interaction.

2.2 RGB-D Reorganization

2.2.1 Contour Detection

Contour detection in images crucially depends on designing good local gradients that can capture discontinuities of different kinds. For detecting contours in RGB-D images, we adopt standard local gradients used for RGB images and design local gradients on depth images that capture different kinds of shape discontinuities.

2.2.1.1 Local Gradients on Depth Images

We estimate three oriented contour signals at each pixel in the image: a depth gradient DG that identifies the presence of a discontinuity in depth, a convex normal gradient NG_+ that captures if the surface bends-out at a given point in a given direction, and a concave normal gradient NG_-, that captures if the surface bends-in.

Computing these gradients on RGB-D images is not trivial because of the characteristics of the data, particularly: (1) a nonlinear noise model of the form $|\delta Z| \propto Z^2|\delta d|$, where δZ is the error in depth observation, Z is the actual depth, δd is the error in disparity observation (due to the triangulation-based nature of the Kinect), causing non-stochastic and systematic quantization of the depth, (2) lack of temporal synchronization between color and depth channels, and (3) missing depth observations. We address these issues by carefully designing geometric contour cues that have a clear physical interpretation, using multiple sizes for the window of analysis, not interpolating for missing depth information, estimating surface normals by least square fits to disparity instead of points in the point cloud, and independently smoothing the orientation channels with Savitsky-Golay [143] parabolic fitting.

In order to estimate the local geometric contour cues, we consider a disk centered at each image location. We split the disk into two halves at a pre-defined orientation and compare the information in the two disk-halves, as suggested originally in [123] for contour detection in monocular images. In the experiments, we consider 4 different disk radii varying from 5 to 20 pixels and 8 orientations. We compute the 3 local geometric gradients DG, NG_+ and NG_- by examining the point cloud in the 2 oriented half-disks. We first represent the distribution of points on each half-disk with a planar model. Then, for DG we calculate the distance between the two planes at the disk center and for NG_+ and NG_- we calculate the angle between the surface normals of the planes.

These local gradients on depth images, can then be used with any standard contour detection framework. We investigate use of these local gradients with *gPb-ucm* technique from Arbeláez *et al.* [6] as well as with the structured forest approach from Dollár *et al.* [35][1]. In both these cases, we see significant improvements in performance for contour detection over the respective baseline. This also resulted in state-of-the-art performance for this task at the time this research was conducted. A number of works that use high-level semantic cues (via convolutional neural network models) have since out-performed our approach.

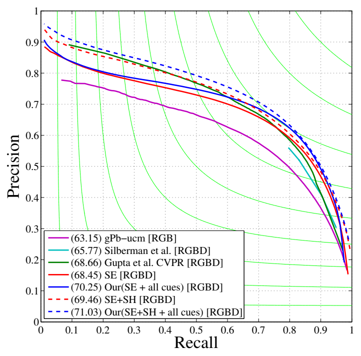
2.2.2 2.5D Region Proposals

2.2.2.1 Candidate Ranking

From the obtained contour signal, we obtain object proposals by generalizing MCG to RGB-D images. MCG for RGB images [5] uses simple features based on the color image and the region shape to train a random forest regressors to rank the object proposals. We follow the same paradigm, but propose additional geometric features computed on the depth image within each proposal. We compute: (1) the mean and standard deviation of the disparity, height above ground, angle with gravity, and world (X, Y, Z) coordinates of the points in the region; (2) the region's (X, Y, Z) extent; (3) the region's minimum and maximum height above ground; (4) the fraction of pixels on vertical surfaces, surfaces facing up, and surfaces facing down; (5) the minimum and maximum

[1]For use with [35], we additionally use geocentric pose (per-pixel height above ground and angle with gravity) and richer appearance cues in the form of contour predictions of a contour detector trained on the BSDS dataset.

Table 2.1: Contour Detection Performance. Left plot shows precision-recall plot for boundary detection on the NYUD2 [156] dataset. Right table reports standard evaluation metrics for this task for different methods (all numbers are percentages).



		ODS ($F_{\max}$)	OIS ($F_{\max}$)	AP
gPb-ucm [6]	RGB	63.15	66.12	56.20
Silberman *et al.* [156]	RGB-D	65.77	66.06	-
Our (gPb-ucm + $DG, NG_{\{+,-\}}$) [58]	RGB-D	68.66	71.57	62.91
SE [35]	RGB-D	68.45	69.92	67.93
Our(SE + normal gradients) [64]	RGB-D	69.55	70.89	69.32
Our(SE + all cues) [64]	RGB-D	70.25	71.5	69.28
SE+SH [34]	RGB-D	69.46	70.84	71.88
Our(SE+SH + all cues) [64]	RGB-D	71.03	72.33	73.81

standard deviation along a direction in the top view of the room. We obtain 29 geometric features for each region in addition to the 14 from the 2D region shape and color image already computed in [5]. Note that the computation of these features for a region decomposes over superpixels and can be done efficiently by first computing the first and second order moments on the superpixels and then combining them appropriately.

2.2.3 Results

We now present results for contour detection and candidate ranking. We work with the NYUD2 dataset and use the standard split of 795 training images and 654 testing images (we further divide the 795 images into a training set of 381 images and a validation set of 414 images). These splits are carefully selected such that images from the same scene are only in one of these sets.

Contour Detection: To measure performance on the contour detection task, we plot the precision-recall curve on contours in Table 2.1 (left) and report the standard maximum F-measure metric ($F_{\max}$) in Table 2.1 (right). We start by reporting the performance when our proposed cues are used with gPb-ucm [6]. We see large improvements in performance over gPB-ucm when additionally using cues computed from depth images. This already performs comparably to the stronger approach from Dollár *et al.* [35] that also uses RGB-D images as input.

Next, we report the performance when out proposed cues are used with the structured forest approach from Dollár *et al.* [35]. We observe that adding normal gradients consistently improves precision for all recall levels and $F_{\max}$ increases by 1.2% points (Table 2.1 (right)). The addition of geocentric pose features and appearance features improves $F_{\max}$ by another 0.6% points, making our final system better than the current state-of-the-art methods by 1.5% points.[2]

2.5D Region Proposals: The goal of the region generation step is to propose a pool of candidates for downstream processing (*e.g.*, object detection and segmentation). Thus, we look at the standard

[2]Dollár *et al.* [34] recently introduced an extension of their algorithm and report performance improvements (SE+SH[RGB-D] dashed red curve in Table 2.1). We can also use our cues with [34], and observe an analogous improvement in performance (Our(SE+SH + all cues) [RGB-D] dashed blue curve in Table 2.1).

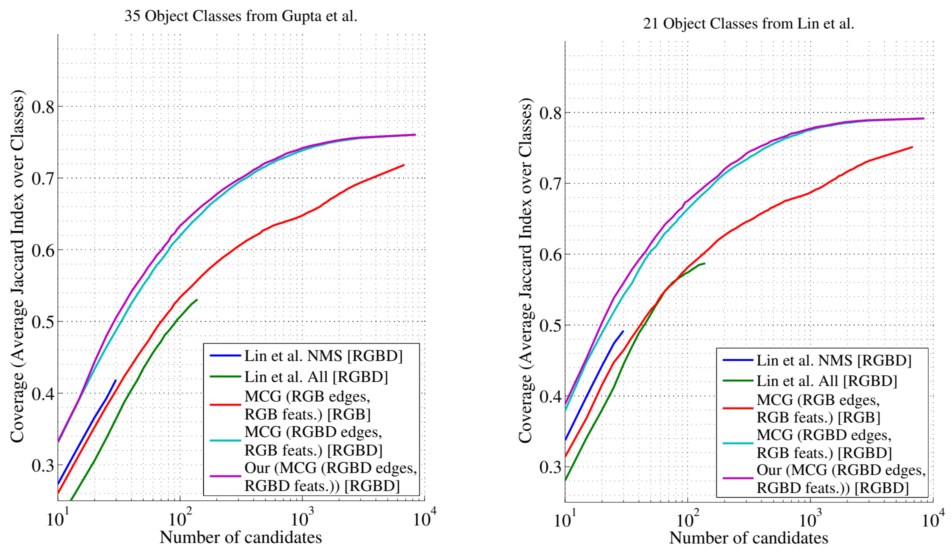


Figure 2.3: Region Proposal Quality: Coverage as a function of the number of region proposal per image for 2 sets of categories: ones which we study in this paper, and the ones studied by Lin *et al.* [118]. Our depth based region proposals using our improved RGB-D contours work better than Lin *et al.*'s [118], while at the same time being more general. Note that the X-axis is on a log scale.

metric of measuring the coverage of ground truth regions as a function of the number of region proposals. Since we are generating region proposals for the task of object detection, where each class is equally important, we measure coverage for K region candidates by

$$\text{coverage}(K) \quad = \quad \frac{1}{C}\sum_{i=1}^{C}\left(\frac{1}{N_i}\left(\sum_{j=1}^{N_i}\max_{k\in[1\ldots K]} O\left(R_k^{l(i,j)}, I_j^i\right)\right)\right), \tag{2.1}$$

where C is the number of classes, N_i is the number of instances for class i, $O(a,b)$ is the intersection over union between regions a and b, I_j^i is the region corresponding to the j^{th} instance of class i, $l\,(i,j)$ is the image which contains the j^{th} instance of class i, and R_k^l is the k^{th} ranked region in image l.

We plot the function coverage(K) in Figure 2.3 (left) for our final method, which uses our RGB-D contour detector and RGB-D features for region ranking (black). As baselines, we show regions from the recent work of Lin *et al.* [118] with and without non-maximum suppression, MCG with RGB contours and RGB features, MCG with RGB-D contours but RGB features and finally our system which is MCG with RGB-D contours and RGB-D features. We note that there is a large improvement in region quality when switching from RGB contours to RGB-D contours, and a small but consistent improvement from adding our proposed depth features for candidate region re-ranking.

Since Lin *et al.* worked with a different set of categories, we also compare on the subset used in their work (in Figure 2.3 (right)). Their method was trained specifically to return candidates for these classes. Our method, in contrast, is trained to return candidates for generic objects and therefore “wastes” candidates trying to cover categories that do not contribute to performance on any fixed subset. Nevertheless, our method consistently outperforms [118], which highlights the effectiveness and generality of our region proposals.

2.3 RGB-D Recognition

2.3.1 Object Detection

We generalize the R-CNN system introduced by Girshick *et al.* [49] to leverage depth information. At test time, R-CNN starts with a set of bounding box proposals from an image, computes features on each proposal using a convolutional neural network, and classifies each proposal as being the target object class or not with a linear SVM. The CNN is trained in two stages: first, pretraining it on a large set of labeled images with an image classification objective, and then finetuning it on a much smaller detection dataset with a detection objective.

We generalize R-CNN to RGB-D images and explore the scientific question: Can we learn rich representations from depth images in a manner similar to those that have been proposed and demonstrated to work well for RGB images?

2.3.1.1 Encoding Depth Images for Feature Learning

Given a depth image, how should it be encoded for use in a CNN? Should the CNN work directly on the raw depth map or are there transformations of the input that the CNN to learn from more effectively?

We propose to encode the depth image with three channels at each pixel: horizontal disparity, height above ground, and the angle the pixel’s local surface normal makes with the inferred gravity direction. We refer to this encoding as HHA. The latter two channels are computed using the algorithms proposed in [58] and all channels are linearly scaled to map observed values across the training dataset to the 0 to 255 range.

The HHA representation encodes properties of geocentric pose that emphasize complementary discontinuities in the image (depth, surface normal and height). Furthermore, it is unlikely that a CNN would automatically learn to compute these properties directly from a depth image, especially when very limited training data is available, as is the case with the NYUD2 dataset.

We use the CNN architecture proposed by Krizhevsky *et al.* in [105] and used by Girshick *et al.* in [49]. The network has about 60 million parameters and was trained on approximately 1.2 million RGB images from the 2012 ImageNet Challenge [32]. We refer the reader to [105] for details about the network. Our hypothesis, to be borne out in experiments, is that there is enough common structure between our HHA geocentric images and RGB images that a network designed for RGB images can also learn a suitable representation for HHA images. As an example, edges in the disparity and angle with gravity direction images correspond to interesting object boundaries (internal or external shape boundaries), similar to ones one gets in RGB images (but probably much cleaner).

Augmentation with synthetic data: An important observation is the amount of supervised training data that we have in the NYUD2 dataset is about one order of magnitude smaller than what is there for PASCAL VOC dataset (400 images as compared to 2500 images for PASCAL VOC 2007). To address this issue, we generate more data for training and finetuning the network. There are multiple ways of doing this: mesh the already available scenes and render the scenes from novel view points, use data from nearby video frames available in the dataset by flowing annotations using optical flow, use full 3D synthetic CAD objects models available over the Internet and render them into scenes. Meshing the point clouds may be too noisy and nearby frames from the video sequence maybe too similar and thus not very useful. Hence, we followed the third alternative and rendered the 3D annotations for NYUD2 available from [57] to generate synthetic scenes from various viewpoints. We also simulated the Kinect quantization model in generating this data (rendered depth images are converted to quantized disparity images and low resolution white noise was added to the disparity values).

2.3.1.2 Experiments

We work with the NYUD2 dataset and use the standard dataset splits into *train*, *val*, and *test* as described in Section 2.2.3. The dataset comes with semantic segmentation annotations, which we enclose in a tight box to obtain bounding box annotations. We work with the major furniture categories available in the dataset, such as chair, bed, sofa, table (listed in Table 2.2).

Experimental setup: There are two aspects to training our model: finetuning the convolutional neural network for feature learning, and training linear SVMs for object proposal classification.

Finetuning: We follow the R-CNN procedure from [49] using the Caffe CNN library [92]. We start from a CNN that was pretrained on the much larger ILSVRC 2012 dataset. For finetuning, the learning rate was initialized at 0.001 and decreased by a factor of 10 every 20k iterations. We finetuned for 30k iterations, which takes about 7 hours on a NVIDIA Titan GPU. Following [49], we label each training example with the class that has the maximally overlapping ground truth instance, if this overlap is larger than 0.5, and *background* otherwise. All finetuning was done on the *train* set.

SVM Training: For training the linear SVMs, we compute features either from pooling layer 5 (*pool5*), fully connected layer 6 (*fc6*), or fully connected layer 7 (*fc7*). In SVM training, we fixed the positive examples to be from the ground truth boxes for the target class and the negative examples were defined as boxes having less than 0.3 intersection over union with the ground truth instances from that class. Training was done on the *train* set with SVM hyper-parameters $C = 0.001$, $B = 10$, $w_1 = 2.0$ using liblinear [41]. We report the performance (detection average precision AP^b) on the *val* set for the control experiments. For the final experiment we train on *trainval* and report performance in comparison to other methods on the *test* set. At test time, we compute features from the *fc6* layer in the network, apply the linear classifier, and non-maximum suppression to the output, to obtain a set of sparse detections on the test image.

Results. We use the PASCAL VOC box detection average precision (denoted as AP^b following the generalization introduced in [70]) as the performance metric. Results are presented in Table 2.2. As a baseline, we report performance of the state-of-the-art non-neural network based detection

Table 2.2: Control experiments for object detection on NYUD2 *val* set. We investigate different ways to encode the depth image for use in a CNN for feature learning. Results are AP (in %). See Section 2.3.1.2.

	DPM		CNN									
inputs?	RGB	RGBD	RGB		disparity		HHA					RGB + HHA
synthetic data?								2x	15x	2x	2x	2x
CNN layer?			fc6	fc6	fc6	fc6	fc6	fc6	fc6	pool5	fc7	fc6
finetuned?			no	yes	no	yes	yes	yes	yes	yes	yes	yes
	A	B	C	D	E	F	G	H	I	J	K	L
bathtub	0.1	12.2	4.9	5.5	3.5	6.1	20.4	20.7	20.7	11.1	19.9	**22.9**
bed	21.2	56.6	44.4	52.6	46.5	63.2	60.6	67.2	**67.8**	61.0	62.2	66.5
bookshelf	3.4	6.3	13.8	19.5	14.2	16.3	20.7	18.6	16.5	20.6	18.1	**21.8**
box	0.1	0.5	1.3	1.0	0.4	0.4	0.9	1.4	1.0	1.0	1.1	**3.0**
chair	6.6	22.5	21.4	24.6	23.8	36.1	38.7	38.2	35.2	32.6	37.4	**40.8**
counter	2.7	14.9	20.7	20.3	18.5	32.8	32.4	33.6	36.3	24.1	35.0	**37.6**
desk	0.7	2.3	2.8	6.7	1.8	3.1	5.0	5.1	7.8	4.2	5.4	**10.2**
door	1.0	4.7	10.6	14.1	0.9	2.3	3.8	3.7	3.4	2.8	3.3	**20.5**
dresser	1.9	23.2	11.2	16.2	3.7	5.7	18.4	18.9	**26.3**	13.1	24.7	26.2
garbage-bin	8.0	26.6	17.4	17.8	2.4	12.7	26.9	29.1	16.4	21.4	25.3	**37.6**
lamp	16.7	25.9	13.1	12.0	10.5	21.3	24.5	26.5	23.6	22.3	23.2	**29.3**
monitor	27.4	27.6	24.8	32.6	0.4	5.0	11.5	14.0	12.3	17.7	13.5	**43.4**
night-stand	7.9	16.5	9.0	18.1	3.9	19.1	25.2	27.3	22.1	25.9	27.8	**39.5**
pillow	2.6	21.1	6.6	10.7	3.8	23.4	35.0	32.2	30.7	31.1	31.2	**37.4**
sink	7.9	**36.1**	19.1	6.8	20.0	28.5	30.2	22.7	24.9	18.9	23.0	24.2
sofa	4.3	28.4	15.5	21.6	7.6	17.3	36.3	37.5	39.0	30.2	34.3	**42.8**
table	5.3	14.2	6.9	10.0	12.0	18.0	18.8	22.0	22.6	21.0	22.8	**24.3**
television	16.2	23.5	29.1	31.6	9.7	14.7	18.4	23.4	26.3	18.9	22.9	**37.2**
toilet	25.1	48.3	39.6	52.0	31.2	**55.7**	51.4	54.2	52.6	38.4	48.8	53.0
mean	8.4	21.7	16.4	19.7	11.3	20.1	25.2	26.1	25.6	21.9	25.3	**32.5**

system, deformable part models (DPM) [42]. First, we trained DPMs on RGB images, which gives a mean AP^b of 8.4% (column A). While quite low, this result agrees with [154]. As a stronger baseline, we trained DPMs on features computed from RGB-D images (by using HOG on the disparity image and a histogram of height above ground in each HOG cell in addition to the HOG on the RGB image). These augmented DPMs (denoted RGBD-DPM) give a mean AP^b of 21.7% (column B). We also report results from the method of Girshick *et al.* [49], without and with fine tuning on the RGB images in the dataset, yielding 16.4% and 19.7% respectively (column C and column D). We compare results from layer *fc6* for all our experiments. Features from layers *fc7* and *pool5* generally gave worse performance.

The first question we ask is: Can a network trained only on RGB images can do anything when given disparity images? (We replicate each one-channel disparity image three times to match the three-channel filters in the CNN and scaled the input so as to have a distribution similar to RGB images.) The RGB network generalizes surprisingly well and we observe a mean AP^b of 11.3% (column E). This results confirms our hypothesis that disparity images have a similar structure to RGB images, and it may not be unreasonable to use an ImageNet-trained CNN as an initialization for finetuning on depth images. In fact, in our experiments we found that it was always better to finetune from the ImageNet initialization than to train starting with a random initialization.

We then proceed with finetuning this network (starting from the ImageNet initialization), and observe that performance improves to 20.1% (column F), already becoming comparable to RGBD-DPMs. However, finetuning with our HHA depth image encoding dramatically improves performance (by 25% relative), yielding a mean AP^b of 25.2% (column G).

We then observe the effect of synthetic data augmentation. Here, we add $2\times$ synthetic data, based on sampling two novel views of the given NYUD2 scene from the 3D scene annotations made available by [57]. We observe an improvement from 25.2% to 26.1% mean AP^b points (column H). However, when we increase the amount of synthetic data further ($15\times$ synthetic data), we see a small drop in performance (column H to I). We attribute the drop to the larger bias that has been introduced by the synthetic data. Guo *et al.*'s [57] annotations replace all non-furniture objects with cuboids, changing the statistics of the generated images. More realistic modeling for synthetic scenes is a direction for future research.

We also report performance when using features from other layers: *pool5* (column J) and *fc7* (column K). As expected the performance for *pool5* is lower, but the performance for *fc7* is also lower. We attribute this to over-fitting during finetuning due to the limited amount of data available.

Finally, we combine the features from both the RGB and the HHA image when finetuned on $2\times$ synthetic data (column L). We see there is consistent improvement from 19.7% and 26.1% individually to 32.5% (column L) mean AP^b. This is the final version of our system.

We also experimented with other forms of RGB and D fusion - early fusion where we passed in a 4 channel RGB-D image for finetuning but were unable to obtain good results (AP^b of 21.2%), and late fusion with joint finetuning for RGB and HHA (AP^b of 31.9%) performed comparably to our final system (individual finetuning of RGB and HHA networks) (AP^b of 32.5%). We chose the simpler architecture.

Test set performance: We ran our final system (column L) on the *test* set, by training on the complete *trainval* set. Performance is reported in Table 2.3. We compare against a RGB DPM, RGB-D DPM as introduced before. Note that our RGBD-DPMs serve as a strong baseline and are already an absolute 8.2% better than published results on the B3DO dataset [90] (39.4% as compared to 31.2% from the approach of Kim *et al.* [97], detailed results are in the supplementary material [65]). We also compare to Lin *et al.* [118]. [118] only produces 8, 15 or 30 detections per image which produce an average F_1 measure of 16.60, 17.88 and 18.14 in the 2D detection problem that we are considering as compared to our system which gives an average $\mathrm{F}_{\max}$ measure of 43.70. Precision Recall curves for our detectors along with the 3 points of operation from [118] are in the supplementary material [65]. We also report performance of a number of variants, where we show the effect of using more data for finetuning, and use of region features additionally computed from underlying regions associated with the bounding box (as detailed in Section 2.3.2).

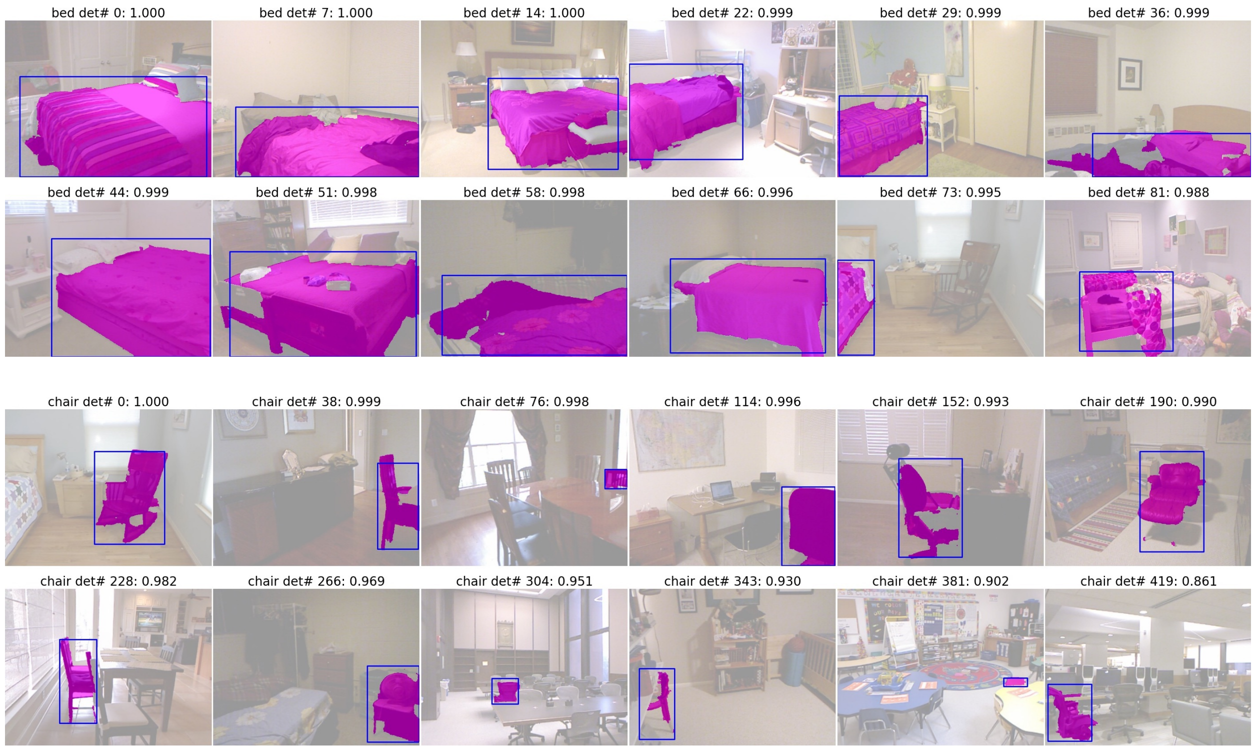


Figure 2.4: Output of our system: We visualize randomly sampled instance segmentation output for bed and chair categories from a more recent version of our system [59] that uses a deeper network and better representation for depth images.

2.3.2 Instance Segmentation

Bounding boxes by themselves are a fairly crude representation for objects in a scene, specially when we want to make 3D inferences about them. To address this, we study the problem of instance segmentation [70, 170]. Instance segmentation additionally involves inferring the pixel support for each object of the category of interest. We investigate two approaches for instance segmentation, via *Region Refinement* and via *Region Classification*.

2.3.2.1 Region Refinement

We formulate mask prediction as a two-class labeling problem (foreground versus background) on the pixels within the detection window. Our proposed method classifies each detection window pixel with a random forest classifier and then smoothes the predictions by averaging them over superpixels.

Learning framework: To train our random forest classifier, we associate each ground truth instance in the *train* set with a detection from our detector. We select the best scoring detection that overlaps the ground truth bounding box by more than 70%. For each selected detection, we warp the enclosed portion of the associated ground truth mask to a 50×50 grid. Each of these

Table 2.3: *Test* set results for detection on NYUD2 dataset. See text for details.

AP^b	fine-tuning set	mean	bath tub	bed	book shelf	box	chair	counter	desk	door	dresser	garbage bin	lamp	monitor	night stand	pillow	sink	sofa	table	tele vision	toilet
RGB DPM		**9.0**	0.9	27.6	9.0	0.1	7.8	7.3	0.7	2.5	1.4	6.6	22.2	10.0	9.2	4.3	5.9	9.4	5.5	5.8	34.4
RGB-D DPM		**23.9**	19.3	56.0	17.5	0.6	23.5	24.0	6.2	9.5	16.4	26.7	26.7	34.9	32.6	20.7	22.8	34.2	17.2	19.5	45.1
RGB R-CNN	train	**22.5**	16.9	45.3	28.5	0.7	25.9	30.4	9.7	16.3	18.9	15.7	27.9	32.5	17.0	11.1	16.6	29.4	12.7	27.4	44.1
Our (RGB + HHA, Box Feats.)	train	**35.9**	39.5	69.4	32.8	1.3	41.9	44.3	13.3	21.2	31.4	35.8	35.8	50.1	31.4	39.0	42.4	50.1	23.5	33.3	46.4
Our (RGB + HHA, Box Feats., Aug)	train	**37.3**	44.4	71.0	32.9	1.4	43.3	44.0	15.1	24.5	30.4	39.4	36.5	52.6	40.0	34.8	36.1	53.9	24.4	37.5	46.8
Our (RGB + HHA, Box Feats.)	trainval	**38.8**	36.4	70.8	35.1	3.6	47.3	46.8	14.9	23.3	38.6	43.9	37.6	52.7	40.7	42.4	43.5	51.6	22.0	38.0	47.7
Our (RGB + HHA, + Region Feats.)	trainval	**41.2**	39.4	73.6	38.4	5.9	50.1	47.3	14.6	24.4	42.9	51.5	36.2	52.1	41.5	42.9	42.6	54.6	25.4	48.6	50.2

2500 locations (per detection) serves as a training point.

Classifier: We train a single, monolithic classifier to process all 2500 locations. We implement it with a random forests [18]. Random forests naturally deal with multi-modal data and have been shown to work well with features derived from depth images [117, 153]. We adapt the open source random forest implementation in [33] to allow training and testing with on-the-fly feature computation. Our forests have ten decision trees.

Features: We compute a set of feature channels at each pixel in the original image (listed in supplementary material [65]). For each detection, we crop and warp the feature image to obtain features at each of the 50×50 detection window locations. The questions asked by our decision tree split nodes are similar to those in Shotton *et al.* [153] (which are themselves a generalization of those originally proposed by Geman *et al.* [48]). Specifically, we use two question types: *unary questions* obtained by thresholding the value in a channel relative to the location of a point, and *binary questions* obtained by thresholding the difference between two values, at different relative positions, in a particular channel. Shotton *et al.* [153] scale their offsets by the depth of the point to classify. We find that depth scaling is unnecessary after warping each instance to a fixed size and scale.

Testing: During testing, we work with the top $5K$ detections for each category (and $10K$ for the *chair* category, this gives us enough detections to get to 10% or lower precision). For each detection we compute features and pass them through the random forest to obtain a 50×50 foreground confidence map. We unwarp these confidence maps back to the original detection window and accumulate the per pixel predictions over superpixels. We select a threshold on the soft mask by optimizing performance on the *val* set.

2.3.2.2 Region Classification

An alternative approach is to classify region proposals as obtained in Section 2.2 into object classes based on region intersection over union with ground truth object segments rather than box intersection over union [49, 70]. We adopt the two-pathway network from Hariharan *et al.* [70], and use CNN features computed from the region's bounding box as well as the masked region itself.

Table 2.4: *Test* set results for instance segmentation on NYUD2. See text for details.

AP^r	fine-tuning set	mean	bath tub	bed	book shelf	box	chair	counter	desk	door	dresser	garbage bin	lamp	monitor	night stand	pillow	sink	sofa	table	tele vision	toilet
Bounding Box		**14.0**	5.9	40.0	4.1	0.7	5.5	0.5	3.2	14.5	26.9	32.9	1.2	40.2	11.1	6.1	9.4	13.6	2.6	35.1	11.9
Region		**28.1**	32.4	54.9	9.4	1.1	27.0	21.4	8.9	20.3	29.0	37.1	26.3	48.3	38.6	33.1	30.9	30.5	10.2	33.7	39.9
Foreground Mask		**28.0**	14.7	59.9	8.9	1.3	29.2	5.4	7.2	22.6	33.2	38.1	31.2	54.8	39.4	32.1	32.0	36.2	11.2	37.4	37.5
Our (RGB + HHA, Random Forest)	train	**32.1**	18.9	66.1	10.2	1.5	35.5	32.8	10.2	22.8	33.7	38.3	35.5	53.3	42.7	31.5	34.4	40.7	14.3	37.4	50.3
Our (RGB + HHA, Region Feats.)	train	**34.0**	33.8	64.4	9.8	2.3	36.6	41.3	9.7	20.4	30.9	47.4	26.6	51.6	27.5	42.1	37.1	44.8	14.7	42.7	62.6
Our (RGB + HHA, Region Feats.)	trainval	**37.5**	42.0	65.1	12.7	5.1	42.0	42.1	9.5	20.5	38.0	50.3	32.8	54.5	38.2	42.0	39.4	46.6	14.8	48.0	68.4

2.3.2.3 Results

To evaluate instance segmentation performance we use the region detection average precision AP^r metric (with a threshold of 0.5) as proposed in [70]. This extends the average precision metric used for bounding box detection by replacing bounding box overlap with region overlap (intersection over union). Note that this metric captures more information than the semantic segmentation metric as it respects the notion of instances, which is a goal of this paper.

We report the performance of our system in Table 2.4. We compare against three baseline methods: 1) *Bounding Box* where we simply assume the mask to be the box for the detection and project it to superpixels, 2) *Region* where we average the region proposals that resulted in the detected bounding box and project this to superpixels, and 3) *Foreground Mask* where we compute an empirical mask from the set of ground truth masks corresponding to the detection associated with each ground truth instance in the *training* set.

Both proposed methods, that of region refinement and region prediction perform better than these baselines, and obtain a performance of 32.1% and 34.0% mean AP^r. Our instance segmentor is effective and corrects mis-localized detections. Some categories even have a higher AP^r than AP^b. Region features additionally also boost performance for the object detection task (Table 2.3). Figure 2.4 shows sample instance segmentations from a more recent version of our system [59].

We use these final instance segmentations for the rest of this work. Of course, one could combine the region prediction and refinement [70] to obtain even better instance segmentations, but we chose to work with this intermediate output to minimize the number of times we train on the same data.

2.4 2.5D to Full 3D

2.4.1 Estimating Coarse Pose

In this section, we propose a convolutional neural network to estimate the coarse pose of rigid objects from a depth image. Contemporary work [173] studies the problem on RGB images.

Assume $C(k, n, s)$ is a convolutional layer with kernel size $k \times k$, n filters and a stride of s, $P_{\{max,ave\}}(k, s)$ a max or average pooling layer of kernel size $k \times k$ and stride s, N a local response normalization layer, RL a rectified linear unit, and $D(r)$ a dropout layer with dropout ratio r. Our network has the following architecture: $C(7, 96, 4) - RL - P_{max}(3, 2) - D(0.5) - N - C(5, 128, 2) - RL - P_{max}(3, 2) - N - C(3, (N_{pose} + 1)N_{class}, 1) - RL - P_{ave}(14, 1)$.

As input to the network we use 3-channel surface normal images, where the three channels encode N_x, N_y and N_z using the angle the normal vector makes with the three geocentric directions obtained with the gravity estimation algorithm from [58]. We use the angle in degrees and shift it to center at 128 instead of 90. Note that we do not use the HHA embedding [64] because it explicitly removes the azimuth information to allow learning pose-invariant representations for object detection.

Given that reliable annotations for such a detailed task are extremely challenging to obtain [57], we use 3D models from ModelNet [178] to train the network. In particular, we use the subset of models as part of the training set and work with the 10 categories for which the models are aligned to a canonical pose (bathtub, bed, chair, desk, dresser, monitor, night-stand, sofa, table, toilet). We sample 50 models for each category and render 10 different poses for each model placed on a horizontal floor at locations and scales estimated from the NYUD2 dataset [156] (some examples are provided in supplementary material). We place one object per scene, and sample boxes with more than 70% overlap with the ground truth box as training examples. We crop and warp the bounding box in the same way as Girshick *et al.* [49]. Note that warping the normals preserves the angles that are represented (as opposed to warping a depth image or a HHA image [64] which will change the orientation of surfaces being represented).

We train this network for classification using a softmax regression loss and share the lower layers of the network among different categories. We also adopt the geocentric constraint and assume that the object rests on a surface and hence must be placed flat on the ground. Thus, we only have to determine the azimuth of the object in the geocentric coordinate frame. We bin this azimuth into $N_{posebin}$ bins (20 in the experiments) and train the network to predict the bin for each example.

At test time, we simply forward propagate the image through the network and take the output pose bin as the predicted pose estimate. Given that the next stage requires a good initialization, in the experimental section we work with the top $k(= 2)$ modes of prediction.

2.4.2 Model Alignment

We now consider the problem of placing a 3D object model in the scene. We start from the instance segmentation output from Section 2.3.2, and infer the coarse pose of the object using the neural network introduced in Section 2.4.1. With this rough estimate of the pixel support of the object and a coarse estimate of its pose, we solve an alignment problem to obtain an optimal placement for the object in the scene.

2.4.2.1 Model Search

Note that our pose estimator provides only an orientation for the model. It does not inform about the size of the object, or about which model would fit the object best. Thus, in this stage, the algorithm searches over scales and CAD models, inferring an optimal rotation R and translation t for each candidate.

To search over scale, we gather category-level statistics from the 3D bounding box annotations of [57]. In particular, we use the area of the bounding box in the top view, estimate the mean of this area and its standard deviation, and take N_{scale} stratified samples from $\mathcal{N}(\mu_{area}, \sigma_{area})$. Such statistics do not require annotations and can also be obtained from online furniture catalogues. To search over scale, we isotropically scale each model to have the sampled area in the top-view.

To search over models, we select a small number N_{models} of 3D models for each category (5 in our experiments). Care was taken to pick distinct models, but this selection could also be done in a data-driven manner (by picking models that explain the data well).

Finally, we optimize over R and t iteratively using iterative closest point (ICP) [138], which we modify by constraining the rotation estimate to be consistent with the gravity direction. We initialize R using the pose estimate obtained from Section 2.4.1, and the inferred direction of gravity [58]. We initialize translation components t_x and t_z by using the median of the world co-ordinates of the points in the segmentation mask, and set t_y such that the model is resting on the floor (this constraint helps with heavily occluded objects, *e.g.* chairs, for which often only the back is visible). The following describes the model alignment procedure.

2.4.2.2 Model Alignment

The input to the model alignment algorithm is a depth image D, a segmentation mask S, a 3D model M at a given fixed scale s and an initial estimate of the transformation (a rotation matrix R_0 and a translation vector t_0) for the model. The output of the algorithm is a rotation R and a transformation t, such that the 3D model M rendered with transformations R and t explains as many points as possible in the segmentation mask S. We solve this problem approximately by the following procedure which we repeat for N iterations.

Render model: First, we use the current estimate of the transformation parameters (s, R, t) to render the model M and obtain a depth image of the model. We then back-project pixels from the segmentation mask S from the given depth image (to obtain point set P_{object}), and the points from the rendered model's depth image (to obtain point set P_{model}) to 3D space.

Re-estimate model transformation parameters: We run ICP to align points in P_{object} to points in P_{model}: a) we form correspondence by associating each point in P_{object} with the closest point in P_{model} (this prevents associations for occluded points in the object), b) we reject the worst 20% of the matches based on the distance (this allows the association to be robust in the presence of over-shoot in the segmentation mask S), and c) we constrain the rotation matrix R to operate only about the direction of gravity.

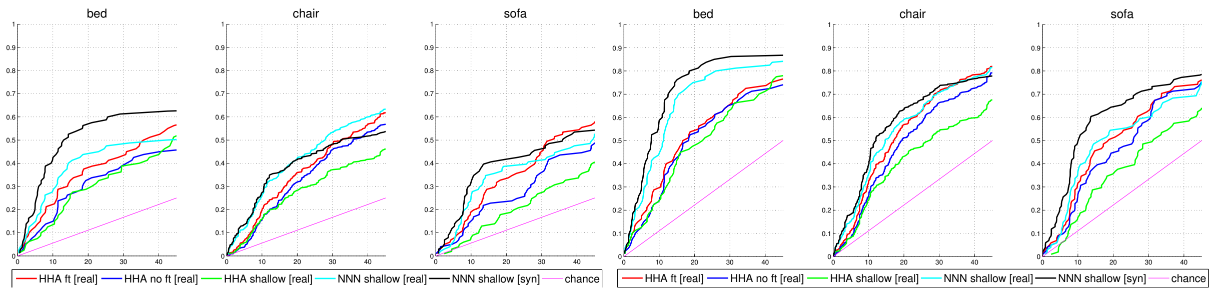


Figure 2.5: Performance on a NYUD2 *val* set. We plot accuracy (fraction of instances for which we are able to predict pose within a δ_θ angle) as a function of δ_θ. The top plots show top_1 accuracy and the bottom plots show top_2 accuracy. Note that real in the legend refers to model trained on real data, syn refers to the model trained on synthetic data and NNN stands for normal image.

2.4.2.3 Model Selection

Now we need to select the fitted model that best explains the data among $N_{scale}N_{model}$ candidates. We pose this selection as a learning problem and compute a set of features to capture the quality of the fit to the data. We compute the following features: number and fraction of pixels of the rendered model that are occluded, which are explained by the data, fraction and number of pixels of the input instance segmentation which are explained by the model, intersection over union overlap of the instance segmentation with mask of the model explained by the data, and mask of the model which is unoccluded. We learn a linear classifier on these features to pick the best fitting model. This classifier is trained with positives coming from rendered models which have more than 50% overlap with a ground truth region.

2.4.3 Experiments

We evaluate our approach on the NYUD2 dataset from Silberman *et al.* [156] and use the standard train set of 795 images and test set with 654 images. We split the 795 training images into 381 train and 414 validation images. For synthetic data we use the collection of aligned models made available by Wu *et al.* [178].

2.4.3.1 Coarse Pose Estimation

Here we describe our experiments to evaluate our coarse pose estimator. We present two evaluations, one on synthetic data and another one on real data.

To measure performance, we work with ground truth boxes and consider the distribution of the angular error in the top view. In particular, we plot the angular error δ_θ on the X-axis and the accuracy (the fraction of data which incurs less than δ_θ error) on the Y-axis. Note that we plot this graph for small ranges of δ_θ ($0°$ to $45°$) as accuracy in the high error ranges is useless from the perspective of model alignment. Moreover, since selecting among multiple hypotheses can

Table 2.5: Experiments for model placement on NYUD2: We report the AP^m for three different setting: using ground truth object segmentation masks, using latent positive segmentation masks and using the detection output from the instance segmentation from [64] (on the *val* set). We report performance on two different values for threshold t_{agree}. We also report performance on the *test* set. See Section 2.4.3.2 for details.

	val set								test set		
	ground truth segm		latent positive setting			detection setting			detection setting		
t_{agree}	0.5, 5 7	0.5, 5 ∞	0.5, 5 7	0.5, 5 ∞	AP^r upper bound	0.5, 5 7	0.5, 5 ∞	AP^r upper bound	0.5, 5 7	0.5, 5 ∞	AP^r upper bound
bathtub	57.4	76.8	55.3	83.3	94.7	6.7	19.4	25.7	7.9	50.4	42.0
bed	42.3	87.3	28.8	86.0	96.1	25.8	63.2	57.0	31.8	68.7	65.0
chair	45.3	74.1	29.0	56.9	70.1	11.8	25.2	30.4	14.7	35.6	42.9
desk	33.9	67.4	20.3	40.9	55.7	3.0	4.0	6.2	4.1	10.8	12.0
dresser	82.7	92.0	76.1	96.0	100.0	13.3	21.1	21.1	26.3	35.0	36.1
monitor	31.4	39.8	18.4	20.8	41.3	12.5	12.5	26.8	5.7	7.4	11.4
night-stand	62.5	77.6	51.3	65.2	87.9	18.9	21.6	25.5	28.1	33.7	34.8
sofa	45.1	85.0	28.5	72.0	92.4	10.5	30.4	37.7	21.8	48.5	47.4
table	18.8	52.2	15.8	34.3	46.8	5.5	11.9	13.3	5.6	12.3	15.0
toilet	66.0	100.0	46.0	86.0	100.0	35.9	72.4	73.2	41.8	68.4	68.4
mean	**48.5**	**75.2**	**37.0**	**64.1**	**78.5**	**14.4**	**28.2**	**31.7**	**18.8**	**37.1**	**37.5**

be beneficial for the alignment stage, a more appropriate metric is the top_k accuracy (fraction of instances which are within δ_θ of the top_k predictions of the model).

To evaluate this task, we work with the annotations from Guo and Hoiem [57], who annotated the NYUD2 dataset with 3D CAD models for the following 6 categories: chair, bed, sofa, table, desk and book shelf. To obtain interpretable results, we work with categories which have a clearly defined pose: chair, sofa and bed (bookshelf is not among the 10 categories which are pose aligned in ModelNet [178]). The top row in Figure 2.5 plots the top_1 accuracy and the second row plots top_2 accuracy. Note that there is a large number of objects which have missing depth data (for instance 30% of chairs have more than 50% missing depth pixels), hence we plot these curves only for instances with less than 50% depth pixels missing. We also experimented with the HHA network from [64] with and without fine-tuning for this task, training a shallow network from random initialization using HHA images and normal images. All these experiments are done by training on the real data, and we see that we are able to outperform these variants by training on clean synthetic data. Evaluation on the synthetic data is provided in the supplementary material.

2.4.3.2 Model Alignment

Performance Metric. Given that the output of our algorithm is a 3D model placed in the scene, it is not immediately obvious how to evaluate performance. One might think of evaluating individual tasks such as pose estimation, sub-type classification, key point prediction or instance segmentation, but doing these independently does not measure the performance of 3D model placement.

Moreover, for many categories we are considering there may not be a consistent definition of pose (*e.g.* table), or key points (*e.g.* sofa), or sub-types (*e.g.* chair).

Thus, to measure performance of placing 3D models in the scene, we propose a new metric which directly evaluates the fit of the inferred model with the observed depth image. We assume that there is a fixed library of 3D models $\mathcal{L}$, and a given algorithm $\mathcal{A}$ has to pick one of these models, and place it appropriately in the scene. We assume we have category-level instance segmentation annotations.

Our proposed metric is a generalization of the Average Precision, the standard metric for evaluating detection and segmentation [70]. Instead of just using the image-level intersection over union of the predicted box (in case of AP^b) or region (in case of AP^r) with the ground truth, we also enforce the constraint that the prediction must agree with the depth values observed in the image. In particular, we modify the way intersection between a prediction and a ground truth instance in computed. We render the model from the library $\mathcal{L}$ as proposed by the algorithm $\mathcal{A}$ to obtain a depth map and a segmentation mask. We then do occlusion checking with the given image to exclude pixels that are definitely occluded (based on a threshold $t_{occlusion}$). This gives us the visible part of the object $P_{visible}$. We then compute the intersection I between the output and the ground truth G by counting the number of pixels which are contained in both $P_{visible}$ and G, but in addition also agree on their depth values by being within a distance threshold of t_{agree} with each other. Union U is computed by counting the number of pixels in the ground truth G and the visible extent of the object $P_{visible}$ as $|G \cup P_{visible}|$. If this $\frac{I}{U}$, is larger than t_{iou} then this prediction is considered to explain the data well, otherwise not. With this modified definition of overlap, we plot a precision recall curve and measure the area under it as measure of the performance of the algorithm $\mathcal{A}$. We denote this average precision as AP^m. To account for the inherent noise in the sensor we operate with disparity as opposed to the depth value, and set thresholds $t_{occluded}$ and t_{agree} on disparity. This allows for larger error in far away objects as opposed to close by objects. While this behavior may not be desirable, it is unavoidable given the noise in the input depth image behaves similarly.

Evaluation. We evaluate our algorithm in 3 different settings: first using ground truth segmentations, second using high scoring instance segmentations from [64] that overlap with the ground truth by more than 50% (denoted as 'latent positive setting'), and third a completely unconstrained setting using only the instance segmentation output without any ground truth (denoted as 'detection setting'). Table 2.5 left summarizes results in these settings on the *val* set.

We use an t_{iou} of 0.5 to count a true positive, $t_{occlusion}$ of 5 disparity units, and report performance at two different values of t_{agree} 7 and ∞. An error of 7 disparity units corresponds to a 20 cm error at 3 meters. A t_{agree} of ∞ corresponds to AP^r subject to the constraint that the segmentation must come from the rendering of a 3D model.

We see that even when working with ground truth segmentations, estimating and placing a 3D model to explain the segment is a hard task. We obtain a (model average precision) AP^m of 48.5% in this setting. Even when evaluating at t_{agree} of ∞, we only get a performance of 75.2% which is indicative of the variety of our 3D model library and accuracy of our pose estimator.

In the second setting, we take the highest scoring detection which overlaps with more than 50% with the ground truth mask. Note that this setup decouples the performance of the detector from the performance of the model placement algorithm while at the same time exposing the

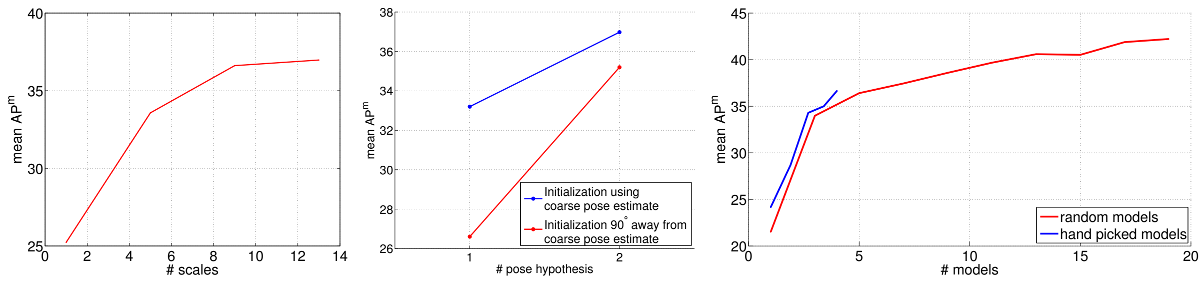


Figure 2.6: Control Experiments: Variation in AP^m. See Section 2.4.3.2 for details.

model placement algorithm with noisier segmentation masks. Under this setting, the AP^r upper bound is 78.5% which means that only that percentage of regions have a bottom-up region which overlaps with more than 0.5 with the ground truth mask, indicating the recall of the region proposal generator that we are using [64]. In this setting the performance at $t_{agree} = \infty$ is 64.1% and at $t_{agree} = 7$ is 37.0%. This shows that our model alignment is fairly robust to segmentation errors and we see a small drop in performance from 48.5% to 37.0% when moving from ground truth setting to latent positive setting.

In the detection setting (using no ground truth information at all), we observe an AP^r upper bound of 31.7% (which is comparable to AP^r reported in Table 2.4 but slightly different because (a) these are on the validation set, and (b) we ignore pixels with missing depth values in computing this metric). In this setting we observe a performance of 14.4% for t_{agree} of 7 and 28.2% for t_{agree} of ∞. We also report AP^m on the *test* set in the detection setting in Table 2.5 right.

Control Experiments. We perform additional control experiments to study the affect of the number of scales, the number of models, difference in hand picking models versus randomly picking models, number of pose hypotheses, and the importance of initialization for the model alignment stage. These experiments are summarized in Figure 2.6 and discussed below.

As expected, performance improves as we search over more scales (but saturates around 10 scales) (Figure 2.6 top). The performance increases as we use more models. Hand picking models so that they capture different modes of variation is better than picking models randomly, and that performance does not seem to saturate as we keep increasing the number of models we use during model alignment step (Figure 2.6 bottom), although this comes at proportionately larger computation time. Finally, using two pose hypothesis is better than using a single hypothesis. The model alignment stage is indeed sensitive to initialization and works better when used with the pose estimate from Section 2.4.1. This difference is more pronounced when using a single pose hypothesis (33% using our pose estimate versus 27% when not using it, Figure 2.6 middle).

Qualitative Visualizations. Finally, we provide qualitative visualizations of the output of our method in Figure 2.7 where we have replaced multiple objects with correspondent 3D models. Many more are available in the supplementary material.

Table 2.6: *Test* set results for 3D detection on NYUD2: We report the 3D detection AP [163]. We use the evaluation code from [163]. '3D all' refers to the setting with all object instances where as '3D clean' refers to the setting when instances with heavy occlusion and missing depth are considered difficult and not used for evaluation [163]. Note, the 'no RGB' version uses only the depth image for all steps except for region proposal generation, we do not expect this to impact this result significantly. See Section 2.4.3.3 for details.

	3D all						3D clean					
	mean	bed	chair	sofa	table	toilet	**mean**	bed	chair	sofa	table	toilet
Our (3D Box on instance segm. from [64])	48.4	74.7	18.6	50.3	28.6	69.7	66.1	90.9	45.9	68.2	25.5	100.0
Our (3D Box around estimated model)	58.5	73.4	44.2	57.2	33.4	84.5	71.1	82.9	72.5	75.3	24.6	100.0
Song and Xiao [163]	39.6	33.5	29.0	34.5	33.8	67.3	64.6	71.2	**78.7**	41.0	**42.8**	89.1
Our [no RGB] (3D Box on instance segm. from [64])	46.5	71.0	18.2	49.6	30.4	63.4	62.3	**86.9**	43.6	57.4	26.6	**96.7**
Our [no RGB] (3D Box around estimated model)	**57.6**	**72.7**	**47.5**	**54.6**	**40.6**	**72.7**	**70.7**	84.9	75.7	**62.8**	33.7	**96.7**

2.4.3.3 3D Detection

We next illustrate the richness of our approach by demonstrating results on the task of 3D object detection. Note that our method outputs a model aligned with objects in the image. A trivial side-product of our output is a 3D bounding box (obtained by putting a box around the inferred 3D model). We use this 3D bounding box as our output for 3D detection task and compare to the method from Song and Xiao [163] which was specifically designed and trained for this task.

We tackle the 3D detection task in the setting proposed by Song and Xiao in [163], who work with images from the NYUD2 dataset but create *different* splits for *different* categories and consider two levels of difficulty: a 'clean' task where they remove instances which are heavily occluded or have missing depth, and an 'all' task in which they consider all instances. Given their use of non-standard splits which are different from the standard NYUD2 dataset splits, we evaluate on the intersection of the standard NYUD2 test set and their test set for each category being studied.

In addition, we also compare to a simple baseline using the instance segmentation from [64] as described in Section 2.3.2 for 3D detection. We use a simple heuristic here: putting a tight fitting box around the 3D points in the inferred instance segmentation. We determine the extent of the box in the top view by searching over the orientation of the rectangular box such that its area is minimized, set the bottom of the box to rest on the floor and estimate the height as the maximum height of the points in the instance segmentation. All these operations are done using percentiles (δ and $100 - \delta$, with $\delta = 2$) to be robust to outliers.

We report the performance in Table 2.6 (Precision Recall curves are available in the supplementary material). We observe that this simple strategy of fitting a box around the inferred instance segmentation (denoted as 'Our (3D Box on instance segmentation from Gupta *et al.* [64])' in Table 2.6) already works better than the method proposed in [163] which was specifically designed for this task. At the same time, this method is faster (40 seconds CPU + 30 seconds on a GPU) and scales well with number of categories, as compared to 25 minutes per categories per image for [163]. This result shows that starting with well established 2D reasoning (since [64] does 2D reasoning, it is more readily able to leverage rich features for RGB images) to prune out large parts of the search space is not only more efficient, but also more accurate than starting from 3D

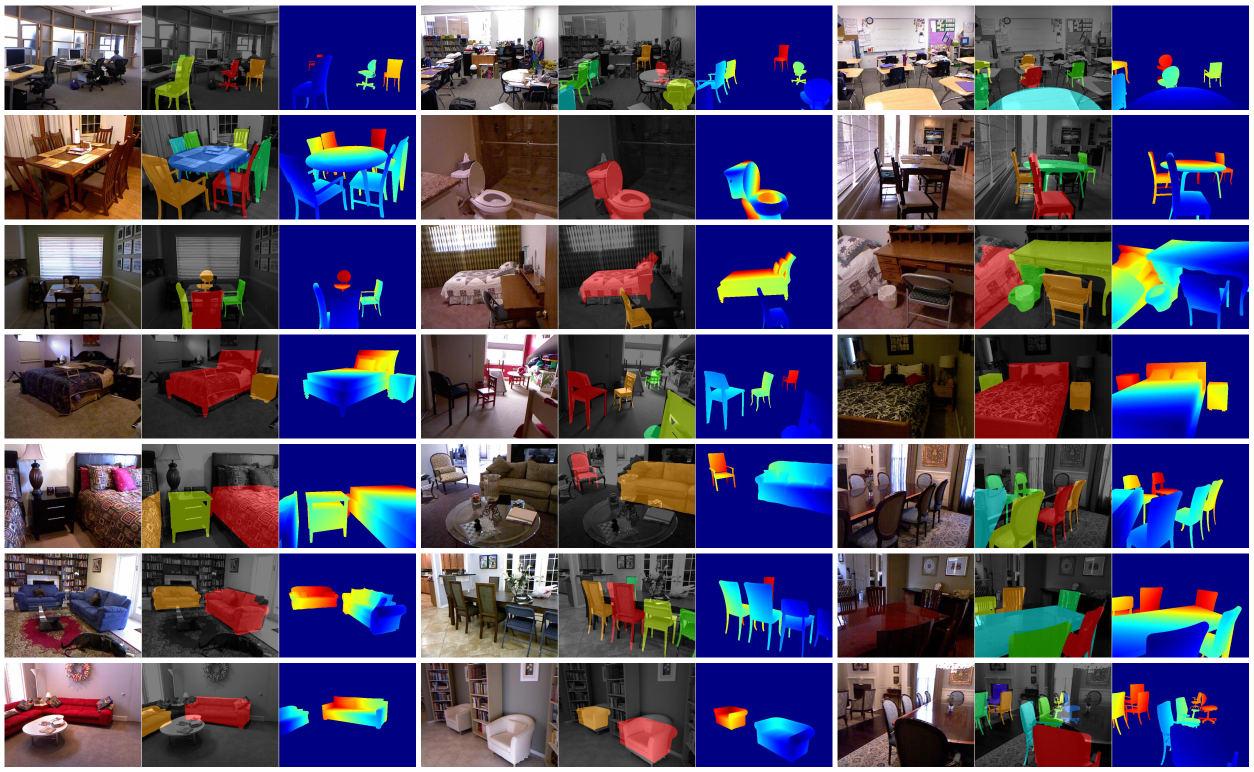


Figure 2.7: Visualizations of the output on the *test* set: We show images with multiple objects replaced with corresponding 3D CAD models. We show the image, models overlaid onto the image and the depth map for models placed in the scene. Depth maps are visualized using the 'jet' colormap, far away points are red and and close by points are blue.

reasoning for such tasks.

Finally, a 3D box around our final output (denoted 'Our (3D Box around estimated model)') outperforms both [163] and the baseline of putting a 3D bounding box around the instance segmentation output, providing further empirical evidence for the efficacy and utility of the methods proposed in the paper. We observe a large improvement over the baseline in performance for non-box like objects, chair, sofa and toilet. The improvement for chair is particularly striking (18.6% to 44.2% in the 'all' setting). This is because chairs are often heavily occluded (*e.g.* chair occluded behind a table) and the box around the visible extent systematically underestimates the actual amodal box.

Guo and Hoiem [56] also align 3D CAD models to objects in the image. We also compare to their work on this 3D detection task. We take the scenes produced by the algorithm from [56], compute tight 3D bounding boxes around detected objects and benchmark them in the same setup as described above to obtain a point on the Precision Recall plot (available in the supplementary material) for categories that both works consider: bed, chair, table and sofa. This comparison is also favorable to our method, and on average we obtain twice as much precision at the same recall and twice as much recall at the same precision.

Lastly, we also report performance of our system when only using the depth image for object detection, pose estimation and model placement steps (last two rows) (the bottom-up region generation step still uses the RGB image, we do not expect this to impact this result significantly). We see that this version of our system is better than the full version for some categories. We believe the reason is RGB information allows our full system to detect objects with missing depth with high scores which become high scoring false positives when the model placement step fails in the absence of enough depth data. On average this ablated version of our system performs comparably to our final system, and continues to outperform the algorithm from [163].

Chapter 3

Visual Navigation

As humans, when we navigate through novel environments, we draw on our previous experience in similar conditions. We reason about free-space, obstacles and the topology of the environment, guided by common sense rules and heuristics for navigation. For example, to go from one room to another, I must first exit the initial room; to go to a room at the other end of the building, getting into a hallway is more likely to succeed than entering a conference room; a kitchen is more likely to be situated in open areas of the building than in the middle of cubicles. The goal of this paper is to design a learning framework for acquiring such expertise, and demonstrate this for the problem of robot navigation in novel environments.

However, classic approaches to navigation rarely make use of such common sense patterns. Classical SLAM based approaches [29, 169] first build a 3D map using LIDAR, depth, or structure from motion, and then plan paths in this map. These maps are built purely geometrically, and nothing is known until it has been explicitly observed, even when there are obvious patterns. This becomes a problem for goal directed navigation. Humans can often guess, for example, where they will find a chair or that a hallway will probably lead to another hallway but a classical robot agent can at best only do uninformed exploration. The separation between mapping and planning also makes the overall system unnecessarily fragile. For example, the mapper might fail on texture-less regions in a corridor, leading to failure of the whole system, but precise geometry may not even be necessary if the robot just has to keep traveling straight.

Inspired by this reasoning, recently there has been an increasing interest in more end-to-end learning-based approaches that go directly from pixels to actions [183, 127, 114] without going through explicit model or state estimation steps. These methods thus enjoy the power of being able to learn behaviors from experience. However, it is necessary to carefully design architectures that can capture the structure of the task at hand. For instance Zhu *et al.* [183] use reactive memory-less vanilla feed forward architectures for solving visual navigation problems, In contrast, experiments by Tolman [171] have shown that even rats build sophisticated representations for space in the form

This chapter is based on work done with Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik and was presented at CVPR 2017 [61].

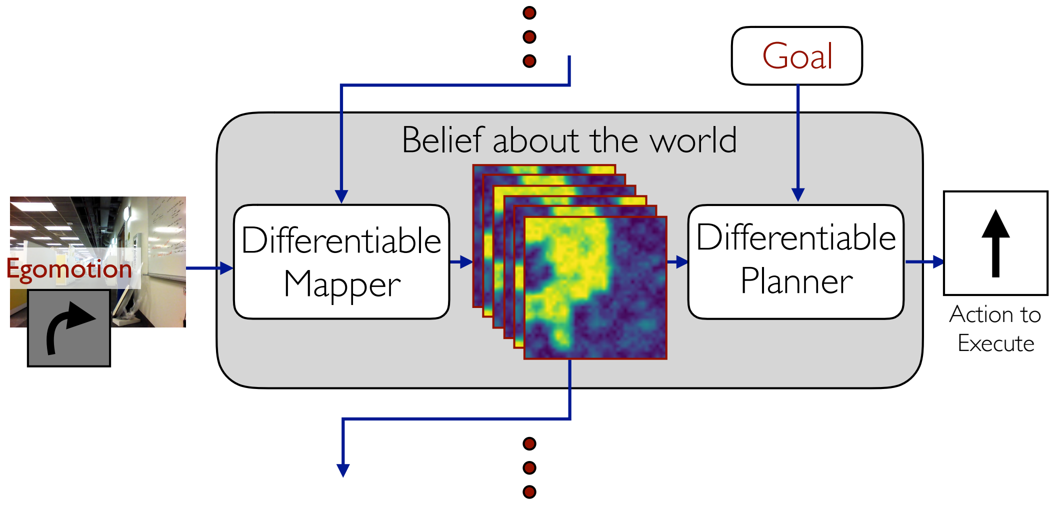


Figure 3.1: Overall network architecture: Our learned navigation network consists of mapping and planning modules. The mapper writes into a latent spatial memory that corresponds to an egocentric map of the environment, while the planner uses this memory alongside the goal to output navigational actions. The map is not supervised explicitly, but rather emerges naturally from the learning process.

of 'cognitive maps' as they navigate, giving them the ability to reason about shortcuts, something that a reactive agent is unable to.

This motivates our Cognitive Mapping and Planning (CMP) approach for visual navigation (Figure 3.1). CMP consists of a) a spatial memory to capture the layout of the world, and b) a planner that can plan paths given partial information. The mapper and the planner are put together into a unified architecture that can be trained to leverage regularities of the world. The mapper fuses information from input views as observed by the agent over time to produce a metric egocentric multi-scale belief about the world in a top-down view. The planner uses this multi-scale egocentric belief of the world to plan paths to the specified goal and outputs the optimal action to take. This process is repeated at each time step to convey the agent to the goal.

At each time step, the agent updates the belief of the world from the previous time step by a) using the ego-motion to transform the belief from the previous time step into the current coordinate frame and b) incorporating information from the current view of the world to update the belief. This allows the agent to progressively improve its model of the world as it moves around. The most significant contrast with prior work is that our approach is trained end-to-end to take good actions in the world. To that end, instead of analytically computing the update to the belief (via classical structure from motion) we frame this as a learning problem and train a convolutional neural network to predict the update based on the observed first person view. We make the belief transformation and update operations differentiable thereby allowing for end-to-end training. This allows our method to adapt to the statistical patterns in real indoor scenes without the need for any explicit supervision of the mapping stage.

Our planner uses the metric belief of the world obtained through the mapping operation described above to plan paths to the goal. We use value iteration as our planning algorithm but

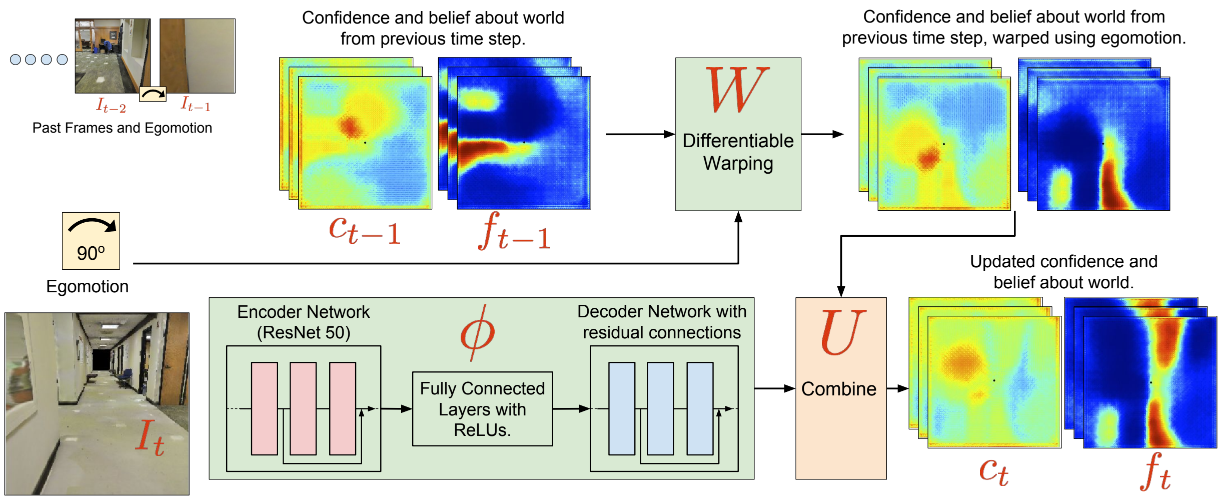

Confidence and belief about world
from previous time step.
Confidence and belief about world from
previous time step, warped using egomotion.
I_{t-2}
I_{t-1}
Past Frames and Egomotion
c_{t-1}
f_{t-1}
W
Differentiable Warping
90°
Egomotion
Updated confidence and
belief about world.
Encoder Network (ResNet 50)
ϕ
Decoder Network with residual connections
Fully Connected Layers with ReLUs.
U
Combine
I_t
c_t
f_t

Figure 3.2: Architecture of the mapper: The mapper module processes first person images from the robot and integrates the observations into a latent memory, which corresponds to an egocentric map of the top-view of the environment. The mapping operation is not supervised explicitly – the mapper is free to write into memory whatever information is most useful for the planner. In addition to filling in obstacles, the mapper also stores confidence values in the map, which allows it to make probabilistic predictions about unobserved parts of the map by exploiting learned patterns.

crucially use a trainable, differentiable and hierarchical version of value iteration. This has three advantages, a) being trainable it naturally deals with partially observed environments by explicitly learning when and where to explore, b) being differentiable it enables us to train the mapper for navigation, and c) being hierarchical it allows us to plan paths to distant goal locations in time complexity that is logarithmic in the number of steps to the goal.

Our approach is a reminiscent of classical work in navigation that also involves building maps and then planning paths in these maps to reach desired target locations. However, our approach differs from classical work in the following significant way: except for the architectural choice of maintaining a metric belief, everything else is learned from data. This leads to some very desirable properties: a) our model can learn statistical regularities of indoor environments in a task-driven manner, b) jointly training the mapper and the planner makes our planner more robust to errors of the mapper, and c) our model can be used in an online manner in novel environments without requiring a pre-constructed map.

3.1 Background

Navigation is one of the most fundamental problems in mobile robotics. The standard approach is to decompose the problem into two separate stages: (1) mapping the environment, and (2) planning a path through the constructed map [96, 39]. Decomposing navigation in this manner allows each stage to be developed independently, but prevents each from exploiting the specific needs of the other. A comprehensive survey of classical approaches for mapping and planning can be found in [169].

Mapping has been well studied in computer vision and robotics in the form of structure from motion and simultaneous localization and mapping [46, 87, 79, 160] with a variety of sensing modalities such as range sensors, RGB cameras and RGB-D cameras. These approaches take a purely geometric approach. Learning based approaches [181, 67] study the problem in isolation thus only learning generic task-independent maps. Path planning in these inferred maps has also been well studied, with pioneering works from Canny [20], Kavraki *et al.* [94] and LaValle and Kuffner [111]. Works such as [40, 44] have studied the joint problem of mapping and planning. While this relaxes the need for pre-mapping by incrementally updating the map while navigating, but still treat navigation as a purely geometric problem, Konolige *et al.* [103] and Aydemir *et al.* [10] proposed approaches which leveraged semantics for more informed navigation. Kuipers *et al.* [106] introduce a cognitive mapping model using hierarchical abstractions of maps. Semantics have also been associated with 3D environments more generally [104, 58].

As an alternative to separating out discrete mapping and planning phases, reinforcement learning (RL) methods directly learn policies for robotic tasks [98, 132, 101]. A major challenge with using RL for this task is the need to process complex sensory input, such as camera images. Recent works in deep reinforcement learning (DRL) learn policies in an end-to-end manner [127] going from pixels to actions. Follow-up works [126, 55, 146] propose improvements to DRL algorithms, [130, 126, 176, 76, 182] study how to incorporate memory into such neural network based models. We build on the work from Tamar *et al.* [167] who study how explicit planning can be incorporated in such agents, but do not consider the case of first-person visual navigation, nor provide a framework for memory or mapping. [130] study the generalization behavior of these algorithms to novel environments they have not been trained on.

In context of navigation, learning and DRL has been used to obtain policies [172, 183, 130, 167, 93, 51, 25, 2]. Some of these works [93, 51], focus on the problem of learning controllers for effectively maneuvering around obstacles directly from raw sensor data. Others, such as [167, 16, 130], focus on the planning problem associated with navigation under full state information [167], designing strategies for faster learning via episodic control [16], or incorporate memory into DRL algorithms to ease generalization to new environments. Most of this research (except [183]) focuses on navigation in synthetic mazes which have little structure to them. Given these environments are randomly generated, the policy learns a random exploration strategy, but has no statistical regularities in the layout that it can exploit. We instead test on layouts obtained from real buildings, and show that our architecture consistently outperforms feed forward and LSTM models used in prior work.

The research most directly relevant to our work is the contemporary work of Zhu *et al.* [183]. Similar to us, Zhu *et al.* also study first-person view navigation using macro-actions in more realistic environments instead of synthetic mazes. Zhu *et al.* propose a feed forward model which when trained in one environment can be finetuned in another environment. Such a memory-less agent cannot map, plan or explore the environment, which our expressive model naturally does. Zhu *et al.* also don't consider zero-shot generalization to previously unseen environments, and focus on smaller worlds where memorization of landmarks is feasible. In contrast, we explicitly handle generalization to new, never before seen interiors, and show that our model generalizes successfully to floor plans not seen during training.

Relationship to contemporary research. In this paper, we used scans of real world environments to construct visually realistic simulation environments to study representations that can enable navigation in novel previously unseen environments. Since conducting this research, there has been a major thrust in this direction in computer vision and related communities. Numerous works such as [22, 26, 3] have collected large-scale datasets consisting of scans of real world environments, while [144, 177, 179] have built more sophisticated simulation environments based on such scans. A related and parallel stream of research studies the question of whether or not models trained in simulators can be effectively transferred to the real world [139], and how the domain gap between simulation and the real world may be reduced [179]. A number of works have studied related navigation problems in such simulation environments [78, 23, 4]. Researchers have also gone beyond specifying goals as a desired location in space and finding objects of interest as done in this paper, for example, Wu *et al.* [177] generalize the goal specification to also include rooms of interest, and Das *et al.* [28] allow goal specification via templated questions. Finally, a number of works have also pursued the problem of building representation for space in context of navigation. [131, 15, 95, 54] use similar $2D$ spatial representations, Mirowski *et al.* [125] use fully-connected LSTMs, while Savinov *et al.* [142] develop topological representations. Interesting reinforcement learning techniques have also been explored for the task of navigation [125, 38].

3.2 Problem Setup

To be able to focus on the high-level mapping and planning problem we remove confounding factors arising from low-level control by conducting our experiments in simulated real world indoor environments. Studying the problem in simulation makes it easier to run exhaustive evaluation experiments, while the use of scanned real world environments allows us to retains the richness and complexity of real scenes. We also only study the static version of the problem, though extensions to dynamic environments would be interesting to explore in future work.

We model the robot as a cylinder of a fixed radius and height, equipped with vision sensors (RGB cameras or depth cameras) mounted at a fixed height and oriented at a fixed pitch. The robot is equipped with low-level controllers which provide relatively high-level macro-actions $\mathcal{A}_{x,\theta}$. These macro-actions are a) stay in place, b) rotate left by θ, c) rotate right by θ, and d) move forward x cm, denoted by a_0, a_1, a_2 and a_3, respectively. We further assume that the environment is a grid world and the robot uses its macro-actions to move between nodes on this graph. The robot also has access to its precise egomotion. This amounts to assuming perfect visual odometry [129], which can itself be learned [66], but we defer the joint learning problem to future work.

We want to learn policies for this robot for navigating in *novel* environments that it has not previously encountered. We study two navigation tasks, a *geometric* task where the robot is required to go to a target location specified in robot's coordinate frame (*e.g.* $250cm$ forward, $300cm$ left) and a *semantic* task where the robot is required to go to an object of interest (*e.g.* a chair).

These tasks are to be performed in novel environments, neither the exact environment map nor its topology is available to the robot.

Our navigation problem is defined as follows. At a given time step t, let us assume the robot is at a global position (position in the world coordinate frame) P_t. At each time step the robot receives as input the image of the environment $\mathcal{E}$, $I_t = I(\mathcal{E}, P_t)$ and a target location $(x_t^g, y_t^g, \theta_t^g)$ (or a semantic goal) specified in the coordinate frame of the robot. The navigation problem is to learn a policy that at every time steps uses these inputs (current image, egomotion and target specification) to output the action that will convey the robot to the target as quickly as possible.

Experimental Testbed. We conduct our experiments on the Stanford large-scale 3D Indoor Spaces (S3DIS) dataset introduced by Armeni *et al.* [8]. The dataset consists of 3D scans (in the form of textured meshes) collected in 6 large-scale indoor areas that originate from 3 different buildings of educational and office use. The dataset was collected using the Matterport scanner [124]. Scans from 2 buildings were used for training and the agents were tested on scans from the 3rd building. We pre-processed the meshes to compute space traversable by the robot. We also precompute a directed graph $\mathcal{G}_{x,\theta}$ consisting of the set of locations the robot can visit as nodes and a connectivity structure based on the set of actions $\mathcal{A}_{x,\theta}$ available to the robot to efficiently generate training problems. More details in supplementary material [62].

3.3 Mapping

We describe how the mapping portion of our learned network can integrate first-person camera images into a top-down 2D representation of the environment, while learning to leverage statistical structure in the world. Note that, unlike analytic mapping systems, the map in our model amounts to a latent representation. Since it is fed directly into a learned planning module, it need not encode purely free space representations, but can instead function as a general spatial memory. The model learns to store inside the map whatever information is most useful for generating successful plans. However to make description in this section concrete, we assume that the mapper predicts free space.

The mapper architecture is illustrated in Figure 3.2. At every time step t we maintain a cumulative estimate of the free space f_t in the coordinate frame of the robot. f_t is represented as a multi-channel 2D feature map that metrically represents space in the top-down view of the world. f_t is estimated from the current image I_t, cumulative estimate from the previous time step f_{t-1} and egomotion between the last and this step e_t using the following update rule:

$$f_t = U\left(W\left(f_{t-1}, e_t\right), f'_t\right) \quad \text{where, } f'_t = \phi\left(I_t\right). \tag{3.1}$$

here, W is a function that transforms the free space prediction from the previous time step f_{t-1} according to the egomotion in the last step e_t, ϕ is a function that takes as input the current image I_t and outputs an estimate of the free space based on the view of the environment from the current location (denoted by f'_t). U is a function which accumulates the free space prediction from the current view with the accumulated prediction from previous time steps. Next, we describe how each of the functions W, ϕ and U are realized.

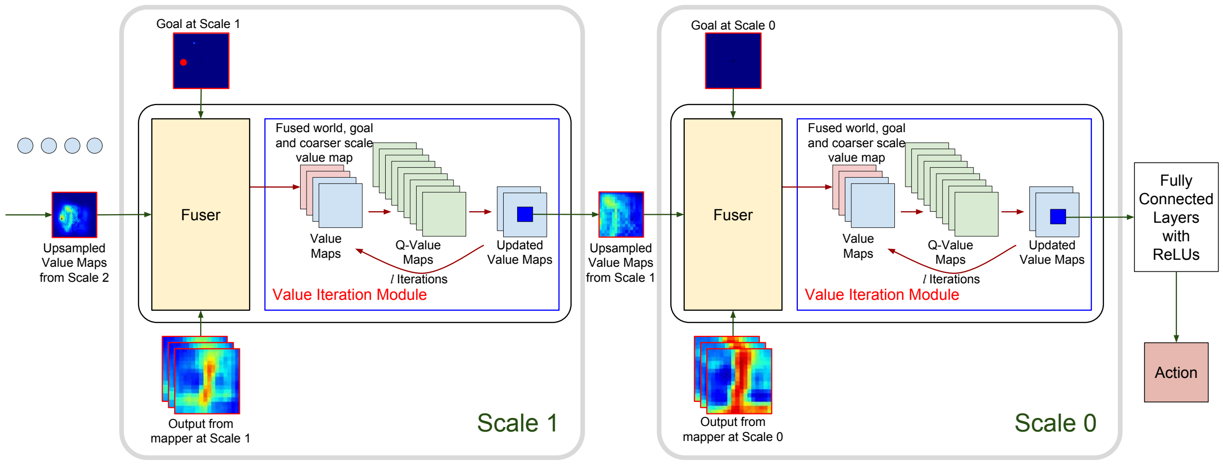

Goal at Scale 1
Upsampled Value Maps from Scale 2
Fuser
Fused world, goal and coarser scale value map
Value Maps
Q-Value Maps
Updated Value Maps
/ Iterations
Value Iteration Module
Output from mapper at Scale 1
Scale 1
Upsampled Value Maps from Scale 1
Goal at Scale 0
Fuser
Fused world, goal and coarser scale value map
Value Maps
Q-Value Maps
Updated Value Maps
/ Iterations
Value Iteration Module
Output from mapper at Scale 0
Scale 0
Fully Connected Layers with ReLUs
Action

Figure 3.3: Architecture of the hierarchical planner: The hierarchical planner takes the egocentric multi-scale belief of the world output by the mapper and uses value iteration expressed as convolutions and channel-wise max-pooling to output a policy. The planner is trainable and differentiable and back-propagates gradients to the mapper. The planner operates at multiple scales (scale 0 is the finest scale) of the problem which leads to efficiency in planning.

The function W is realized using bi-linear sampling. Given the ego-motion, we compute a backward flow field $\rho(e_t)$. This backward flow maps each pixel in the current free space image f_t to the location in the previous free space image f_{t-1} where it should come from. This backward flow ρ can be analytically computed from the ego-motion (as shown in supplementary material [62]). The function W uses bi-linear sampling to apply this flow field to the free space estimate from the previous frame. Bi-linear sampling allows us to back-propagate gradients from f_t to f_{t-1} [89], which will make it possible to train this model end to end.

The function ϕ is realized by a convolutional neural network. Because of our choice to represent free space always in the coordinate frame of the robot, this becomes a relatively easy function to learn, given the network only has to output free space in the current coordinate, rather than in an arbitrary world coordinate frame determined by the cumulative egomotion of the robot so far.

Intuitively, the network can use semantic cues (such as presence of scene surfaces like floor and walls, common furniture objects like chairs and tables) alongside other learned priors about size and shapes of common objects to generate free space estimates, even for object that may only be partiality visible. Qualitative results in supplementary material [62] show an example for this where our proposed mapper is able to make predictions for spaces that haven't been observed.

The architecture of the neural network that realizes function ϕ is shown in Figure 3.2. It is composed of a convolutional encoder which uses residual connections [71] and produces a representation of the scene in the 2D image space. This representation is transformed into one that is in the egocentric 2D top-down view via fully connected layers. This representation is up-sampled using up-convolutional layers (also with residual connections) to obtain the update to the belief about the world from the current frame.

In addition to producing an estimate of the free space from the current view f'_t the model also produces a confidence c'_t. This estimate is also warped by the warping function W and accumulated over time into c_t. This estimate allows us to simplify the update function, and can be thought of as playing the role of the update gate in a gated recurrent unit. The update function U takes in the tuples (f_{t-1}, c_{t-1}), and (f'_t, c'_t) and produces (f_t, c_t) as follows:

$$f_t = \frac{f_{t-1}c_{t-1} + f'_t c'_t}{c_{t-1} + c'_t} \quad \text{and} \quad c_t = c_{t-1} + c'_t \tag{3.2}$$

We chose an analytic update function to keep the overall architecture simple. This can be replaced with more expressive functions like those realized by LSTMs [81].

Mapper performance in isolation. To demonstrate that our proposed mapper architecture works we test it in isolation on the task of free space prediction. supplementary material [62] shows qualitative and quantitative results.

3.4 Planning

Our planner is based on value iteration networks proposed by Tamar *et al.* [167], who observed that a particular type of planning algorithm called value iteration [14] can be implemented as a neural network with alternating convolutions and channel-wise max pooling operations, allowing the planner to be differentiated with respect to its inputs. Value iteration can be thought of as a generalization of Dijkstra's algorithm, where the value of each state is iteratively recalculated at each iteration by taking a max over the values of its neighbors plus the reward of the transition to those neighboring states. This plays nicely with 2D grid world navigation problems, where these operations can be implemented with small 3×3 kernels followed by max-pooling over channels. Tamar *et al.* [167] also showed that this reformulation of value iteration can also be used to learn the planner (the parameters in the convolutional layer of the planner) by providing supervision for the optimal action for each state. Thus planning can be done in a trainable and differentiable manner by very deep convolutional network (with channel wise max-pooling). For our problem, the mapper produces the 2D top-view of the world which shares the same 2D grid world structure as described above, and we use value iteration networks as a trainable and differentiable planner.

Hierarchical Planning. Value iteration networks as presented in [167](v2) are impractical to use for any long-horizon planning problem. This is because the planning step size is coupled with the action step size thus leading to a) high computational complexity at run time, and b) a hard learning problem as gradients have to flow back for as many steps. To alleviate this problem, we extend the hierarchical version presented in [167].

Our hierarchical planner plans at multiple spatial scales. We start with a k times spatially downsampled environment and conduct l value iterations in this downsampled environment. The output of this value iteration process is center cropped, upsampled, and used for doing value iterations at a finer scale. This process is repeated to finally reach the resolution of the original problem. This procedure allows us to plan for goals which are as far as $l2^k$ steps away while performing (and backpropagating through) only lk planning iterations. This efficiency increase comes at the cost of approximate planning.

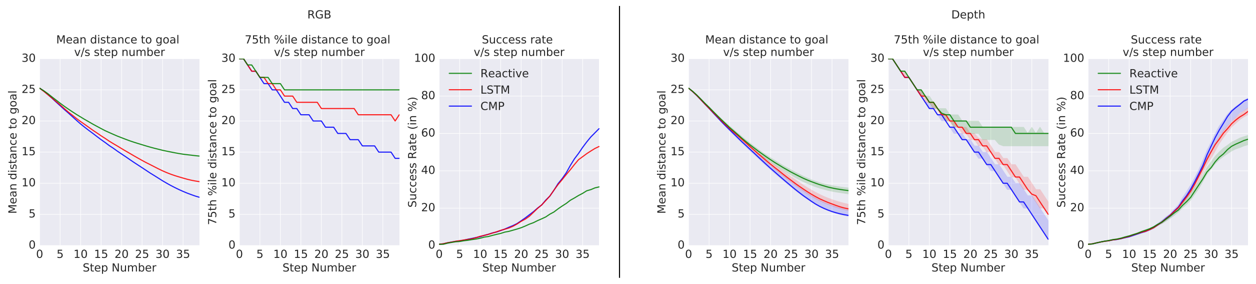


Figure 3.4: We report the mean distance to goal, 75th percentile distance to goal (lower is better) and success rate (higher is better) as a function of the number of steps for the 4 frame reactive agent, LSTM based agent and our proposed CMP based agent when using RGB images as input (left three plots) and when using depth images as input (right three plots). We note that CMP outperforms the two baselines in both cases, and generally using depth images as input leads to better performance than using RGB images as input. We also show the variance in performance over five re-trainings from different random initializations of the agents when using depth images as input (the solid line plots the median performance and the surrounding shaded region represents the minimum and maximum value across five different runs). We note that the variation in performance is reasonably small for all models and CMP consistently outperforms the two baseline.

Planning in Partially Observed Environments. Value iteration networks have only been evaluated when the environment is fully observed, *i.e.* the entire map is known while planning. However, for our navigation problem, the map is only partially observed. Because the planner is not hand specified but learned from data, it can learn policies which naturally take partially observed maps into account. Note that the mapper produces not just a belief about the world but also an uncertainty c_t, the planner knows which parts of the map have and haven't been observed.

3.5 Joint Architecture

Our final architecture, Cognitive Mapping and Planning (CMP) puts together the mapper and planner described above. At each time step, the mapper updates its multi-scale belief about the world based on the current observation. This updated belief is input to the planner which outputs the action to take. As described previously, all parts of the network are differentiable and allow for end-to-end training, and no additional direct supervision is used to train the mapping module – rather than producing maps that match some ground truth free space, the mapper produces maps that allow the planner to choose effective actions.

Training Procedure. We optimize the CMP network with fully supervised training using DAGGER [136]. We generate training trajectories by sampling arbitrary start and goal locations on the graph $\mathcal{G}_{x,\theta}$. We generate supervision for training by computing shortest paths on the graph. We use an online version of DAGGER, where during each episode we sample the next state based on the action from the agent's current policy, or from the expert policy. We use scheduled sampling and anneal the probability of sampling from the expert policy using inverse sigmoid decay.

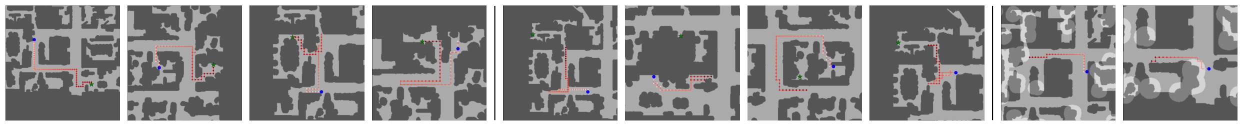


Figure 3.5: Representative Success and Failure Cases for CMP: We visualize trajectories for some typical success and failure cases for CMP. Dark gray regions show occupied space, light gray regions show free space. The agent starts from the blue dot and is required to reach the green star (or semantic regions shown in light gray). The agent's trajectory is shown by the dotted red line. While we visualize the trajectories in the top view, note that the agent only receives the first person view as input. **Left plots** show success cases for geometric task. We see that the agent is able to traverse large distances across multiple rooms to get to the target location, go around obstacles and quickly resolve that it needs to head to the next room and not the current room. The last two plots show cases where the agent successfully backtracks. **Center plots** show failure cases for geometric task: problems with navigating around tight spaces (entering through a partially opened door, and getting stuck in the corner (the gap is not big enough to pass through)), missing openings which would have lead to shorter paths, thrashing around in space without making progress. **Right plots** visualize trajectories for 'go to the chair' semantic task. The first figure shows a success case, while the right figure shows a typical failure case where the agent walks right through a chair region.

Note that the focus of this work is on studying different architectures for navigation. Our proposed architecture can also be trained with alternate paradigms for learning such policies, such as reinforcement learning. We chose DAGGER for training our models because we found it to be significantly more sample efficient and stable in our domain, allowing us to focus on the architecture design.

3.6 Experiments

All our models are trained asynchronously with 16 parallel GPU workers and 16 parameter servers using TensorFlow [1]. We used ADAM [99] to optimize our loss function and trained for 60K iterations with a learning rate of 0.001 which was dropped by a factor of 10 every 20K iterations (we found this necessary for consistent training across different runs). We use weight decay of 0.0001 to regularize the network and use batch-norm [86].

We use ResNet-50 [72] pre-trained on ImageNet [31] to represent RGB images. We transfer supervision from RGB images to depth images using cross modal distillation [59] between RGB-D image pairs rendered from meshes in the training set to obtain a pre-trained ResNet-50 model to represent depth images.

We compare our proposed CMP architecture to other alternate architectures such as a reactive agent and a LSTM based agent. Since the goal of this paper is to study various architectures for navigation we train all these architectures the same way using DAGGER [136] as described earlier.

Table 3.1: Navigation Results: We report the mean distance to goal location, 75th percentile distance to goal and success rate after executing the policy for 39 time steps. The top part presents results for the case where the goal is specified geometrically in terms of position of the goal in the coordinate frame of the robot. The bottom part presents aggregate results for the case where the goal is specified semantically in the form of 'go to a chair' (or door or table).

Method	Mean		75th %ile		Success %age	
	RGB	Depth	RGB	Depth	RGB	Depth
Geometric Task						
Initial	25.3	25.3	30	30	0.7	0.7
No Image LSTM	20.8	20.8	28	28	6.2	6.2
Reactive (1 frame)	20.9	17.0	28	26	8.2	21.9
Reactive (4 frames)	14.4	8.8	25	18	31.4	56.9
LSTM	10.3	5.9	21	5	53.0	71.8
Our (CMP)	**7.7**	**4.8**	**14**	**1**	**62.5**	**78.3**
Semantic Task (Aggregate)						
Initial	16.2	16.2	25	25	11.3	11.3
Reactive	14.2	14.2	22	23	23.4	22.3
LSTM	13.5	13.4	20	23	23.5	27.2
Our (CMP)	**11.3**	**11.0**	**18**	**19**	**34.2**	**40.0**

3.6.1 Geometric Task

We first present results for the task where the goal is specified geometrically in terms of position of the goal in robot's coordinate frame in Table 3.1 (top part) and Figure 3.4. Problems for this task are generated by first sampling a start node on the graph and then sampling an end node which is within 32 steps from the starting node and preferably in another room or in the hallway (we use room and hallway annotations from the dataset [8]). The same sampling process is used during training and testing. We sample 4000 problems for testing and these remain fixed across different algorithms that we compare. We measure performance using the distance to goal after running the learned policy for the episode length (39 time steps). We report multiple error metrics, the mean distance to goal, the 75th percentile distance to goal and the success rate (the agent succeeds if it is within a distance of three steps to the goal location at the end of the episode). Table 3.1 reports these metrics at end of the episode, while Figure 3.4 plots them across time steps. We report all numbers on the test set. The test set consists of a floor from an altogether different building not contained in the training set. (See dataset website and supplementary material [62] for environment visualizations.)

Nearest Neighbor Trajectory Transfer: To quantify similarity between training and testing environments, we transfer optimal trajectories from the train set to the test set using visual nearest neighbors (in RGB ResNet-50 feature space). This transfer is done as follows. At each time step we pick the location in the training set which results in the most similar view to that seen by the agent at the current time step. We then compute the optimal action that conveys the robot to the same relative offset in the training environment from this location and execute this action at the current time step. This procedure is repeated at each time step. Such a transfer leads to very poor results. The mean and median distance to goal is 22 and 25 steps respectively, highlighting the differences between the train and test environments.

No image, goal location only with LSTM: This refers to the experimental setting where we ignore the image and simply use the relative goal location (in robot's current coordinate frame) as input to a LSTM, and predict the action that the agent should take. The relative goal location is embedded into a K dimensional space via fully connected layers with ReLU non-linearities before being input to the LSTM. As expected, this does rather poorly.

Reactive Policy, Single Frame: We next compare to a reactive agent which uses the first-person view of the world. As described above we use ResNet-50 to extract features. These features are passed through a few fully connected layers, and combined with the representation for the relative goal location which is used to predict the final action. We experimented with additive and multiplicative combination strategies and both performed similarly. Note that this reactive baseline is able to perform well on the training environments obtaining a mean distance to goal of about 9 steps, but perform poorly on the test set only being able to get to within 17 steps of the goal on average. This suggests that a reactive agent is able to effectively memorize the environments it was trained on, but fails to generalize to novel environments, this is not surprising given it does not have any form of memory to allow it to map or plan. We also experimented with using Drop Out in the fully connected layers for this model but found that to hurt performance on both the train and the test sets.

Reactive Policy, Multiple Frames: We also consider the case where the reactive policy receives 3 previous frames in addition to the current view. Given the robot's step-size is fairly large we consider a late fusion architecture and fuse the information extracted from ResNet-50. Note that this architecture is similar to the one used in [183]. The primary differences are: goal is specified in terms of relative offset (instead of an image), training uses DAGGER (which utilizes denser supervision) instead of A3C, and testing is done in novel environments. These adaptations are necessary to make an interpretable comparison on our task. Using additional frames as input leads to a large improvement in performance, specially when using depth images.

LSTM Based Agent: Finally, we also compare to an agent which uses an LSTM based memory. We introduce LSTM units on the multiplicatively combined image and relative goal location representation. Such an architecture also gives the LSTM access to the egomotion of the agent (via how the relative goal location changes between consecutive steps). Thus this model has access to all the information that our method uses. We also experimented with other LSTM based models (ones without egomotion, inputting the egomotion more explicitly, *etc.*), but weren't able to reliably train them in early experiments and did not pursue them further. This LSTM based model is able to consistently outperform the reactive baseline.

We compare these baselines with of our proposed method. CMP is able to outperform all these baselines across all metrics for both RGB and depth image case. CMP achieves a lower 75$^{\text{th}}$ %ile distance to goal (14 and 1 as compared to 21 and 5 for the LSTM) and improves the success rate to 62.5% and 78.3% from 53.0% and 71.8%.

We also report variance in performance over five re-trainings from different random initial-

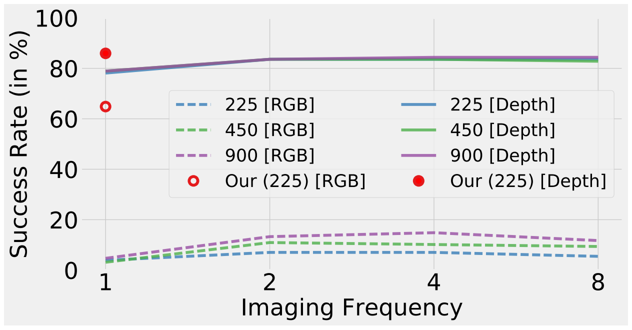


Figure 3.6: Comparison to Purely Geometric Mapping. We plot succcess rate for CMP and a purely geometric incremental mapping and path planning policy. The latter is implemented by constructing occupancy maps by projecting 3D points (either directly visible in case of depth images, or triangulated using SIFT matching in case of RGB images), and using an explicit planner on this occupancy map. We experiment with different image resolutions $(225, 450, 900)$ as well as different imaging frequency. We note that when using depth images as input, this baseline performs similar to CMP, but CMP performs much better when considering RGB images as input.

izations of the network for the 3 most competitive methods (Reactive with 4 frames, LSTM & CMP) for the depth image case. Figure 3.4 (right) shows the performance, the solid line shows the median metric value and the surrounding shaded region represents the minimum and maximum metric value over the five re-trainings. Variation in performance is reasonably small for all models and CMP leads to significant improvements.

Ablations. We also studied ablated versions of our proposed method. We summarize the key takeaways, a learned mapper leads to better navigation performance than an analytic mapper, planning is crucial (specially for when using RGB images as input) and single-scale planning works slightly better than the multi-scale planning at the cost of increased planning cost. More details in supplementary material [62].

Additional comparisons between LSTM and CMP. We also conducted additional experiments to further compare the performance of the LSTM baseline with our model in the most competitive scenario where both methods use depth images. We summarize the key conclusions here and provide more details in supplementary material [62]. We studied how performance changes when the target is much further away (64 time steps away). We see a larger gap in performance between LSTM and CMP for this test scenarios. We also compared performance of CMP and LSTM over problems of different difficulty and observed that CMP is generally better across all values of hardness, but for RGB images it is particularly better for cases with high hardness. We also evaluate how well these models generalize when trained on a single scene, and when transferring across datasets. We find that there is a smaller drop in performance for CMP as compared to LSTM. More details in supplementary material [62]. Figure 3.5 visualizes and discusses some representative success and failure cases for CMP, video examples are available on the project website.

Comparison to purely geometric mapping. We also implemented a purely geometric incremental mapping and path planning policy. We projected observed 3D points incrementally into a

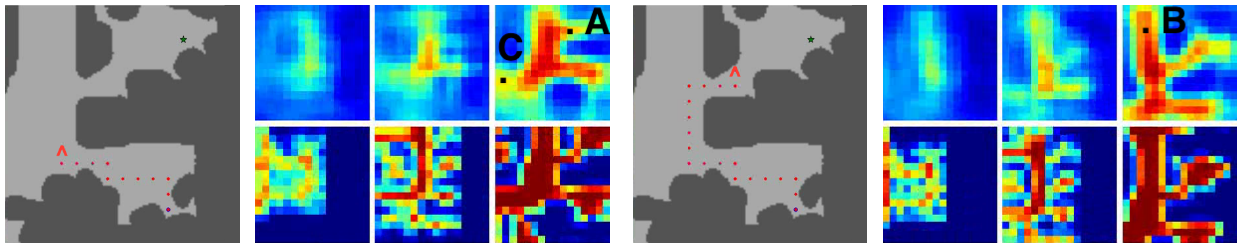


Figure 3.7: We visualize the output of the map readout function trained on the representation learned by the mapper (see text for details) as the agent moves around. The two rows show two different time steps from an episode. For each row, the gray map shows the current position and orientation of the agent (red $\wedge$), and the locations that the agent has already visited during this episode (red dots). The top three heatmaps show the output of the map readout function and the bottom three heatmaps show the ground truth free space at the three scales used by CMP (going from coarse to fine from left to right). We observe that the readout maps capture the free space in the regions visited by the agent (room entrance at point A, corridors at points B and C).

top-down occupancy map using the ground truth egomotion and camera extrinics and intrinsics. This occupancy map was used to compute a grid-graph. The policy outputs the action that leads the agent on the shortest path from the current location to the goal location on this grid-graph. We conducted this experiment with both depth and RGB images. When using RGB images as input, we triangulated SIFT feature points in the RGB images (registered using the ground truth egomotion) to obtain the observed 3D points (we used the COLMAP library [145]). We experimented with different image sizes, as well as increased the frequency at which RGB or depth images were captured. We plot the success rate in Figure 3.6 and compare to CMP policies that were trained and tested at 225×225 resolution with 1 image per time step. We see that when using depth images as input, this baseline performs similar to our method, but our approach performs much better when considering RGB images as input. We note that it took multiple minutes to detect and triangulate feature points per test case when using RGB images, and thus we report this evaluation only over a subset of 128 test cases.

3.6.2 Semantic Task

Here we present experiments where the target is specified semantically. We consider three tasks: ‘go to a chair’, ‘go to a door’ and ‘go to a table’. The agent receives a one-hot vector indicating the object category it must go to and is considered successful if it can reach any instance of the indicated object category. We use object annotations from the S3DIS dataset [8] to label nodes in the graph $\mathcal{G}_{x,\theta}$ with object categories. Note that this is only done to generate supervision for optimal actions during training and to evaluate the performance of the agents at test time. This supervision is not used by the agent in any way, it must learn appearance models for chairs jointly with the policy to reach them. We initialize the agent such that it is within 32 time steps of at least one instance of the indicated category, and train it to go towards the nearest instance. Table 3.1 (bottom) reports average and 75th %ile distance to nearest category instance and the success rate

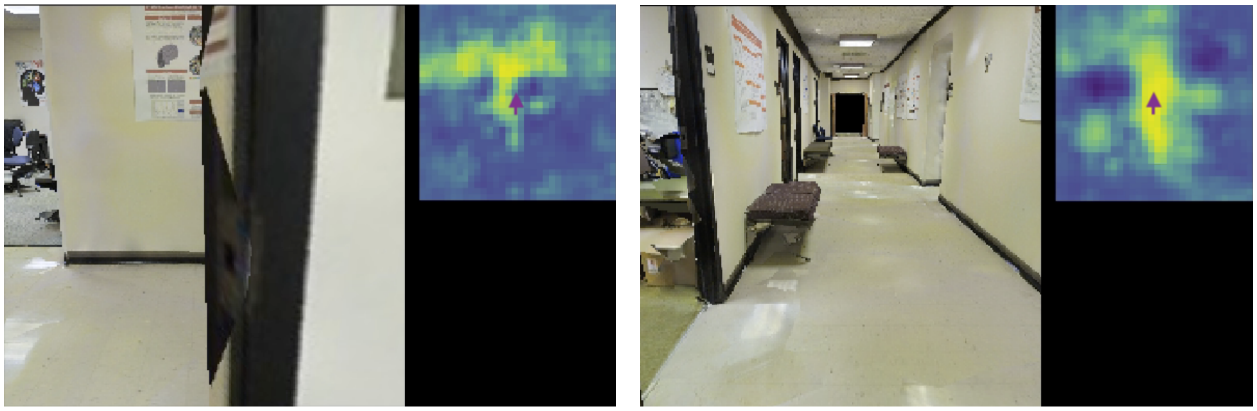


Figure 3.8: We visualize first-person images and the output of the readout function output for free-space prediction derived from the representation produced by the mapper module (in egocentric frame, that is the agent is at the center looking upwards (denoted by the purple arrow)). In the left example, we can make a prediction behind the wall, and in the right example, we can make predictions inside the room.

after executing a fixed number of steps (39 steps) across tasks from all three categories.

We compare our method to the best performing reactive and LSTM based baseline models from the geometric navigation task[1]. This is a challenging task specially because the agent may start in a location from which the desired object is not visible, and it must learn to explore the environment to find the desired object. CMP is able to achieve a higher success rate than the baselines. Figure 3.5 shows some sample trajectories for this task for CMP. Per category performance and more analysis is presented in supplementary material [62]. Performance is currently hindered by the limited ability of the network at recognizing objects and incorporating stronger appearance models may boost performance.

3.6.3 Visualizations

We visualize activations at different layers in the CMP network to check if the architecture conforms to the intuitions that inspired the design of the network. We check for the following three aspects: a) is the representation produced by the mapper indeed spatial, b) does the mapper capture anything beyond what a purely geometric mapping pipeline captures, and c) do the value maps obtained from the value iteration module capture the behavior exhibited by the agent.

Is the representation produced by the mapper spatial? We train simple readout functions on the learned mapper representation to predict free space around the agent. Figure 3.7 visualizes the output of these readout functions at two time steps from an episode as the agent moves. We see that the representation produced by the mapper is in correspondence with the actual free space

[1]This LSTM is impoverished because it no longer receives the egomotion of the agent as input (because the goal can not be specified as an offset relative to the robot). We did experiment with a LSTM model which received egomotion as input but weren't able to train it in initial experiments.

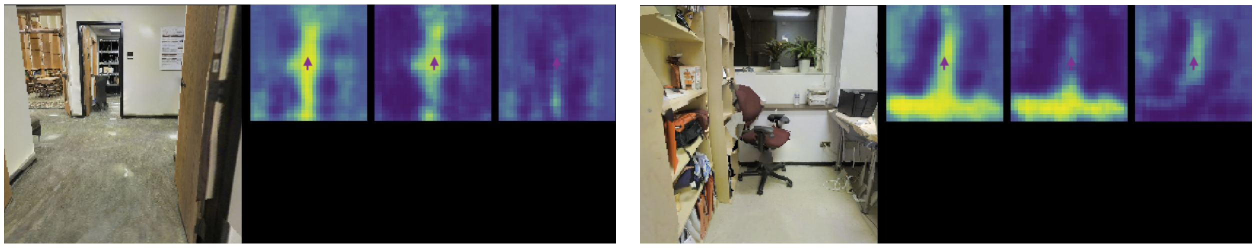


Figure 3.9: We visualize the first person image, prediction for all free space, prediction for free space in a hallway, and prediction for free space inside a room (in order). Once again, the predictions are in an egocentric coordinate frame (agent (denoted by the purple arrow) is at the center and looking upwards). The top figure pane shows the case when the agent is actually in a hallway, while the bottom figure pane shows the case when the agent is inside a room.

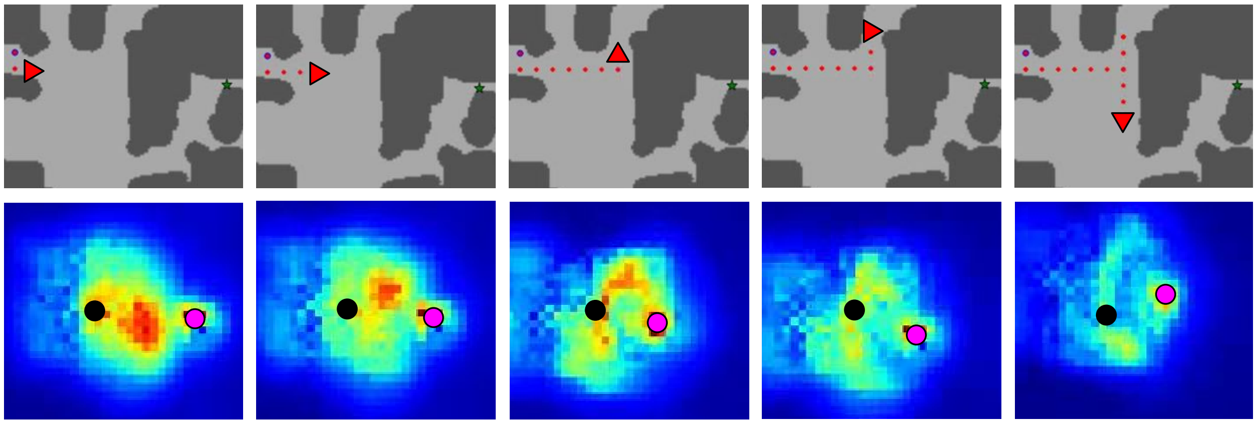


Figure 3.10: We visualize the value function for five snapshots for an episode for the single scale version of our model. The top row shows the agent's location and orientation with a red triangle, nodes that the agent has visited with red dots and the goal location with the green star. Bottom row shows a 1 channel projection of the value maps (obtained by taking the channel wise max) and visualizes the agent location by the black dot and the goal location by the pink dot. Initially the agent plans to go straight ahead, as it sees the wall it develops an inclination to turn left. It then turns into the room (center figure), planning to go up and around to the goal but as it turns again it realizes that that path is blocked (center right figure). At this point the value function changes (the connection to the goal through the top room becomes weaker) and the agent approaches the goal via the downward path.

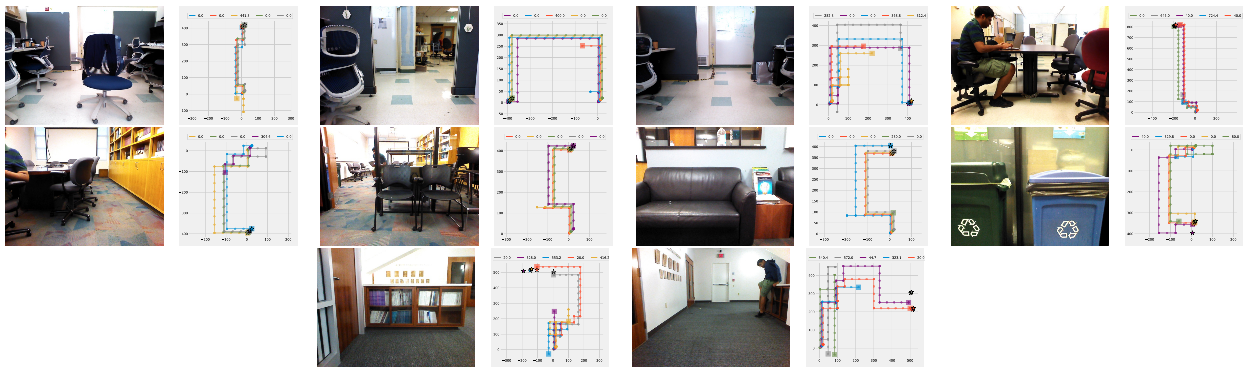


Figure 3.11: Real World Experiments: Images and schematic sketch of the executed trajectory for each of the 5 runs for the 10 test cases that were used to test the policy in the real world. Runs are off-set from each other for better visualization. Start location (always $(0, 0)$) is denoted by a solid circle, goal location by a start, and the final location of the agent is denoted by a square. Legend notes the distance of the goal location from the final position. Best seen in color on screen.

around the agent. The representation produced by the mapper is indeed spatial in nature. We also note that readouts are generally better at finer scales.

What does the mapper representation capture? We next try to understand as to what information is captured in these spatial representations. First, as discussed above the representation produced by the mapper can be used to predict free space around the agent. Note that the agent was never trained to predict free space, yet the representations produced by the mapper carry enough information to predict free space reasonable well. Second, Figure 3.8 shows free space prediction for two cases where the agent is looking through a doorway. We see that the mapper representation is expressive enough to make reasonable predictions for free space behind the doorway. This is something that a purely geometric system that only reasons about directly visible parts of the environment is simply incapable of doing. Finally, we show the output of readout functions that were trained for differentiating between free space in a hallway *vs.* free space in a room. Figure 3.9 (top) shows the prediction for when the agent is out in the hallway, and Figure 3.9 (bottom) shows the prediction for when the agent is in a room. We see that the representation produced by the mapper can reasonably distinguish between free space in a hallway *vs.* free space in a room, even though it was never explicitly trained to do so. Once again, this is something that a purely geometric description of the world will be unable to capture.

Do the value maps obtained from the value iteration module capture the behaviour exhibited by the agent? Finally, Figure 3.10 visualizes a one channel projection of the value map for the single scale version of our model at five time steps from an episode. We can see that the value map is indicative of the current actions that the agent takes, and how the value maps change as the agent discovers that the previously hypothesized path was infeasible.

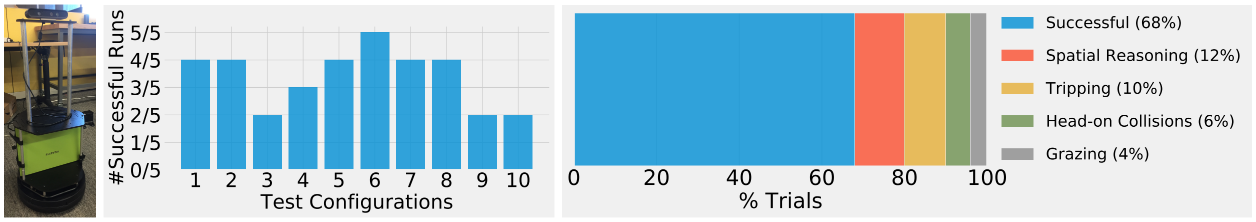


Figure 3.12: Real World Deployment: We report success rate on different test cases for real world deployment of our policy on TurtleBot 2. The policy was trained for the geometric task using RGB images in simulation. The right plot shows breakdown of runs. 68% runs succeeded, 20% runs failed due to infractions, and the remaining 12% runs failed as the agent was unable to go around obstacles.

3.7 Real World Deployment

We have also deployed these learned policies on a real robot. We describe the robot setup, implementation details and our results below.

Robot description. We conducted our experiments on a TurtleBot 2 robot. TurtleBot 2 is a differential drive platform based on the Yujin Kobuki Base. We mounted an Orbbec Astra camera at a height of $80cm$, and a GPU-equipped high-end gaming laptop (Gigabyte Aero 15“ with an NVIDIA 1060 GPU). The robot is shown in Figure 3.12 (left). We used ROS to interface with the robot and the camera. We read out images from the camera, and an estimate of the robot's $2D$ position and orientation obtained from wheel encoders and an on-board inertial measurement unit (IMU). We controlled the robot by specifying desired linear and angular velocities. These desired velocity commands are internally used to determine the voltage that is applied to the two motors through a proportional integral derivative (PID) controller. Note that TurtleBot 2 is a non-holonomic system. It only moves in the direction it is facing, and its dynamics can be approximated as a Dubins Car.

Implementation of macro-actions. Our policies output macro actions (rotate left or right by $90°$, move forward $40cm$). We implement these macro-actions using an iterative linear–quadratic regulator (iLQR) controller [88, 115]. iLQR leverages known system dynamics to output a dynamically feasible local reference trajectory (sequence of states and controls) that can convey the system from a specified initial state to a specified final state (in our case, rotation of $90°$ or forward motion of $40cm$). Additionally, iLQR is a state-space feedback controller. It estimates time-varying feedback matrices, that can adjust the reference controls to compensate for deviations from the reference trajectory (due to mis-match in system dynamics or noise in the environment). These adjusted controls are applied to the robot.

More formally, we use the robot 2D location and orientation as the state $\vec{s}$, the linear and angular velocity as the control inputs $\vec{u}$ to the system, and function f to model the dynamics of the system as follows:

$$\vec{s_t} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} \quad \vec{u_t} = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} \quad f(\vec{s_t}, \vec{u_t}) = \begin{bmatrix} x_t + v_t \Delta t \cos(\theta_t) \\ y_t + v_t \Delta t \sin(\theta_t) \\ \theta_t + \omega_t \Delta t \end{bmatrix} \tag{3.3}$$

Given an initial state $\vec{s_0}$, and a desired final state $\vec{s_T}$ ($= \vec{0}$ without loss of generality), iLQR solves the following optimization problem:

$$\underset{\vec{u_t}}{\arg\min} \qquad \textstyle\sum_t \vec{s_t}^t Q \vec{s_t} + \vec{u_t}^t R \vec{u_t} \tag{3.4}$$

$$\text{subject to} \quad \vec{s_{t+1}} = f(\vec{s_t}, \vec{u_t}) \text{for } t \in [1, \dots, T] \tag{3.5}$$

where, matrices Q and R are specified to be appropriately scaled identity matrices, Δt controls the frequency with which we apply the control input, and T determines the total time duration we have to finish executing the macro-action. Matrix Q incentives the system to reach the target state quickly, and matrix R incentives applying small velocities. The exact scaling of matrices Q and R, Δt and T are set experimentally by running the robot on a variety of start and goal state pairs.

Given Dubins Car dynamics are non-linear, iLQR optimizes the cost function by iteratively linearizing the system around the current solution. As mentioned, iLQR outputs $\vec{x_t^{ref}}$, $\vec{u_t^{ref}}$, and a set of feedback matrices K_t. The control to be applied to the system at time step t is obtained as $\vec{u_t^{ref}} + K_t \left(\tilde{\vec{s_t}} - \vec{s_t^{ref}} \right)$, where $\tilde{\vec{s_t}}$ is the estimated state of the system as measured from the robots wheel encoders and IMU (after appropriate coordinate transforms).

Policy. We deployed the policy for the geometric task onto the robot. As all other policies, this policy was trained entirely in simulation. However, to facilitate transfer to the real world, we a) trained the policy for longer (120K *vs.* 60K iterations), b) used aggressive data augmentation (varying the camera height and elevation, in addition to the usual color distortion), and c) used six additional environments from the MP3D dataset [22] (on top of the 4 environments from the S3DIS [8] dataset). These changes improved performance in simulation from 62.5% to 79.7% as well as exhibited better real world behavior in preliminary runs.

Results. We ran the robot in 10 different test configurations (shown in Figure 3.11). These tests were picked such that there was no straight path to the goal location, and involved situation like getting out of a room, going from one cubicle to another, and going around tables and kitchen counters. We found the depth as sensed from the Orbbec camera to be very noisy (and different from depth produced in our simulator), and hence only conducted experiments with RGB images as input. We conducted 5 runs for each of the 10 different test configurations, and report the success rate for the 10 configurations in Figure 3.12 (middle). A run was considered successful if the robot made it close to the specified target location (within $80cm$) without brushing against or colliding with any objects. Sample videos of execution are available on the project website. The policy achieved a success rate of 68%. Executed trajectories are plotted in Figure 3.11. This is a very encouraging result, given that the policy was trained entirely in simulation on very different buildings, and the lack of any form of domain adaptation. Our robot, that only uses monocular

RGB images, successfully avoids running into obstacles and arrives at the goal location for a number of test cases.

Figure 3.12 (right) presents failure modes of our runs. 10 of the 16 failures are due to infractions (head-on collisions, grazing against objects, and tripping over rods on the floor). These failures can possibly be mitigated by use of a finer action space for more dexterous motion, additional instrumentation such as near range obstacle detection, or coupling with a collision avoidance system. The remaining 6 failures correspond to not going around obstacles, possibly due to inaccurate perception.

3.8 Discussion

In this paper, we introduced a novel end-to-end neural architecture for navigation in novel environments. Our architecture learns to map from first-person viewpoints and uses a planner with the learned map to plan actions for navigating to different goals in the environment. Our experiments demonstrate that such an approach outperforms other direct methods which do not use explicit mapping and planning modules. While our work represents exciting progress towards problems which have not been looked at from a learning perspective, a lot more needs to be done for solving the problem of goal oriented visual navigation in novel environments.

A central limitations in our work is the assumption of perfect odometry. Robots operating in the real world do not have perfect odometry and a model that factors in uncertainty in movement is essential before such a model can be deployed in the real world.

A related limitation is that of building and maintaining metric representations of space. This does not scale well for large environments. We overcome this by using a multi-scale representation for space. Though this allows us to study larger environments, in general it makes planning more approximate given lower resolution in the coarser scales which could lead to loss in connectivity information. Investigating representations for spaces which do not suffer from such limitations is important future work.

In this work we have exclusively used DAGGER for training our agents. Though this resulted in good results, it suffers from the issue that the optimal policy under an expert may be unfeasible under the information that the agent currently has. Incorporating this in learning through guided policy search or reinforcement learning may lead to better performance specially for the semantic task.

Chapter 4

Cross Modal Distillation

Current paradigms for recognition in computer vision involve learning a generic feature representation on a large dataset of labeled images, and then specializing or finetuning the learned generic feature representation for the specific task at hand. Successful examples of this paradigm include almost all state-of-the-art systems: object detection [49], semantic segmentation [120], object segmentation [70], and pose estimation [173], which start from generic features that are learned on the ImageNet dataset [31] using over a million labeled images and specialize them for each of the different tasks. Several different architectures for learning these generic feature representations have been proposed over the years [105, 158], but all of these rely on the availability of a large dataset of labeled images to learn feature representations.

The question we ask in this work is, what is the analogue of this paradigm for images from modalities which do not have such large amounts of labeled data? There are a large number of image modalities beyond RGB images which are dominant in computer vision, for example depth images coming from a Microsoft Kinect, infra-red images from thermal sensors, aerial images from satellites and drones, LIDAR point clouds from laser scanners, or even images of intermediate representations output from current vision systems *e.g.* optical flow and stereo images. The number of labeled images from such modalities are at least a few orders of magnitude smaller than the RGB image datasets used for learning features, which raises the question: do we need similar large scale annotation efforts to learn generic features for images from each such different modality?

We answer this question in this paper and propose a technique to transfer learned representations from one modality to another. Our technique uses 'paired' images from the two modalities and utilizes the mid-level representations from the labeled modality to supervise learning representations on the paired un-labeled modality. We call our scheme ***supervision transfer*** and show that our learned representations perform well on standard tasks like object detection. We also show that our technique leads to learning useful feature hierarchies in the unlabeled modality, which can be improved further with finetuning, and are still complementary to representations in the source modality.

This chapter is based on work done with Judy Hoffman and Jitendra Malik, and is presented here primarily as it appeared at CVPR 2016 [59]. Statements about past work should be read with this context in mind.

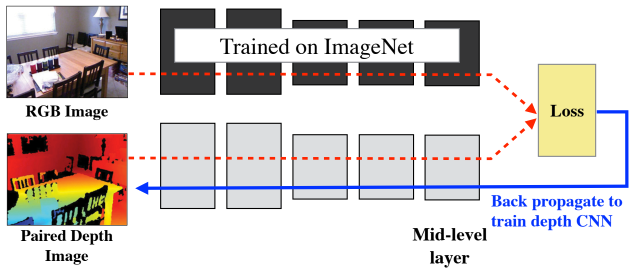


Figure 4.1: Architecture for supervision transfer: We train a CNN model for a new image modality (like depth images), by teaching the network to reproduce the mid-level semantic representations learned from a well labeled image modality (such as RGB images) for modalities for which there are paired images.

As a motivating example, consider the case of depth images. While the largest labeled RGB dataset, ImageNet [31] consists of over a million labeled images, the size of most existing labeled depth datasets is of the order of a few thousands [156, 162]. At the same time there are a large number of unlabeled RGB and depth image pairs. Our technique leverages this large set of unlabeled paired images to transfer the ImageNet supervision on RGB images to depth images. Our technique is illustrated in Figure 4.1. We use a convolutional neural network that has been trained on labeled images in the ImageNet dataset [31], and use the mid-level representation learned by these CNNs as a supervisory signal to train a CNN on depth images. This results in improvements in performance for the end task of object detection on the NYUD2 dataset, where we improve the state-of-the-art from 34.2% to 41.7% when using just the depth image and from 46.2% to 49.1% when using both RGB and depth images together. We report similar improvements for the task of simultaneous detection and segmentation [70] and also show how supervision transfer can be used for a zero-shot transfer of object detectors trained on RGB images to detectors that can run on depth images.

Though we show detailed experimental results for supervision transfer from RGB to depth images, our technique is equally applicable to images from other paired modalities. To demonstrate this, we show additional transfer results from RGB images to optical flow images where we improve mean average precision for action detection on the JHMDB dataset [91] from 31.7% to 35.7% when using just the optical flow image and no supervised pre-training.

Our technique is reminiscent of the distillation idea from Hinton *et al.* [80] (its recent extension FitNets by Romero *et al.* in [135], and its application to domain adaptation by Tzeng *et al.* in [174]). Hinton *et al.* [80] extended the model compression idea from Bucilua *et al.* [19] to what they call ‘distillation’ and showed how large models trained on large labeled datasets can be compressed by using the soft outputs from the large model as targets for a much smaller model operating on the same modality. Our work here is a generalization of this idea: we explore transfer of supervision at arbitrary semantic levels, and investigate how we can transfer supervision between *different* modalities using paired images. More importantly, our work allows us to extend the success of recent deep CNN architectures to new imaging modalities without having to collect large scale labeled datasets necessary for training deep CNNs.

4.1 Background

There has been a large body of work on transferring knowledge between different visual domains, belonging to the *same* modality. Initial work *e.g.* [11, 37, 53, 107] studied the problem in context of shallow image representations. More recently, with the introduction of supervised CNN models by Krizhevsky *et al.* [105], the community has been moving towards a generic set of features which are specialized to specific tasks and domains at hand [36, 49, 150] and traditional visual adaptation techniques can be used in conjunction with such features [47, 82, 121, 174].

All these lines of work study and solve the problem of domain adaptation within the same modality. In contrast, our work here tackles the problem of domain adaptation across *different* modalities. Most methods for intra-modality domain adaptation described above start from an initial set of features on the target domain, and *a priori* it is unclear how this can be done when moving across modalities, limiting the applicability of aforementioned approaches to our problem. This cross-model transfer problem has received much less attention. Notable among those include [24, 45, 128, 161, 165]. While [24, 165] hallucinate modalities during training time, [45, 128, 161] focus on the problem of jointly embedding or learning representations from multiple modalities into a shared feature space to improve learning [128] or enabling zero-shot learning[45, 161]. Our work here instead transfers high quality representations learned from a large set of labeled images of one modality to completely unlabeled images from a new modality, thus leading to a generic feature representations on the new modalities which we show are useful for a variety of tasks.

4.2 Supervision Transfer

Let us assume we have a modality $\mathcal{U}$ with unlabeled data, D_u for which we would like to train a rich representation. We will do so by transferring information from a separate modality, $\mathcal{L}$, which has a large labeled set of images, D_l, and a corresponding $\#_l$ layered rich representation. We assume this rich representation is layered although our proposed method will work equally well for non-layered representations. We use convolutional neural networks as our layered rich representation.

We denote this image representation as $\Phi = \{\phi^i \ \forall i \in \{1, \ldots, \#_l\}\}$. ϕ^i is the i^{th} layer representation for modality $\mathcal{L}$ which has been trained on labeled images from dataset D_l, and it maps an input image from modality $\mathcal{L}$ to a feature vector in $\mathbb{R}^{n_i}$

$$\phi^i : \mathcal{L} \mapsto \mathbb{R}^{n_i} \tag{4.1}$$

Feature vectors from consecutive layers in such layered representations are related to one another by simple operations like non-linearities, convolutions, pooling, normalizations and dot products (for example layer 2 features may be related to layer 1 features using a simple non-linearity like $\max$ with 0: $\phi^2(x) = \max(0, \phi^1(x))$). Some of these operations like convolutions and dot products have free parameters. We denote such parameters associated with operation at layer i by w_l^i. The structure of such architectures (the sequence of operations, and the size of representations at each layer, *etc.*) is hand designed or validated using performance on an end task. Such validation can be done on a small set of annotated images. Estimating the model parameters w_l^i is much more difficult. The number of these parameters for most reasonable image models can easily go up to a few millions, and state-of-the-art models employ discriminative learning and use large scale labeled training datasets.

Now suppose we want to learn a rich representation for images from modality $\mathcal{U}$, for which we do not have access to a large dataset of labeled images. We assume we have already hand designed an appropriate architecture $\Psi = \{\psi^i \ \forall i \in \{1, \ldots, \#_u\}\}$. The task then is to effectively learn the parameters associated with various operations in the architecture, without having access to a large set of annotated images for modality $\mathcal{U}$. As before, we denote these parameters to be learned by $W_u^{\{1,\ldots,\#_u\}} = \{w_u^i \ \forall i \in \{1, \ldots, \#_u\}\}$.

In addition to D_l, let us assume that we have access to a large dataset of *un-annotated paired* images from modalities $\mathcal{L}$ and $\mathcal{U}$. We denote this dataset by $P_{l,u}$. By paired images we mean a set of images of the same scene in two different modalities. Our proposed scheme for training rich representations for images of modality $\mathcal{U}$ is to learn the representation Ψ such that the image representation $\psi^{\#_u}(I_u)$ for image I_u matches the image representation $\phi^{i^*}(I_l)$ for its image pair I_l in modality l for some chosen and fixed layer $i^* \in \{1, \ldots, \#_l\}$. We measure the similarity between the representations using an appropriate loss function f (for example, euclidean loss). Note that the representations ϕ^{i^*} and $\psi^{\#_u}$ may not have the same dimensions. In such cases we embed features $\psi^{\#_u}$ into a space with the same dimension as ϕ^{i^*} using an appropriate simple transformation function t (for example a linear or affine function)

$$\min_{W_u^{\{1,\ldots,\#_u\}}} \sum_{(I_l, I_u) \in P_{l,u}} f\left(t\left(\psi^{\#_u}(I_u)\right),\ \phi^{i^*}(I_l)\right) \tag{4.2}$$

We call this process *supervision transfer* from layer i^* in Φ of modality $\mathcal{L}$ to layer $\#_u$ in Ψ of modality $\mathcal{U}$.

The recent distillation method from Hinton *et al.* [80] is a specific instantiation of this general method, where a) they focus on the specific case when the two modalities $\mathcal{L}$ and $\mathcal{U}$ are the same and b) the *supervision transfer* happens at the very last prediction layer, instead of an arbitrary internal layer in representation Φ.

Our experiments in Section 4.3 demonstrate that this proposed method for transfer of supervision is a) effective at learning good feature hierarchies, b) these hierarchies can be improved further with finetuning, and c) the resulting representation can be complementary to the representation in the source modality $\mathcal{L}$ if the modalities permit.

4.3 Experiments

In this section we present experimental results on 1) the NYUD2 dataset where we use color and depth images as the modality pairs, and 2) the JHMDB video dataset where we use the RGB and optical flow frames as the modality pairs.

Table 4.1: We evaluate different aspects of our *supervision transfer* scheme on the object detection task on the NYUD2 *val* set using the mAP metric. Left column demonstrates that our scheme for pre-training is better than alternatives like no pre-training, and copying over weights from RGB networks. The middle column demonstrates that our technique leads to transfer of mid-level semantic features which by themselves are highly discriminative, and that improving the quality of the supervisory network translated to improvements in the learned features. Finally, the right column demonstrates that the learned features on the depth images are still complementary to the features on the RGB image they were supervised with.

Does supervision transfer work?			How good is the transferred representation by itself?			Are the representations complementary?		
Exp. 1A	no init	22.7	Exp. 2A	copy from RGB (ft fc only)	19.8	Exp. 3A	[RGB]: RGB network on RGB images AlexNet	22.3
Exp. 1B	copy from RGB	25.1	Exp. 2B	supervision transfer (ft fc only) AlexNet * $\rightarrow$ AlexNet	30.0	Exp. 3B	[RGB] + copy from RGB	33.8
Exp. 1C	supervision transfer AlexNet $\rightarrow$ AlexNet	29.7	Exp. 2C	supervision transfer (ft fc only) VGG * $\rightarrow$ AlexNet	32.2	Exp. 3C	[RGB] + supervision transfer AlexNet * $\rightarrow$ AlexNet	35.6
Exp. 1D	supervision transfer AlexNet * $\rightarrow$ AlexNet	30.5	Exp. 2D	supervision transfer VGG * $\rightarrow$ AlexNet	33.6	Exp. 3D	[RGB]+ supervision transfer VGG * $\rightarrow$ AlexNet	37.0

Our general experimental framework consists of two steps. The first step is *supervision transfer* as proposed in Section 4.2, and the second step is to assess the quality of the transferred representation by using it for a downstream task. For both of the datasets we study, we consider the domain of RGB images as $\mathcal{L}$ for which there is a large dataset of labeled images in the form of ImageNet [31], and treat depth and optical flow respectively as $\mathcal{U}$. These choices for $\mathcal{L}$ and $\mathcal{U}$ are of particular practical significance, given the lack of large labeled datasets for depth images, at the same time, the abundant availability of paired images coming from RGB-D sensors (for example Microsoft Kinect) and videos on the Internet respectively.

For our layered image representation models, we use convolutional neural networks (CNNs) [105, 112]. These networks have been shown to be very effective for a variety of image understanding tasks [36]. We experiment with the network architectures from Krizhevsky *et al.* [105] (denoted AlexNet), Simonyan and Zisserman [158] (denoted VGG), and use the models pre-trained on ImageNet [31] from the Caffe [92] Model Zoo.

We use an architecture similar to [105] for the layered representations for depth and flow images. We do this in order to be able to compare to past works which learn features on depth and flow images [52, 64]. Validating different CNN architectures for depth and flow images is a worthwhile scientific endeavor, which has not been undertaken so far, primarily because of lack of large scale labeled datasets for these modalities. Our work here provides a method to circumvent the need for a large labeled dataset for these and other image modalities, and will naturally enable exploring this question in the future, however we do not delve in this question in the current work.

We next describe our design choices for which layers to transfer supervision between, and the specification of the loss function f and the transformation function t. We experimented with what layer to use for transferring supervision, and found transfer at mid-level layers works best, and use the last convolutional layer pool5 for all experiments in the paper. Such a choice also resonates well with observations from [12, 113, 180] that lower layers in CNNs are modality specific (and thus harder to transfer across modalities) and visualizations from [49] that neurons in mid-level layers are semantic and respond to parts of objects. Transferring at pool5 also has the computational

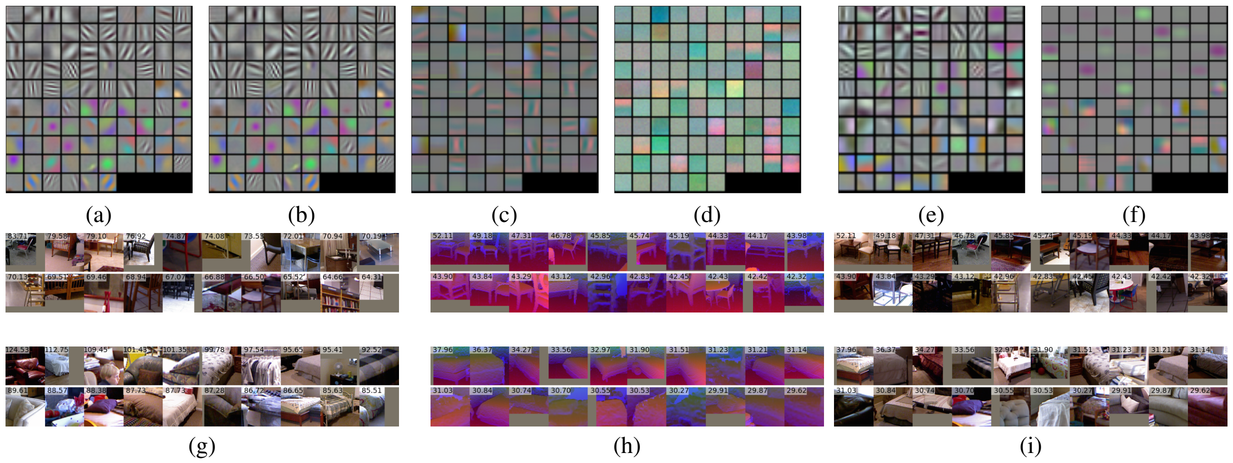


Figure 4.2: Visualization of learned filters (best viewed in color): (a) visualizes filters learned on RGB images from ImageNet data by AlexNet. (b) shows these filters after the finetuning on HHA images, and hardly anything changes visually. (c) shows HHA image filters from our pre-training scheme, which are much different from ones that are learned on RGB images. (d) shows HHA image filters learned without any pre-training. (e) shows optical flow filters learned by [52]. Note that they initialize these filters from RGB filters and these also do not change much over their initial RGB filters. (f) shows filters we learn on optical flow images, which are again very different from filters learned on RGB or HHA images. (g) shows image patches corresponding to highest scoring activations for two neurons in the RGB CNN. (h) shows HHA image patches corresponding to highest scoring activations of the same neuron in the supervision transfer depth CNN. (i) shows the corresponding RGB image patch for these depth image patches for ease of visualization.

benefit that training can be efficiently done in a fully convolutional manner over the whole image.

For the function f, we use $L2$ distance between the feature vectors, $f(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2$. We also experimented with $f(\mathbf{x}, \mathbf{y}) = \mathbf{1}(\mathbf{y} > \tau) \cdot \log p(\mathbf{x}) + \mathbf{1}(\mathbf{y} \leq \tau) \cdot \log(1 - p(\mathbf{x}))$ (where $p(x) = \frac{e^{\alpha x}}{1+e^{\alpha x}}$, $\mathbf{1}(x)$ is the indicator function), for some reasonable choices of α and τ but this resulted in worse performance in initial experiments.

Finally, the choice of the function t varies with different pairs of networks. As noted above, we train using a fully convolutional architecture. This requires the spatial resolution of the two layers i^* in Φ and $\#_u$ in Ψ to be similar, which is trivially true if the architectures Φ and Ψ are the same. When they are not (for example when we transfer from VGG net to AlexNet), we adjust the padding in the AlexNet to obtain the same spatial resolution at pool5 layer.

This apart, we introduce an adaptation layer comprising of 1×1 convolutions followed by $ReLU$ to map from the representation at layer $\#_u$ in Ψ to layer i^* in Φ. This accounts for difference in the number of neurons (for example when adapting from VGG to AlexNet), or even when the number of neurons are the same, allows for domain specific fitting. For VGG to AlexNet transfer we also needed to introduce a scaling layer to make the average norm of features comparable between the two networks.

4.3.1 Transfer to Depth Images

We first demonstrate how we transfer supervision from color images to depth images as obtained from a range sensor like the Microsoft Kinect. As described above, we do this set of experiments on the NYUD2 dataset [156] and show results on the task of object detection and instance segmentation [64]. The NYUD2 dataset consists of 1449 paired RGB and depth images. These images come from 464 different scenes and were selected from the full video sequence to ensure diverse scene content [156]. The full video sequence that comes with the dataset has over 400K RGB-D frames, we use 10K of these frame pairs for supervision transfer.

In all our experiments we report numbers on the standard *val* and *test* splits that come with the dataset [64, 156]. Images in these splits have been selected while ensuring that all frames belonging to the same scene are contained entirely in exactly one split. We additionally made sure only frames from the corresponding training split were used for supervision transfer.

The downstream task that we study here is that of object detection. We follow the experimental setup from Gupta *et al.* [64] for object detection and study the 19 category object detection problem, and use mean average precision (mAP) to measure performance.

Baseline Detection Model We use the model from Gupta *et al.* [64] for object detection. Their method builds off R-CNN [49]. In our initial experiments we adapted their model to the more recent Fast R-CNN framework [50]. We summarize our key findings here. First, [64] trained the final detector on both RGB and depth features jointly. We found training independent models all the way and then simply averaging the class scores before the SoftMax performed better. While this is counter-intuitive, we feel it is plausible given limited amount of training data. Second, [64] use features from the fc6 layer and observed worse performance when using fc7 representation; in our framework where we are training completely independent detectors for the two modalities, using fc7 representation is better than using fc6 representation. Finally, using bounding box regression boosts performance. Here we simply average the predicted regression target from the detectors on the two modalities. All this analysis helped us boost the mean AP^b on the *test* set from 38.80% as reported by [60, 64] to 44.39%, using the same CNN network and supervision. This already is the state-of-the-art result on this dataset and we use this as a baseline for the rest of our experiments. We denote this model as '[64] + Fast R-CNN'. We followed the default setup for training Fast R-CNN, $40K$ iterations, base learning rate of 0.001 and stepping it down by a factor of 10 after $30K$ iterations, except that we finetune all the layers, and use 688px length for the shorter image side. We used RGB-D box proposals from [64] for all experiments.

Note that Gupta *et al.* [64] embed depth images into a geocentric embedding which they call HHA (HHA encodes horizontal disparity, height above ground and angle with gravity) and use the AlexNet architecture to learn HHA features and *copy over the weights from the RGB CNN that was trained for 1000 way classification [105] on ImageNet [31]* to initialize this network. All through this paper, we stick with using HHA embedding[1] to represent the input depth images, and their network architecture, and show how our proposed *supervision transfer* scheme improves performances over their technique for initialization. We summarize our various transfer experiments below:

[1]We use the term depth and HHA interchangeably.

Does *supervision transfer* work? The first question we investigate is if we are able to transfer supervision to a new modality. To understand this we conducted the following three experiments:

1. no init (1A): randomly initialize the depth network using weight distributions typically used for training on ImageNet and simply train this network for the final task. While training this network we train for 100K iterations, start with a learning rate on 0.01 and step it down by a factor of 10 every 30K iterations.

2. copy from RGB (1B): copy weights from a RGB network that was trained on ImageNet. This is same as the scheme proposed in [64]. This network is then trained using the standard Fast R-CNN settings.

3. *supervision transfer* (1C): train layers conv1 through pool5 from random initialization using the *supervision transfer* scheme as proposed in Section 4.2, on the 5K paired RGB and depth images from the video sequence from NYUD2 for scenes contained in the training set. We then plug in these trained layers along with randomly initialized fc6, fc7 and classifier layers for training with Fast R-CNN. We report the results in Table 4.1. We see that 'copy from RGB' surprisingly does better than 'no init', which is consistent with what Gupta *et al.* report in [64], but our scheme for *supervision transfer* outperforms both these baselines by a large margin pushing up mean AP from 25.1% to 29.7%. We also experimented with using a RGB network Ψ that has been adapted for object detection on this dataset for supervising the transfer (1D) and found that this boosted performance further from 29.7% to 30.5% (1D in Table 4.1, AlexNet* indicates RGB AlexNet that has been adapted for detection on the dataset). We use this scheme for all subsequent experiments.

Visualizations. We visualize the filters from the first layer for these different schemes of transfer in Figure 4.2(a-f), and observe that our training scheme learns reasonable filters and find that these filters are of different nature than filters learned on RGB images. In contrast, note that schemes which initialize depth CNNs with RGB CNNs weights, filters in the first layer change very little. We also visualize patches giving high activations for neurons paired across RGB and depth images Figure 4.2(g-i). High scoring patches from RGB CNN (AlexNet in this case), correspond to parts of object (g), high scoring patches from the depth CNN also corresponds to parts of the same object class (h and i).

How good is the transferred representation by itself? The next question we ask is if our *supervision transfer* scheme transfers good representations or does it only provide a good initialization for feature learning. To answer this question, we conducted the following experiments:

1. Quality of transferred pool5 representation (2A, 2B): The first experiment was to evaluate the quality of the transferred pool5 representation. To do this, we froze the network parameters for layers conv1 through pool5 to be those learned during the transfer process, and only learn parameters in fc6, fc7 and classifier layers during Fast R-CNN training (2B 'supervision transfer adapted (ft fc only)'). We see that there is only a moderate degradation in performance for our learned features from 30.5% (1D) to 30.0% (2B) indicating that the features learned on depth images at pool5 are discriminative by themselves. In contrast, when freezing weights when copying from ImageNet (2A), performance degrades significantly to 19.8%.

2. Improved transfer using better supervising network Φ (2C, 2D): The second experiment investigated if performance improves as we improve the quality of the supervising network. To do

Table 4.2: Region detection average precision AP^r on NYUD2 *val* set: Performance on NYUD2 *val* set where we observe similar boosts in performance when using hyper-column transform with our learned feature hierarchies (learned using supervision transfer on depth images) as obtained with more standard feature hierarchies learned on ImageNet on RGB images.

val	AP^r at 0.5		AP^r at 0.7	
	fc7	+pool2+conv4	fc7	+pool2+conv4
RGB	26.3	29.8	14.8	18.3
depth	28.4	31.5	17.4	19.6

Table 4.3: Region detection average precision on NYUD2 *test* set.

test	modality	RGB Arch.	Depth Arch.	AP^r at 0.5	AP^r at 0.7
[69]	RGB	AlexNet	-	23.4	13.4
Gupta *et al.* [60]	RGB + depth	AlexNet	AlexNet	37.5	21.8
Our (*supervision transfer*)	RGB + depth	AlexNet	AlexNet	**40.5**	**25.4**
[69]	RGB	VGG	-	31.0	17.7
Our (*supervision transfer*)	RGB + depth	VGG	AlexNet	**42.1**	**26.9**

this, we transferred supervision from VGG net instead of AlexNet (2C)[2]. VGG net has been shown to be better than AlexNet for a variety of vision tasks. As before we report performance when freezing parameters till pool5 (2C), and learning all the way (2D). We see that using a better supervising net results in learning better features for depth images: when the representation is frozen till pool5 we see performance improves from 30.0% to 32.2%, and when we finetune all the layers performance goes up to 33.6% as compared to 30.5% for AlexNet.

Is the learned representation complementary to the representation on the source modality? The next question we ask is if the representation learned on the depth images complementary to the representation on the RGB images from which it was learned. To answer this question we look at the performance when using both the modalities together. We do this the same way that we describe for the baseline model and simply average the category scores and regression targets from the RGB and depth detectors. Table 4.1(right) reports our findings. Just using RGB images (3A) gives us a performance of 22.3%. Combining this with the HHA network as initialized using the scheme from Gupta *et al.* [64] (3B) boosts performance to 33.8%. Initializing the HHA network using our proposed supervision transfer scheme when transferring from AlexNet* to AlexNet (3C) gives us 35.6% and when transferring from VGG* to AlexNet (3D) gives us 37.0%. These results show that the representations are still complementary and using the two together can help the final performance.

Transfer to other architectures. We also conducted preliminary experiments of transferring supervision from RGB VGG to a depth VGG network, and found a performance of 33.5% (RGB only VGG performance on the *val* set is 28.0%). Thus, *supervision transfer* can be used to transfer

[2] To transfer from VGG to AlexNet, we use $150K$ transfer iterations instead of $100K$. Running longer helps for VGG to AlexNet transfer by 1.5% and much less (about 0.5%) for AlexNet to AlexNet transfer.

supervision to different target architectures.

Does supervision transfer lead to meaningful intermediate layer representations? The next questions we investigate is if the intermediate layers learned in the target modality $\mathcal{U}$ through *supervision transfer* carry useful information. [100] hypothesize that information from intermediate layers in such hierarchies carry information which may be useful for fine grained tasks. Recent work as presented in [69, 120, 151] operationalize this and demonstrate improvements for fine grained tasks like object and part segmentation. Here we investigate if the representations learned using *supervision transfer* also share this property. To test this, we follow the hyper-column architecture from Hariharan *et al.* [69] and study the task of simultaneous detection and segmentation (SDS) [70] and investigate if the use of hyper-columns with our trained networks results in similar improvements as obtained when using more traditionally trained CNNs. We report the results in Table 4.2. On the NYUD2 dataset, the hyper-column transform improves AP^r from 26.3% to 29.8% when using AlexNet for RGB images. We follow the same experimental setup as proposed in [68], and fix the CNN parameters (to a network that was finetuned for detection on NYUD2 dataset) and only learn the classifier parameters and use features from pool2 and conv4 layers in addition to fc7 for figure ground prediction. When doing the same for our *supervision transfer* network we observe a similar boost in performance from 28.4% to 31.5% when using the hyper-column transform. This indicates that models trained using *supervision transfer* not only learn good representations at the point of supervision transfer (pool5 in this case), but also in the intermediate layers of the net.

How does performance vary as the transfer point is changed? We now study how performance varies as we vary the layer used for supervision transfer. We stick to the same experimental setup as used for Exp. 1D in Table 4.1, and conduct supervision transfer at different layers of the network. Layers above the transfer point are initialized randomly and learned during detector training. For transferring features from layers 1 to 5, we use fully convolutional training as before. But when transferring fc6 and fc7 features we compute them over bounding box proposals (we use RGB-D MCG bounding box proposals [64]) using Spatial Pyramid Pooling on conv5 [50, 74].

We report the obtained AP^b on the NYUD2 *val* set in Table 4.4. We see performance is poor when transferring at lower layers (pool1 and pool2). Transfer at layers conv3, conv4, pool5, fc6 works comparably, but performance deteriorates when moving to further higher layers (fc7). This validates our choice for using an intermediate layer as a transfer point. We believe the drop in performance at higher layers is an artifact of the amount of data used for supervision transfer. With a richer and more diverse dataset of paired images we expect transfer at higher layers to work similar or better than transfer at mid-layers. Explained variance during supervision transfer is also higher for transfers at layers 3, 4, and 5 than other layers.

We also conducted some initial experiments with using multiple transfer points. When transferring at conv3 and fc7 we observe performance improves over transferring at either layer alone, indicating learning is facilitated when supervision is closer to parameters being learned. We defer exploration of other choices in this space for future work.

Is input representation in the form of HHA images still important? Given our tool for training CNNs on depth images, we can now investigate the question whether hand engineering the input representation is still important. We conduct an experiment in exactly the same settings as

Table 4.4: Mean AP^b on NYUD2 *val* set as a function of layer used for supervision transfer.

pool1	pool2	conv3	conv4	pool5	fc6	fc7	conv3 + fc7
24.4	28.4	30.6	29.9	30.5	29.7	27.7	31.3

Table 4.5: Adapting RGB object detectors to RGB-D images: We transfer object detectors trained on RGB images (on MS COCO dataset) to RGB-D images in the NYUD2 dataset, without using any annotations on depth images. We do this by learning a model on depth images using supervision transfer and then use the RGB object detector trained on the representation learned on depth images. We report detection AP(%) on NYUD2 *test* set. These transferred detectors work well on depth images even without using any annotations on depth images. Combining predictions from the RGB and depth image improves performance further.

	Train on MS COCO and adapt to NYUD2 using supervision transfer								Train on NYUD2
	bed	chair	sink	sofa	table	tv	toilet	mean	mean
RGB	51.6	26.6	25.1	43.1	14.4	12.9	57.5	33.0	35.7
depth	59.4	27.1	23.8	32.2	13.0	13.6	43.8	30.4	45.0
RGB + depth	**60.2**	**35.3**	**27.5**	**48.2**	**16.5**	**17.1**	**58.1**	**37.6**	**54.4**

Exp. 1D except that we work with disparity images (replicated to have 3 channels) instead of HHA images. This gives a mAP of 29.2% as compared to 30.5% for the HHA images. The difference in performance is smaller than what [64] reports but still significant (14 of 19 categories improve), which suggests that encoding the depth image into a geocentric coordinate frame using the HHA embedding is still useful.

Applications to zero-shot detection on depth images. *Supervision transfer* can be used to transfer detectors trained on RGB images to depth images. We do this by the following steps. We first train detectors on RGB images. We then split the network into two parts at an appropriate mid-level point to obtain two networks Γ_{rgb}^{lower} and Γ_{rgb}^{upper}. We then use the lower domain specific part of the network Γ_{rgb}^{lower} to train a network Γ_{d}^{lower} on depth images to generate the same representation as the RGB network Γ_{rgb}^{lower}. This is done using the same supervision transfer procedure as before on a set of unlabeled paired RGB-D images. We then construct a 'franken' network with the lower domain specific part coming from Γ_{d}^{lower} and the upper more semantic network coming from Γ_{rgb}^{upper}. We then simply use the output of this franken network on depth images to obtain zero-shot object detection output.

More specifically, we use Fast R-CNN with AlexNet CNN to train object detectors on the MS COCO dataset [119]. We then split the network right after the convolutional layers pool5, and train a network on depth images to predict the same pool5 features as this network on unlabeled RGB-D images from the NYUD2 dataset (using frames from the *trainval* video sequences). We study all 7 object categories that are shared between MS COCO and NYUD2 datasets, and report the performance in Table 4.5. We observe our zero-shot scheme for transferring detectors across modalities works rather well. While the RGB detector trained on MS COCO obtains a mean AP^b

Table 4.6: Object detection mean AP(%) on NYUD2 *test* set: We compare our performance against several state-of-the-art methods. RGB Arch. and depth Arch. refers to the CNN architecture used by the detector. We see when using just the depth image, our method is able to improve performance from 34.2% to 41.7%. When used in addition to features from the RGB image, our learned features improve performance from 44.4% to 47.1% (when using AlexNet RGB features) and from 46.2% to 49.1% (when using VGG RGB features) over past methods for learning features from depth images. We see improvements across almost all categories, performance on individual categories is tabulated in supplementary material.

method	modality	RGB Arch.	depth Arch.	mAP
Fast R-CNN [50]	RGB	AlexNet	-	27.8
Fast R-CNN [50]	RGB	VGG	-	**38.8**
Gupta *et al.* [64]	RGB + depth	AlexNet	AlexNet	38.8
Gupta *et al.* [60]	RGB + depth	AlexNet	AlexNet	41.2
Gupta *et al.* [64] + Fast R-CNN	RGB + depth	AlexNet	AlexNet	44.4
Our (*supervision transfer*)	RGB + depth	AlexNet	AlexNet	**47.1**
Gupta *et al.* [64] + Fast R-CNN	RGB + depth	VGG	AlexNet	46.2
Our (*supervision transfer*)	RGB + depth	VGG	AlexNet	**49.1**
Gupta *et al.* [64] + Fast R-CNN	depth	-	AlexNet	34.2
Our (*supervision transfer*)	depth	-	AlexNet	**41.7**

of 33.0% on these categories, our zero-shot detector on depth images performs comparably and has a mean AP^b of 30.4%. Note that in doing so we have not used any annotations from the NYUD2 dataset (RGB or depth images). Furthermore, combining predictions from RGB and depth object detectors results in boost over just using the detector on the RGB image giving a performance of 37.6%. Performance when training detectors using annotations from the NYUD2 dataset (Table 4.5 last column) is much higher as expected. This can naturally be extended to incorporate annotations from auxiliary categories as explored in [83], but we defer this to future work.

Performance on test set. Finally, we report the performance of our best performing supervision transfer scheme (VGG $^* \rightarrow$ AlexNet) on the *test* set in Table 4.6. When used with AlexNet for obtaining color features, we obtain a final performance of 47.1% which is about 2.7% higher than the current state-of-the-art on this task (Gupta *et al.* [64] Fast R-CNN). We see similar improvements when using VGG for obtaining color features (46.2% to 49.1%). The improvement when using just the depth image is much larger, 41.7% for our final model as compared to 34.2% for the baseline model which amounts to a 22% relative improvement. Note that in obtaining these performance improvements we are using exactly the same CNN architecture and amount of labeled data. We also report performance on the SDS task in Table 4.3 and obtain state-of-the-art performance of 40.5% as compared to previous best 37.5% [60] when using AlexNet, using VGG CNN for the RGB image improves performance further to 42.1%.

Training Time. Finally, we report the amount of time it takes to learn a model using supervision transfer. For AlexNet to AlexNet supervision transfer we trained for 100K iterations which took a total of 2.5 hours on a NVIDIA k40 GPU. This is a many orders of magnitude faster than

Table 4.7: Action Detection AP(%) on the JHMDB *test* set: We report action detection performance on the *test* set of JHMDB using RGB or flow images. Right part of the table compares our method *supervision transfer* against the baseline of random initialization, and the ceiling using fully supervised pre-training method from [52]. Our method reaches more than half the way towards fully supervised pre-training.

	RGB		optical flow			
	[52]	[52] + [50]	[52] Sup PreTr	[52] + [50] Sup PreTr	Random Init No PreTr	Our Sup Transfer
mAP	27.0	**32.0**	24.3	**38.4**	31.7	35.7

training models from random initialization on ImageNet scale data using class labels.

4.3.2 Transfer to Flow Images

We now report our experiments for transferring supervision to optical flow images. We consider the end task of action detection on the JHMDB dataset. The task is to detect people doing actions like catch, clap, pick, run, sit in frames of a video. Performance is measured in terms of mean average precision as in the standard PASCAL VOC object detection task and what we used for the NYUD2 experiments in Section 4.3.1.

A popular technique for getting better performance at such tasks on video data is to additionally use features computed on the optical flow between the current frame and the next frame [52, 157]. We use *supervision transfer* to learn features for optical flow images in this context.

Detection model For JHMDB we use the experimental setup from Gkioxari and Malik [52] and study the 21 class task. Here again, Gkioxari and Malik build off of R-CNN and we first adapt their system to use Fast R-CNN, and observe similar boosts in performance as for NYUD2 when going from R-CNN to Fast R-CNN framework (Table 4.7, full table with per class performance is in the supplementary material). We denote this model as [52]+[50]. We attribute this large difference in performance to a) bounding box regression and b) number of iterations used for training.

***Supervision transfer* performance** We use the videos from UCF 101 dataset [164] for our pre-training. Note that we do not use any labels provided with the UCF 101 dataset, and simply use the videos as a source of paired RGB and flow images. We take 5 frames from each of the $9K$ videos in the *train1* set. We report performance on JHMDB *test* set in Table 4.7. Note that JHMDB has 3 splits and as in past work, we report the AP averaged across these 3 splits.

We report performance for three different schemes for initializing the flow model: a) **Random Init** (No PreTr) when the flow network is initialized randomly using the weight initialization scheme used for training a RGB model on ImageNet, b) **Supervised Pre-training** ([52]+[50] Sup PreTr) on flow images from UCF 101 for the task of video classification starting from RGB weights as done by Gkioxari and Malik [52] and c) **supervision transfer** (Our Sup Transfer) from an RGB model to train optical flow model as per our proposed method. We see that our scheme for *supervision transfer* improves performance from 31.7% achieved when using random initialization to 35.7%, which is more than half way towards what fully supervised pre-training can achieve (38.4%), thereby illustrating the efficacy of our proposed technique.

Chapter 5

Conclusion

In this thesis, we argued for revisiting the connection between computer vision and robotics. We showed that thinking about computer vision in context of robotics motivates new tasks in computer vision. We showed that recent advances in representation learning in computer vision can be extended to obtain representations for sensing modalities useful in robotics. We also demonstrated that jointly learning perception and action modules leads to more efficient and effective solutions.

This thesis presents only a few initial steps towards reviving the link between computer vision and robotics. There are a number of other ways in which a better understanding of the world will improve how agents interact with the unstructured world around them. But perhaps more importantly, mobile agents that can interact with the world will help scale up visual learning.

Learning for Moving in the World. In this work, we used low-level controllers for implementing policies onto real robots. While this modularity allowed us to quickly deploy our policies into the real world, this strict separation between high-level planning and low-level motor control is limiting. It resulted in unnatural stop-and-go motion and failure to adjust the high-level plan based on feedback from the low-level control. Future research will need to visit this connection between high-level planning and low-level control.

A central question in building agents that can successfully move around is that of how to represent space and how to efficiently build these representations for previously unseen environments. Our work, in this thesis, investigated metric (but abstract) representations of space. While this performed well on the task of going from one room to another under perfect odometry assumptions, it does not scale well to large environments. Topological representations are attractive in their robustness to imperfect odometry, but they do not allow spatial reasoning to find shortcuts. Future research will need to investigate representations for space that allow for spatial reasoning, are robust to registration error, and can scale to large environments.

Future research will also need to study *navigation in dynamic environments* with inanimate objects (that can be made to move) and animate other agents (that can move at their own will). Object-centered 3D representations for scenes (such as those pursued in this thesis) and techniques for understanding humans and their actions will form important substrate for tackling these problem. At the same time, memory representations that can *track the state of the world over time* will be crucial as objects and agents undergo occlusions while they move around. Beyond these technical challenges, a somewhat bigger interdisciplinary question is that of the design of *socially acceptable* artificial mobility.

Artificial Embodied Cognition. As we think about a future with artificial robotic agents embodied in the real world, a number of new research problems will emerge. Most current high-performing computer vision systems are limited by the data that is available to train them. This data is limited as it needs to be gathered by humans. As artificial robotic agents become mobile, they can be instrumented to scale-up data gathering, both in terms of quantity and richness. As we acquire more and more data, it will no longer be feasible for humans to manually annotate this data. Thus, unsupervised learning, self-supervised learning, and cross-modal learning will become important. Simultaneous access to multiple modalities plays a central role in bootstrapping learning in children [159]. It will be interesting to investigate if representations in computer vision can similarly be bootstrapped when given access to rich multi-sensory observation of the environment that can be gathered through mobile agents that move and interact with the world around them.

Moreover, rather than passively collecting and analyzing such datasets, machine learning algorithms will have the opportunity to actively collect their own data. It will require designing algorithms that *know what they don't know* and can design real-world behaviors to *teach themselves what they don't know*. Embodied agents that continuously learn by themselves by interacting with the world will lead to a paradigm shift in the way computer vision problems are thought of today.

And perhaps, computer vision will transform yet again to get back together with its old buddy, robotics, and evolve from being Internet Computer Vision to *Embodied Computer Vision*. These are exciting times in computer vision and artificial intelligence. I think the best is yet to come, and I look forward to it.

Bibliography

[1] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[2] D. Abel, A. Agarwal, F. Diaz, A. Krishnamurthy, and R. E. Schapire. Exploratory Gradient Boosting for Reinforcement Learning in Complex Domains. In: *arXiv preprint arXiv: 1603.04119* (2016).

[3] P. Ammirato, P. Poirson, E. Park, J. Kosecka, and A. C. Berg. A Dataset for Developing and Benchmarking Active Vision. In: *ICRA*. 2017.

[4] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. v. d. Hengel. Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In: *CVPR*. 2018.

[5] P. Arbeláez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. Multiscale Combinatorial Grouping. In: *CVPR*. 2014.

[6] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik. Contour Detection and Hierarchical Image Segmentation. In: *TPAMI* (2011).

[7] P. Arbeláez, B. Hariharan, C. Gu, S. Gupta, L. Bourdev, and J. Malik. Semantic segmentation using regions and parts. In: *CVPR*. 2012.

[8] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3D Semantic Parsing of Large-Scale Indoor Spaces. In: *CVPR*. 2016.

[9] M. Aubry, D. Maturana, A. Efros, B. Russell, and J. Sivic. Seeing 3D chairs: exemplar part-based 2D-3D alignment using a large dataset of CAD models. In: *CVPR*. 2014.

[10] A. Aydemir, A. Pronobis, M. Göbelbecker, and P. Jensfelt. Active visual object search in unknown environments using uncertain semantics. In: *IEEE Transactions on Robotics* (2013).

[11] Y. Aytar and A. Zisserman. Tabula Rasa: Model Transfer for Object Category Detection. In: *ICCV*. 2011.

[12] H. Azizpour, A. Razavian, J. Sullivan, A. Maki, and S. Carlsson. From generic to specific deep representations for visual recognition. In: *CVPR Workshops*. 2015.

[13] D. Banica and C. Sminchisescu. CPMC-3D-O2P: Semantic segmentation of RGB-D images using CPMC and Second Order Pooling. In: *CoRR* abs/1312.7715 (2013).

[14] R. Bellman. *A Markovian decision process*. Tech. rep. DTIC Document, 1957.

[15] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N Siddharth, and P. H. Torr. Playing Doom with SLAM-Augmented Deep Reinforcement Learning. In: *arXiv preprint arXiv: 1612.00380* (2016).

[16] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, and D. Hassabis. Model-free episodic control. In: *arXiv preprint arXiv:1606.04460* (2016).

[17] L. Bo, X. Ren, and D. Fox. Learning hierarchical sparse features for RGB-(D) object recognition. In: *IJRR* (2014).

[18] L. Breiman. Random Forests. In: *Machine Learning* (2001).

[19] C. Bucilua, R. Caruana, and A. Niculescu-Mizil. Model compression. In: *ACM SIGKDD*. 2006.

[20] J. Canny. *The complexity of robot motion planning*. MIT press, 1988.

[21] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In: *ECCV*. 2012.

[22] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3D: Learning from RGB-D Data in Indoor Environments. In: *3DV*. 2017.

[23] D. S. Chaplot, E. Parisotto, and R. Salakhutdinov. Active neural localization. In: *ICLR*. 2018.

[24] C. M. Christoudias, R. Urtasun, M. Salzmann, and T. Darrell. Learning to recognize objects from unseen modalities. In: *ECCV*. 2010.

[25] S. Daftry, J. A. Bagnell, and M. Hebert. Learning Transferable Policies for Monocular Reactive MAV Control. In: *ISER*. 2016.

[26] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In: *CVPR*. 2017.

[27] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In: *CVPR*. 2005.

[28] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra. Embodied Question Answering. In: *CVPR*. 2018.

[29] A. J. Davison and D. W. Murray. Mobile robot localisation using active vision. In: *ECCV*. 1998.

[30] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In: *SIGGRAPH*. ACM. 1996.

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

[32] J. Deng, A. Berg, S. Satheesh, H Su, A. Khosla, and L. Fei-Fei. *ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012)*. http://www.image-net.org/challenges/LSVRC/2012/.

[33] P. Dollár. *Piotr's Image and Video Matlab Toolbox (PMT)*. http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html.

[34] P. Dollár and C. L. Zitnick. Fast Edge Detection Using Structured Forests. In: *PAMI* (2015).

[35] P. Dollár and C. L. Zitnick. Structured Forests for Fast Edge Detection. In: *ICCV*. 2013.

[36] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In: *ICML*. 2014.

[37] L. Duan, D. Xu, and I. W. Tsang. Learning with Augmented Features for Heterogeneous Domain Adaptation. In: *ICML*. 2012.

[38] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning. In: *arXiv preprint arXiv:1611.02779* (2016).

[39] A. Elfes. Sonar-based real-world mapping and navigation. In: *RA* (1987).

[40] A. Elfes. Using occupancy grids for mobile robot perception and navigation. In: *Computer* (1989).

[41] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. In: *JMRL* (2008).

[42] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part Based Models. In: *TPAMI* (2010).

[43] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In: *NIPS*. 2016.

[44] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor MAV. In: *IROS*. 2012.

[45] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, T. Mikolov, et al. Devise: A deep visual-semantic embedding model. In: *NIPS*. 2013.

[46] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha. Visual simultaneous localization and mapping: a survey. In: *Artificial Intelligence Review* (2015).

[47] Y. Ganin and V. Lempitsky. Unsupervised Domain Adaptation by Backpropagation. In: *ICML*. 2015.

[48] D. Geman, Y. Amit, and K. Wilder. Joint Induction of Shape Features and Tree Classifiers. In: *TPAMI* (1997).

[49] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *CVPR*. 2014.

[50] R. Girshick. Fast R-CNN. In: *ICCV*. 2015.

[51] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. In: *RAL* (2016).

[52] G. Gkioxari and J. Malik. Finding Action Tubes. In: *CVPR*. 2015.

[53] B. Gong, Y. Shi, F. Sha, and K. Grauman. Geodesic Flow Kernel for Unsupervised Domain Adaptation. In: *CVPR*. 2012.

[54] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi. IQA: Visual Question Answering in Interactive Environments. In: *CVPR*. 2018.

[55] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous Deep Q-Learning with Model-based Acceleration. In: *ICML*. 2016.

[56] R. Guo and D. Hoiem. Scene understanding with complete scenes and structured representations. PhD thesis. UIUC, 2014.

[57] R. Guo and D. Hoiem. Support Surface Prediction in Indoor Scenes. In: *ICCV*. 2013.

[58] S. Gupta, P. Arbeláez, and J. Malik. Perceptual Organization and Recognition of Indoor Scenes from RGB-D Images. In: *CVPR*. 2013.

[59] S. Gupta, J. Hoffman, and J. Malik. Cross Modal Distillation for Supervision Transfer. In: *CVPR*. 2016.

[60] S. Gupta, P. Arbeláez, R. Girshick, and J. Malik. Aligning 3D Models to RGB-D Images of Cluttered Scenes. In: *CVPR*. 2015.

[61] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In: *CVPR*. 2017.

[62] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. *Cognitive mapping and planning for visual navigation : Supplementary Material*. https://arxiv.org/pdf/1702.03920v2.pdf. 2017.

[63] S. Gupta, P. Arbeláez, R. Girshick, and J. Malik. Indoor Scene Understanding with RGB-D Images: Bottom-up Segmentation, Object Detection and Semantic Segmentation. In: *IJCV* (2014).

[64] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning Rich Features from RGB-D Images for Object Detection and Segmentation. In: *ECCV*. 2014.

[65] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. *Learning Rich Features from RGB-D Images for Object Detection and Segmentation : Supplementary Material*. http://www.cs.berkeley.edu/~sgupta/pdf/rcnn-depth-supp.pdf. 2014.

[66] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel. Backprop KF: Learning Discriminative Deterministic State Estimators. In: *NIPS*. 2016.

[67] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun. Learning long-range vision for autonomous off-road driving. In: *Journal of Field Robotics* (2009).

[68] B. Hariharan. Beyond Bounding Boxes: Precise Localization of Objects in Images. PhD thesis. EECS Department, University of California, Berkeley, 2015.

[69] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for Object Segmentation and Fine-grained Localization. In: *CVPR*. 2015.

[70] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous Detection and Segmentation. In: *ECCV*. 2014.

[71] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In: *CVPR*. 2016.

[72] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In: *ECCV*. 2016.

[73] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In: *ICCV*. 2017.

[74] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In: *ECCV*. 2014.

[75] V. Hedau, D. Hoiem, and D. Forsyth. "Thinking inside the box: Using appearance models and context based on room geometry". In: *ECCV*. 2010.

[76] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver. Memory-based control with recurrent neural networks. In: *arXiv preprint arXiv:1512.04455* (2015).

[77] M. Hejrati and D. Ramanan. Analyzing 3D objects in cluttered images. In: *NIPS*. 2012.

[78] J. F. Henriques and A. Vedaldi. MapNet: An Allocentric Spatial Memory for Mapping Environments. In: *CVPR*. 2018.

[79] 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: *ISER*. 2010.

[80] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In: *NIPS 2014 Deep Learning Workshop*. 2014.

[81] S. Hochreiter and J. Schmidhuber. Long short-term memory. In: *Neural computation* (1997).

[82] J. Hoffman, E. Tzeng, J. Donahue, Y. Jia, K. Saenko, and T. Darrell. One-Shot Learning of Supervised Deep Convolutional Models. In: *arXiv 1312.6204; presented at ICLR Workshop*. 2014.

[83] J. Hoffman, S. Gupta, J. Leong, S. Guadarrama, and T. Darrell. Cross-Modal Adaptation for RGB-D Detection. In: *ICRA*. 2016.

[84] B. Horn, B. Klaus, and P. Horn. *Robot vision*. MIT press, 1986.

[85] B. K. Horn and B. G. Schunck. Determining optical flow. In: *Artificial intelligence* (1981).

[86] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: *ICML*. 2015.

[87] S. Izadi et al. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In: *UIST* (2011).

[88] D. H. Jacobson and D. Q. Mayne. Differential dynamic programming. In: (1970).

[89] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In: *NIPS*. 2015.

[90] 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*. 2013.

[91] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black. Towards understanding action recognition. In: *ICCV*. 2013.

[92] Y. Jia. *Caffe: An Open Source Convolutional Architecture for Fast Feature Embedding*. http://caffe.berkeleyvision.org/. 2013.

[93] G. Kahn, T. Zhang, S. Levine, and P. Abbeel. PLATO: Policy Learning using Adaptive Trajectory Optimization. In: *ICRA*. 2017.

[94] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In: *RA* (1996).

[95] A. Khan, C. Zhang, N. Atanasov, K. Karydis, V. Kumar, and D. D. Lee. Memory Augmented Control Networks. In: *ICLR*. 2018.

[96] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In: *IJRR* (1986).

[97] B. soo Kim, S. Xu, and S. Savarese. Accurate Localization of 3D Objects from RGB-D Data Using Segmentation Hypotheses. In: *CVPR*. 2013.

[98] H. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng. Autonomous helicopter flight via reinforcement learning. In: *NIPS*. 2003.

[99] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In: *arXiv preprint arXiv:1412.6980* (2014).

[100] J. J. Koenderink and A. J. van Doorn. Representation of local geometry in the visual system. In: *Biological cybernetics* (1987).

[101] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In: *ICRA*. 2004.

[102] C. Kong, D. Lin, M. Bansal, R. Urtasun, and S. Fidler. What are you talking about? Text-to-Image Coreference. In: *CVPR*. 2014.

[103] K. Konolige, J. Bowman, J. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua. View-based maps. In: *IJRR* (2010).

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

[105] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In: *NIPS*. 2012.

[106] B. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. In: *Robotics and autonomous systems* (1991).

[107] B. Kulis, K. Saenko, and T. Darrell. What You Saw is Not What You Get: Domain Adaptation Using Asymmetric Kernel Transforms. In: *CVPR*. 2011.

[108] K. Lai, L. Bo, and D. Fox. Unsupervised Feature Learning for 3D Scene Labeling. In: *ICRA*. 2014.

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

[110] K. Lai, L. Bo, X. Ren, and D. Fox. Detection-based Object Labeling in 3D Scenes. In: *ICRA*. 2012.

[111] S. M. Lavalle and J. J. Kuffner Jr. Rapidly-Exploring Random Trees: Progress and Prospects. In: *Algorithmic and Computational Robotics: New Directions*. 2000.

[112] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. In: *Neural Computation* (1989).

[113] K. Lenc and A. Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In: *CVPR*. 2015.

[114] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. In: *JMLR* (2016).

[115] W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In: *ICINCO*. 2004.

[116] J. J. Lim, A. Khosla, and A. Torralba. FPM: Fine pose Parts-based Model with 3D CAD models. In: *ECCV*. 2014.

[117] J. J. Lim, C. L. Zitnick, and P. Dollár. Sketch Tokens: A Learned Mid-level Representation for Contour and Object Detection. In: *CVPR*. 2013.

[118] D. Lin, S. Fidler, and R. Urtasun. Holistic scene understanding for 3D object detection with RGBD cameras. In: *ICCV*. 2013.

[119] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common Objects in Context. In: *ECCV*. 2014.

[120] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. In: *CVPR*. 2015.

[121] M. Long and J. Wang. Learning Transferable Features with Deep Adaptation Networks. In: *CoRR* abs/1502.02791 (2015). URL: http://arxiv.org/abs/1502.02791.

[122] D. Marr and T. Poggio. Cooperative computation of stereo disparity. In: *Science* (1976).

[123] D. Martin, C. Fowlkes, and J. Malik. Learning to Detect Natural Image Boundaries Using Local Brightness, Color and Texture Cues. In: *TPAMI* (2004).

[124] *Matterport*. https://matterport.com/.

[125] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. In: *ICLR*. 2017.

[126] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In: *ICML*. 2016.

[127] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. In: *Nature* (2015).

[128] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In: *ICML*. 2011.

[129] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In: *CVPR*. 2004.

[130] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of Memory, Active Perception, and Action in Minecraft. In: *ICML*. 2016.

[131] E. Parisotto and R. Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In: *ICLR*. 2018.

[132] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. In: *Neural networks* (2008).

[133] X. Ren, L. Bo, and D. Fox. RGB-(D) scene labeling: Features and algorithms. In: *CVPR*. 2012.

[134] L. G. Roberts. Machine perception of three-dimensional solids. PhD thesis. Massachusetts Institute of Technology, 1963.

[135] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In: *arXiv preprint arXiv:1412.6550* (2014).

[136] S. Ross, G. J. Gordon, and D. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In: *AISTATS*. 2011.

[137] H. Rowley, S. Baluja, and T. Kanade. *Human Face Detection in Visual Scenes*. Tech. rep. CMU-CS-95-158R. Carnegie Mellon University, 1995.

[138] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In: *International Conference on 3-D Digital Imaging and Modeling*. 2001.

[139] F. Sadeghi and S. Levine. (CAD)^{2}RL: Real Singel-Image Flight without a Singel Real Image. In: *RSS*. 2017.

[140] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison. SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. In: *CVPR*. 2013.

[141] S. Satkin, M. Rashid, J. Lin, and M. Hebert. 3DNN: 3D Nearest Neighbor. In: *IJCV* (2014).

[142] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-parametric topological memory for navigation. In: *ICLR*. 2018.

[143] A Savitsky and M. Golay. Smoothing and differentiation of data by simplified least squares procedures. In: *Analytical Chemistry* (1964).

[144] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun. MINOS: Multimodal Indoor Simulator for Navigation in Complex Environments. In: *arXiv:1712.03931* (2017).

[145] J. L. Schönberger and J.-M. Frahm. Structure-from-Motion Revisited. In: *CVPR*. 2016.

[146] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. In: *ICML*. 2015.

[147] A. G. Schwing, S. Fidler, M. Pollefeys, and R. Urtasun. Box in the box: Joint 3D layout and object reasoning from single images. In: *ICCV*. 2013.

[148] S. Seitz and R. Szeliski. Applications of computer vision to computer graphics. In: *Computer Graphics* (1999).

[149] S. M. Seitz and C. R. Dyer. View morphing. In: *SIGGRAPH*. ACM. 1996.

[150] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In: *ICLR*. 2014.

[151] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In: *CVPR*. 2013.

[152] T. Shao, W. Xu, K. Zhou, J. Wang, D. Li, and B. Guo. An interactive approach to semantic modeling of indoor scenes with an rgbd camera. In: *ACM TOG* (2012).

[153] J. Shotton, A. W. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In: *CVPR*. 2011.

[154] A. Shrivastava and A. Gupta. Building Part-Based Object Detectors via 3D Geometry. In: *ICCV*. 2013.

[155] N. Silberman, D. Sontag, and R. Fergus. Instance Segmentation of Indoor Scenes using a Coverage Loss. In: *ECCV*. 2014.

[156] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor Segmentation and Support Inference from RGBD Images. In: *ECCV*. 2012.

[157] K. Simonyan and A. Zisserman. Two-Stream Convolutional Networks for Action Recognition in Videos. In: *NIPS*. 2014.

[158] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In: *ICLR*. 2015.

[159] L. Smith and M. Gasser. The development of embodied cognition: Six lessons from babies. In: *Artificial life* (2005).

[160] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. In: *IJCV* (2008).

[161] R. Socher, M. Ganjoo, C. D. Manning, and A. Ng. Zero-shot learning through cross-modal transfer. In: *NIPS*. 2013.

[162] S. Song, S. P. Lichtenberg, and J. Xiao. SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite. In: *CVPR*. 2015.

[163] S. Song and J. Xiao. Sliding shapes for 3D object detection in depth images. In: *ECCV*. 2014.

[164] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A Dataset of 101 Human Action Classes From Videos in The Wild. In: *CRCV-TR-12-01*. 2012.

[165] N. Srivastava and R. Salakhutdinov. Multimodal Learning with Deep Boltzmann Machines. In: *JMRL* (2014).

[166] R. Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and environment maps. In: *SIGGRAPH*. ACM. 1997.

[167] A. Tamar, S. Levine, and P. Abbeel. Value Iteration Networks. In: *NIPS*. 2016.

[168] S. Tang, X. Wang, X. Lv, T. X. Han, J. Keller, Z. He, M. Skubic, and S. Lao. Histogram of Oriented Normal Vectors for Object Recognition with a Depth Sensor. In: *ACCV*. 2012.

[169] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[170] J. Tighe, M. Niethammer, and S. Lazebnik. Scene Parsing with Object Instances and Occlusion Ordering. In: *CVPR*. 2014.

[171] E. C. Tolman. Cognitive maps in rats and men. In: *Psychological review* 55.4 (1948), p. 189.

[172] M. Toussaint. Learning a world model and planning with a self-organizing, dynamic neural system. In: *NIPS*. 2003.

[173] S. Tulsiani and J. Malik. Viewpoints and Keypoints. In: *CVPR*. 2015.

[174] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko. Simultaneous Deep Transfer Across Domains and Tasks. In: *ICCV*. 2015.

[175] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In: *CVPR*. 2001.

[176] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Recurrent policy gradients. In: *Logic Journal of IGPL* (2010).

[177] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian. Building Generalizable Agents with a Realistic and Rich 3D Environment. In: *arXiv preprint arXiv: 1801.02209* (2018).

[178] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D Shapenets: A deep representation for volumetric shapes. In: *CVPR*. 2015.

[179] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. Gibson Env: Real-World Perception for Embodied Agents. In: *CVPR*. 2018.

[180] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In: *NIPS*. 2014.

[181] A. R. Zamir, T. Wekel, P. Agrawal, C. Wei, J. Malik, and S. Savarese. Generic 3D Representation via Pose Estimation and Matching. In: *ECCV*. 2016.

[182] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel. Learning deep neural network policies with continuous memory states. In: *ICRA*. 2016.

[183] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In: *ICRA*. 2017.

[184] M. Z. Zia, M. Stark, B. Schiele, and K. Schindler. Detailed 3D representations for object recognition and modeling. In: *TPAMI* (2013).