Robust and Scalable Perception For Autonomy

Peiyun Hu

CMU-RI-TR-21-67

August 27, 2021

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

Abstract

Autonomous mobile robots have the potential to drastically improve the quality of our daily life. For example, self-driving vehicles could make transportation safer and more affordable. To safely navigate complex environments, such robots need a perception system that translates raw sensory data to high-level understanding. This thesis focuses on two fundamental challenges toward building such perception systems via machine learning: robustness and scalability.

First, how can we learn a perception system that is robust to different types of variance in sensory data? For example, the sensory data of an object may look completely different depending on the distance and the presence of occlusion. Also, a perception system may encounter objects it has never seen during learning. To capture such variances, we develop approaches that make use of novel characterizations of context, visibility, and geometric prior.

Second, how can we rearchitect perception that requires less human supervision during learning? For example, standard perception software stacks build perceptual modules to recognize objects and forecast their movements. Training these modules requires object labels such as trajectories and semantic categories. To learn from large-scale unlabeled logs, we explore freespace supervision as an alternative to the predominant object supervision. We integrate freespace self-supervision with motion planners and demonstrate promising results.

Acknowledgments

First and foremost, I am very thankful to my advisor, Deva Ramanan, who has encouraged me to work on important research problems and provided countless insightful discussions throughout the development of my research. Our late-afternoon meetings would often go overtime and end with him saying "I am going to be yelled at". He is a wonderful teacher who always takes time to illustrate his thought process. I listen closely whenever he goes "Here is how I would think about it". He has been a wonderful mentor throughout my graduate studies, more than I could hope for, and for that, I am forever grateful.

I would also like to thank the members of my thesis committee, David Held, Christopher Atkeson, Drew Bagnell, and Raquel Urtasun, who have provided thoughtful comments and constructive feedback at various stages of my research. In addition, many thanks to David Held for giving me the opportunity to work with him over the last three years of my PhD, and collaborate closely on exciting research problems. Working with him has been a great pleasure.

I am also grateful for my previous research advisors, Jia Jia, my undergraduate research advisor at Tsinghua University, Jonathan Gratch, my summer internship advisor at University of Southern California, and Kaiming He, my internship advisor at Microsoft Research Asia. They have helped me develop the skills and taste for research, and supported me towards pursuing a PhD.

I am fortunate to have collaborated with amazing colleagues. First, I would like to thank Zachary Pezzementi and Herman Herman for the opportunity to spend a wonderful summer at National Robotics Engineering Center. I would also like to thank Zack Lipton and Anima Anandkumar for their advices during my summer internship at Amazon AI. In addition, I would like to thank Jason Ziglar for his mentorship during my summer internship at Argo AI.

My journey through graduate school would not have been as enjoyable without all the lovely friends. First, I want to thank folks from University of California, Irvine: Shu Kong, Minhaeng Lee, Bailey Kong, Phuc Nguyen, James Supančič, Sam Hallman, Songfan Yang, Yi Yang, Xiangxin Zhu, Golnaz Ghiasi, Raúl Díaz, Grégory Rogez, Chaitanya Desai. I miss the Sunday morning dim sum trips and the weekend hikes. I also want to shout out to folks at CMU: Aayush Bansal, Achal Dave, Martin Li, Ravi Mullapudi, Rohit Girdhar, Jason Zhang, Gengshan Yang, Tarasha Khurana, William Qi, Aaron Huang, Haochen Wang, Ishan Nigam, Siva Mynepalli, Krishna Uppala, and Sean Cha. I will miss our routine coffee breaks and closed door discussions (rants).

A very special thank you to my girlfriend, Yaying Wang, who has supported me throughout my graduate schools, endured a long-distance relationship, and filled my life with happiness and positivity. The time we spent together traveling and exploring the world makes some of my best memories. I am looking forward to what the future has in store for us.

Most importantly, I want to thank my parents, Keqin Tang and Zelin Hu, for their unconditional love and support. They recognized the impact of education for a kid growing up in a small town of China and were very committed to providing me the best education they could offer. For that, I am forever indebted.

Contents

Chapter 1

Introduction

Autonomous robots have the potential to drastically improve the quality of our daily life. For example, self-driving vehicles could make transportation of people and goods safer, more affordable, and available as a utility. To navigate complex and dynamic environments safely, such robots need a perception system that translates raw sensory data to high-level understanding.

We have witnessed exciting progress in the field of sensing and machine learning. On one hand, state-of-the-art sensors today are able to capture billions of pixels and millions of 3D points every second. On the other hand, deep learning has surpassed human performance on multiple computer vision benchmarks. However, there remain fundamental challenges in learning perception systems for autonomy. This thesis focuses on addressing two fundamental challenges in perception for autonomy: robustness and scalability.

First, how can we learn a perception system that is robust to different types of variance in sensory data? For example, the sensory data of an object may look completely different depending on the distance and the presence of occlusion. Also, a perception system may encounter objects it has never seen during learning. To model such variances, we develop approaches that make use of novel characterizations of context, visibility, and prior.

Second, how can we rearchitect perception such that the learning process is more efficient with human supervision? For example, standard perception software stacks build perceptual modules to recognize objects and forecast their movements. Training these modules requires object labels such as semantic categories and trajectories, which are laborious and expensive to annotate. As a result, the vast majority of data has not been used for training. To learn from large-scale unlabeled data logs, we explore freespace supervision – a new source of self-supervision.

1.1 Overview

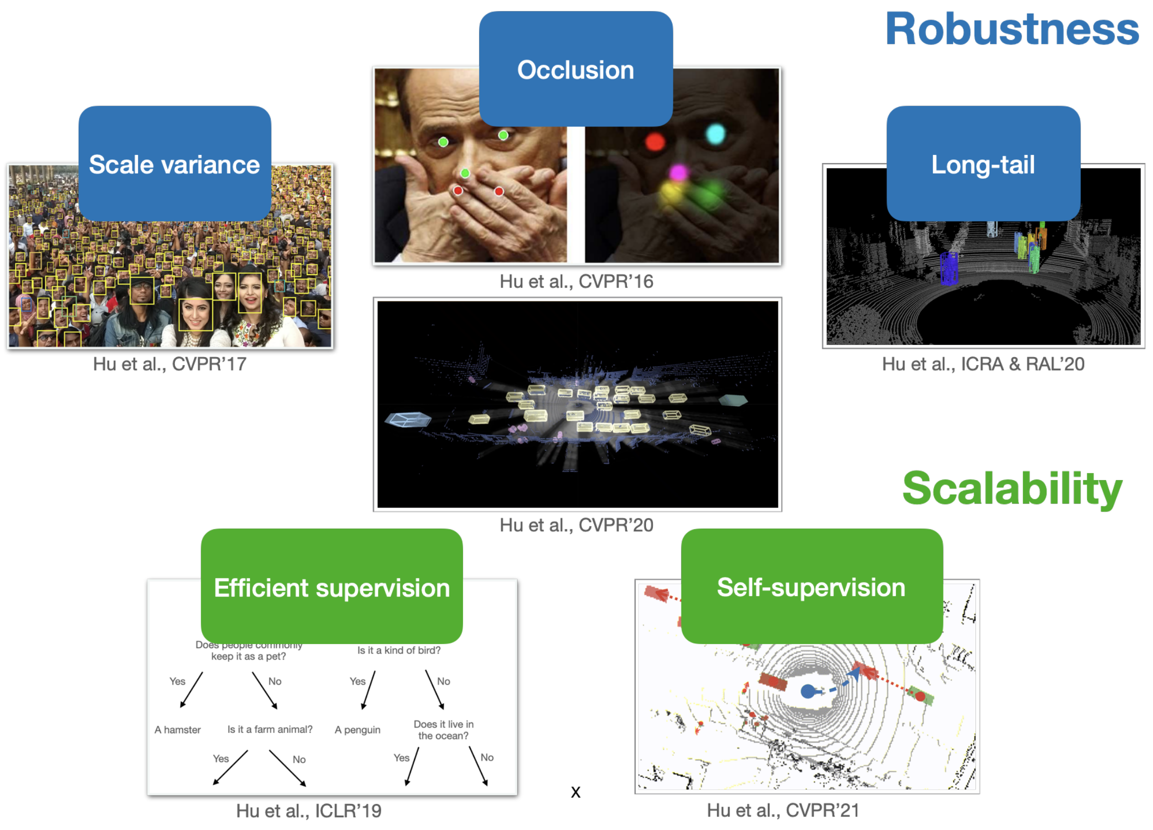


Figure 1.1

Part I: Robustness

Scale variance The sensory data of an object may look completely different depending on the object's distance to the sensor. This applies to both 2D and 3D sensors. How do we build visual systems that reliably recognize objects at different scales? Though tremendous strides have been made in object recognition, detecting small objects remains one of the open challenges. Traditional approaches aim to be scale-invariant, i.e. building one model that works for all scales. We argue that the cues for recognizing a 3px tall object must be fundamentally different than those for recognizing a 300px tall one. Our key insight is to build scale-variant representations, that is, different representations for different scales. Importantly, we argue that an effective representation must utilize contextual information beyond the extent of object, particularly for smaller objects, in order to make up for the lack of signal on the object itself. We apply this idea to face detection. On the massively benchmarked face detection dataset WIDERFACE, our results reduce error by a factor of 2.

Occlusion The presence of occlusion can drastically change the appearance of an object in sensory data. This applies to both 2D and 3D sensors. Neither cameras nor LiDARs can reliably observe what is behind objects. A multi-view system is often utilized to resolve occlusion issues. However, for sensors mounted on mobile robots, such line-of-sight occlusions are inevitable. Importantly, it may be extremely important to recognize what is occluded. For example, an autonomous vehicle needs to recognize if there could be a kid playing behind a parked car. We propose two approaches to help perception be robust to variance caused by occlusion.

First, top-down context could provide important cues to reason through occlusion. For example, a ball rolling into the middle of a street suggests the possibility of a kid chasing after it. The key question is: how do we architect perception models to enable such top-down reasoning? Classic approaches tend to architect "unidirectional" bottom-up convolutional nets. To enable top-down feedback, we propose a novel "bidirectional" convolutional net architecture and demonstrate its efficacy by improving the performance of challenging tasks such as localizing facial and human body keypoints under severe occlusion.

Second, 3D sensors know where they cannot see since they measure visibility directly. Most popular representations (such as PointNet [138]) are proposed in the context of processing truly 3D data (such as points sampled from mesh models), where there is little to no occlusion. Such approaches often destroy the valuable visibility information during preprocessing. We argue that perceptions systems built for autonomy need to exploit such visibility measurements. We develop a simple approach to augmenting state-of-the-art 3D object detectors with visibility. We show that doing so can significantly improve the 3D detection accuracy, especially on partially visible objects.

Long-tail Standard learning-based perception systems are known to struggle upon never-before-seen or rarely-seen objects. Unfortunately, there exists a long tail of such objects. Practical autonomous robots makes heavy use of perceptual priors such as background HD maps and adopt bottom-up grouping approaches. These approaches often requires less to no training and can pick up such novel objects. However, they tend to be brittle on subtle variance in everyday objects and requires post-hoc fixes in the downstream. In the meanwhile, data-driven learning has made an undeniable impact on learning 3D representations. We propose to combine data-driven learning and bottom-up grouping that exploits geometric priors. We demonstrate that the proposed segmentation framework combines the best of both worlds on segmenting objects on LIDAR point clouds.

Part II: Scalability

Efficient supervision Human feedback is expensive, especially in relative to the scale of which data is being collected. How do we curate more labels with less human feedback? Intuitively, only 1 bit worth of feedback is needed if the query is framed as a binary quality assurance (QA) task at the appropriate granularity; "is the output of the current perception module correct?" We develop an active binary learning regime, where a recognition model must choose the data and frame the query. We can draw a parallel to the famous game called 20 Questions. Here, the active learner asks the questions and human answers binary questions. We demonstrate that such an active learner can learn more accurate models at the same limited annotation cost and also fully label the dataset at a much lower cost.

Self-supervision The challenge in scalability goes beyond making the most bang for the buck out of every human interaction. Large self-driving fleets collect enormous amounts of raw sensory data every day, the majority of which will never reach annotation. How can we use unlabeled logs to improve autonomous navigation? Self-supervision! Standard autonomy software stacks for perception focus on recognizing objects and forecasting their movements. The learning of such object-centric perception comes at an enormous cost of human supervision. We rearchitect perception to enable self-supervised learning. Instead of forecasting objects, we propose to forecast how freespace evolves. Importantly, such freespace forecasting can be self-supervised via LIDAR raycasting. We demonstrate the new geometric perception-planning interface can significantly reduce collision rates in downstream local motion planning without requiring human feedback.

1.2 Thesis Outline

Chapter 2 In Chapter 2, we address the challenge of capturing scale variance when learning to perceive 2D image data. Context is key when finding small objects. In the context of building scale-invariant object detectors, we propose a scale-specific approach that allows contextual features to play a bigger role when detecting smaller objects.

Chapter 3 In Chapter 3, we address the challenge of top-down reasoning when learning to perceive 2D image data. Top-down reasoning is crucial when key low-level features are missing due to reasons such as severe occlusion. In the context of localizing facial keypoints, we derive a novel neural net architecture that allows such top-down feedback.

Chapter 4 In Chapter 4, we address the challenge of line-of-sight visibility constraints when learning to perceive data from 3D sensors. Knowing what we do not know could be useful. In the context of 3D object detection, we propose an approach that exploits such 3D visibility to improve detection accuracy on partially visible objects.

Chapter 5 In Chapter 5, we address the challenge of perceiving unknown objects from 3D data. Geometric prior is the key to identifying unknown objects. Classic grouping approaches, despite being brittle, can often outperform learning approaches on unseen objects. In the context of LiDAR segmentation, we propose an approach that combines the merits of grouping and learning to improve generalization.

Chapter 6 In Chapter 6, we address the challenge of learning to perceive with limited human supervision. Labeling can be greatly simplified with deductive reasoning. In the context of multi-class classification, we formulate active learning with partial feedback as a game of 20 questions between human experts and evolving models.

Chapter 7 In Chapter 7, we address the challenge of scalability in learning perception systems. Data is being labeled at a much slower pace than it is created. In the context of developing perception systems for self-driving vehicles, we propose to forecast freespace, which can be directly used to assist local planning while requires no human supervision.

Chapter 8 In Chapter 8, we address an important criticism over future freespace as a representation: view-specific. We propose a novel approach that allows us to forecast view-independent occupancy with view-specific freespace supervision.

Part I

Robustness

Chapter 2

Exploiting Context for Scale Variance

2.1 Introduction

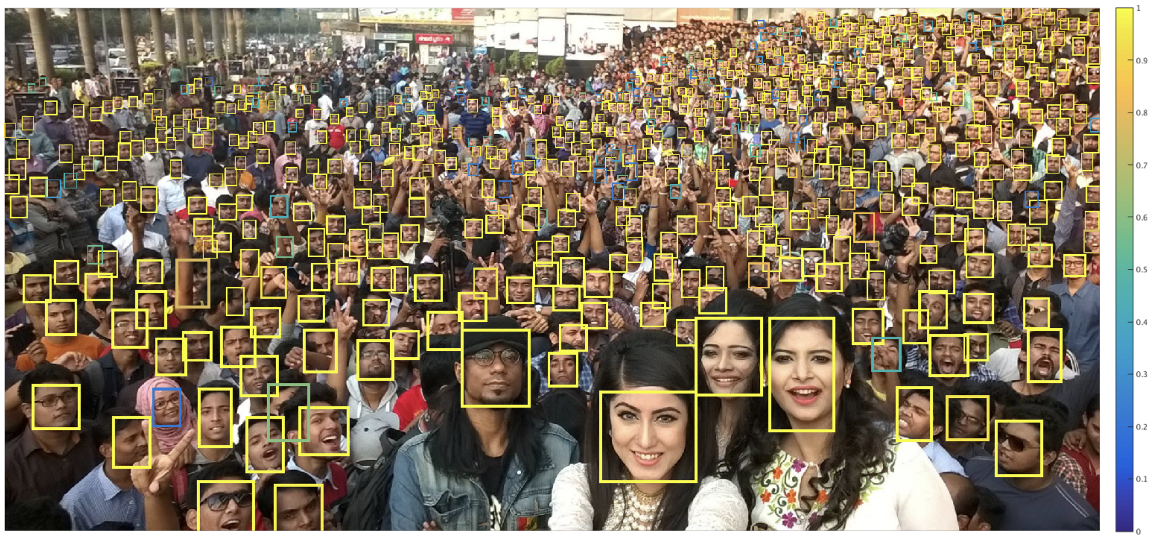


Figure 2.1: We describe a detector that can find 685 faces out of the reportedly 1000 present, by making use of novel characterizations of scale, resolution, and context to find small objects. Detector confidence is given by the colorbar on the right: can you confidently identify errors?

Though tremendous strides have been made in object recognition, one of the remaining open challenges is detecting small objects. We explore three aspects of the problem in

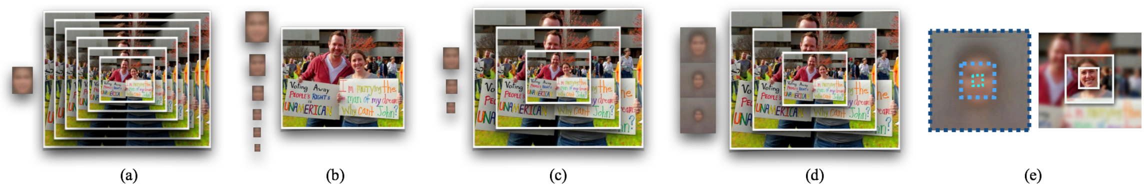


Figure 2.2: Different approaches for capturing scale-invariance. Traditional approaches build a single-scale template that is applied on a finely-discretized image pyramid (a). To exploit different cues available at different resolutions, one could build different detectors for different object scales (b). Such an approach may fail on extreme object scales that are rarely observed in training (or pre-training) data. We make use of a coarse image pyramid to capture extreme scale challenges in (c). Finally, to improve performance on small faces, we model additional context, which is efficiently implemented as a fixed-size receptive field across all scale-specific templates (d). We define templates over features extracted from multiple layers of a deep model, which is analogous to foveal descriptors (e).

the context of face detection: the role of scale invariance, image resolution and contextual reasoning. Scale-invariance is a fundamental property of almost all current recognition and object detection systems. But from a practical perspective, scale-invariance cannot hold for sensors with finite resolution: the cues for recognizing a 300px tall face are qualitatively different that those for recognizing a 3px tall face.

Multi-task modeling of scales: Much recent work in object detection makes use of scale-normalized classifiers (e.g., scanning-window detectors run on a an image pyramid [43] or region-classifiers run on "ROI"-pooled image features [52, 143]). When resizing regions to a canonical template size, we ask a simple question –*what should the size of the template be?* On one hand, we want a small template that can detect small faces; on the other hand, we want a large template that can exploit detailed features (of say, facial parts) to increase accuracy. Instead of a "one-size-fits-all" approach, we train separate detectors tuned for different scales (and aspect ratios). Training a large collection of scale-specific detectors may suffer from lack of training data for individual scales and inefficiency from running a large number of detectors at test time. To address both concerns, we train and run scale-specific detectors in a *multi-task* fashion : they make use of features defined over multiple layers of single (deep) feature hierarchy. While such a strategy results in detectors of high accuracy for large objects, finding small things is still challenging.

How to generalize pre-trained networks? We provide two remaining key insights to the problem of finding small objects. The first is an analysis of how best to extract scale-invariant features from pre-trained deep networks. We demonstrate that existing networks are tuned for objects of a characteristic size (encountered in pre-training datasets such as

ImageNet). To extend features fine-tuned from these networks to objects of novel sizes, we employ a simply strategy: resize images at test-time by interpolation and decimation. While many recognition systems are applied in a "multi-resolution" fashion by processing an image pyramid, we find that interpolating the lowest layer of the pyramid is particularly crucial for finding small objects [43]. Hence our final approach (Fig. 2.2) is a delicate mixture of scale-specific detectors that are used in a scale-invariant fashion (by processing an image pyramid to capture large scale variations).

How best to encode context? Finding small objects is fundamentally challenging because there is little signal on the object to exploit. Hence we argue that one must use image evidence beyond the object extent. This is often formulated as "context". In Fig. 2.3, we present a simple human experiment where users attempt to classify true and false positive faces (as given by our detector). It is dramatically clear that humans need context to accurately classify small faces. Though this observation is quite intuitive and highly explored in computer vision [129, 179], it has been notoriously hard to quantifiably demonstrate the benefit of context in recognition [34, 46, 189]. One of the challenges appears to be how to effectively encode large image regions. We demonstrate that convolutional deep features extracted from multiple layers (also known as "hypercolumn" features [61, 109]) are effective "foveal" descriptors that capture both high-resolution detail and coarse low-resolution cues across large receptive field (Fig. 2.2 (e)). We show that high-resolution components of our foveal descriptors (extracted from lower convolutional layers) are crucial for such accurate localization in Fig. 2.5.

Our contribution: We provide an in-depth analysis of image resolution, object scale, and spatial context for the purposes of finding small faces. We demonstrate state-of-the-art results on massively-benchmarked face datasets (FDDB and WIDER FACE). In particular, when compared to prior art on WIDER FACE, our results **reduce error by a factor of 2** (our models produce an AP of 81% while prior art ranges from 29-64%).

2.2 Related work

Scale-invariance: The vast majority of recognition pipelines focus on scale-invariant representations, dating back to SIFT[112]. Current approaches to detection such as Faster RCNN [143] subscribe to this philosophy as well, extracting scale-invariant features through ROI pooling or an image pyramid [144]. We provide an in-depth exploration of scale-variant templates, which have been previously proposed for pedestrian detection[133], sometimes in the context of improved speed [10]. SSD [107] is a recent technique based on deep features that makes use of scale-variant templates. Our work differs in our exploration of context

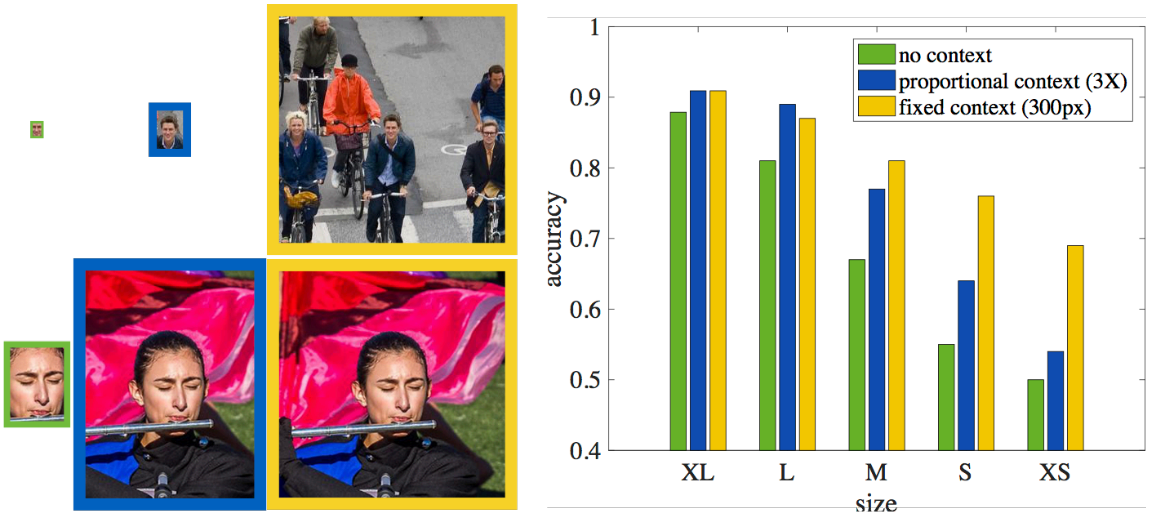


Figure 2.3: On the **left**, we visualize a large and small face, both with and without context. Context is not needed for a human user to recognize the large face, while the small face is dramatically unrecognizable without its context. We quantify this observation with a simple human experiment on the **right**, where users classify true and false positive faces of our proposed detector. Adding fixed context (300px) reduces error by 20% on small size (S) comparing to no context, but only 2% for extra large (XL). Also, proportional context (3X) becomes less helpful as size goes smaller, suggesting context should be modeled in a scale-variant way as well. We operationalize this observation with foveal templates of massively-large receptive fields (around 300x300, the size of contextual images shown as yellow boxes).

for tiny object detection.

Context: Context is key to finding small instances as shown in multiple recognition tasks. In object detection, [9] stacks spatial RNNs (IRNN[101]) model context outside the region of interest and shows improvements on small object detection. In pedestrian detection, [133] uses ground plane estimation as contextual features and improves detection on small instances. In face detection, [210] simultaneously pool ROI features around faces and bodies for scoring detections, which significantly improve overall performance. Our proposed work makes use of large local context (as opposed to a global contextual descriptor [9, 133]) in a scale-variant way (as opposed to [210]). We show that context is mostly useful for finding low-resolution faces.

Multi-scale representation: Multi-scale representation has been proven useful for many recognition tasks. [6, 61, 109] show that deep multi-scale descriptors (known as "hypercolumns") are useful for semantic segmentation. [9, 107] demonstrate improvements for such models on object detection. [210] pools multi-scale ROI features. Our model uses "hypercolumn" features, pointing out that fine-scale features are most useful for localizing small objects (Sec. 2.3.1 and Fig. 2.5).

RPN: Our model superficially resembles a region-proposal network (RPN) trained for a specific object class instead of a general "objectness" proposal generator [143]. The important differences are that we use foveal descriptors (implemented through multi-scale features), we select a range of object sizes and aspects through cross-validation, and our models make use of an image pyramid to find extreme scales. In particular, our approach for finding small objects make use of scale-specific detectors tuned for interpolated images. Without these modifications, performance on small-faces dramatically drops by more than 10% (Table 2.1).

2.3 Exploring context and resolution

In this section, we present an exploratory analysis of the issues at play that will inform our final model. To frame the discussion, we ask the following simple question: *what is the best way to find small faces of a fixed-size (25x20)?*. By explicitly factoring out scale-variation in terms of the desired output, we can explore the role of context and the canonical template size. Intuitively, context will be crucial for finding small faces. Canonical template size may seem like a strange dimension to explore - given that we want to find faces of size 25x20, why define a template of any size other than 25x20? Our analysis gives a surprising answer of when and why this should be done. To better understand the implications of our analysis, along the way we also ask the analogous question for a large object size: *what is the best way to find large faces of a fixed-size (250x200)?*.

Setup: We explore different strategies for building scanning-window detectors for fixed-size (e.g., 25x20) faces. We treat fixed-size object detection as a *binary heatmap prediction problem*, where the predicted heatmap at a pixel position (x, y) specifies the confidence of a fixed-size detection centered at (x, y). We train heatmap predictors using a fully convolutional network (FCN) [109] defined over a state-of-the-art architecture ResNet [64]. We explore multi-scale features extracted from the last layer of each res-block, i.e. (res2cx, res3dx, res4fx, res5cx) in terms of ResNet-50. We will henceforth refer to these as (res2, res3, res4, res5) features. We discuss the remaining particulars of our training pipeline in Section 7.4.

2.3.1 Context

Fig. 2.4 presents an analysis of the effect of context, as given by the size of the receptive field (RF) used to make heatmap prediction. Recall that for fixed-size detection window,

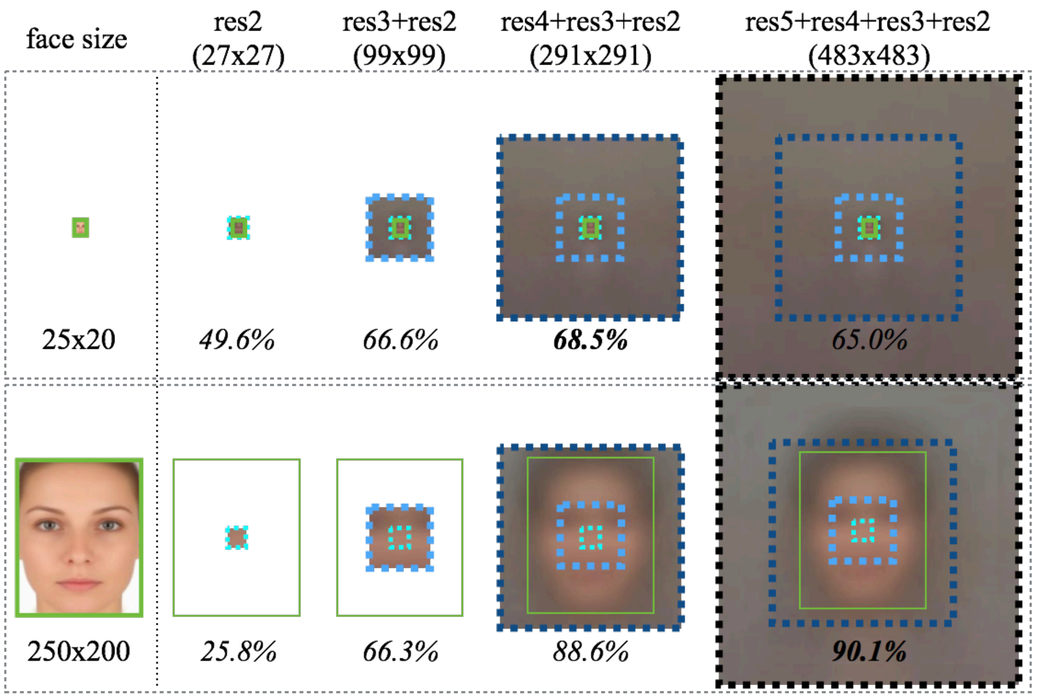


Figure 2.4: Modeling additional context helps, especially for finding small faces. The improvement from adding context to a tight-fitting template is greater for small faces (18.9%) than for large faces (1.5%). Interestingly smaller receptive fields do better for small faces, because the entire face is visible. The green box represents the actual face size, while dotted boxes represent receptive fields associated with features from different layers (cyan = res2, light-blue = res3, dark-blue = res4, black = res5). Same colors are used in Figures 2.5 and 2.7.

we can choose to make predictions using features with arbitrarily smaller or larger receptive fields compared to this window. Because convolutional features at higher layers tend to have larger receptive fields (e.g., res4 features span 291x291 pixels), smaller receptive fields necessitate the use of lower layer features. We see a number of general trends. Adding context almost always helps, though eventually additional context for tiny faces (beyond 300x300 pixels) hurts. We verified that this was due to over-fitting (by examining training and test performance). Interestingly, smaller receptive fields do better for small faces, because the entire face is visible - it is hard to find large faces if one looks for only the tip of the nose. More importantly, we analyze the impact of context by comparing performance of a "tight" RF (restricted to the object extent) to the best-scoring "loose" RF with additional context. Accuracy for small faces improves by 18.9%, while accuracy for large faces improves by 1.5%, consistent with our human experiments (that suggest that context is most useful for small instances). Our results suggest that we can build multi-task templates for detectors

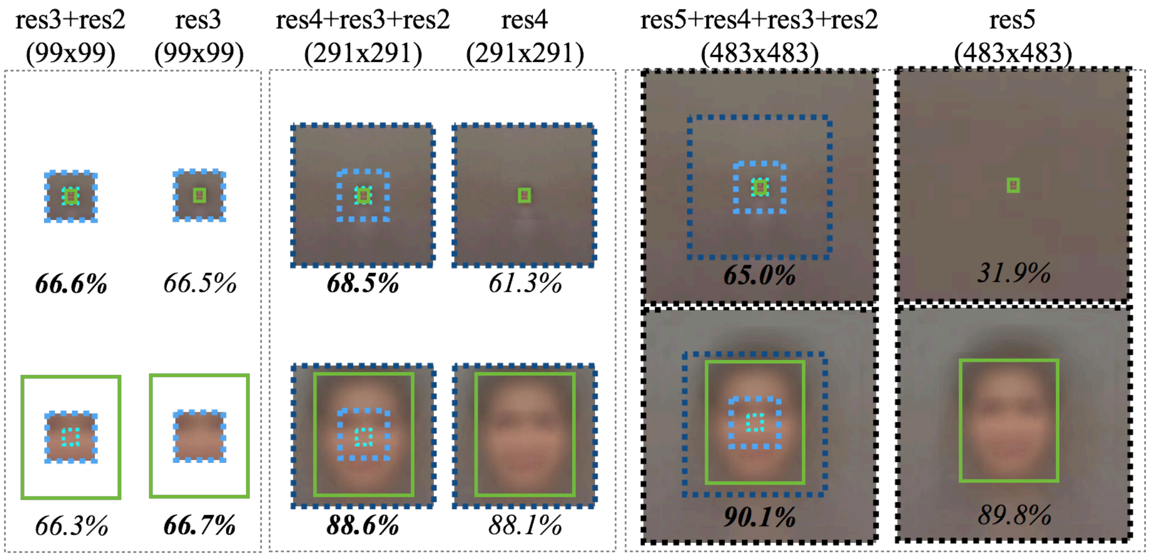


Figure 2.5: Foveal descriptor is crucial for accurate detection on small objects. The small template (**top**) performs 7% worse with only res4 and 33% worse with only res5. On the contrary, removing foveal structure does not hurt the large template (**bottom**), suggesting high-resolution from lower layers is mostly useful for finding small objects!

of different sizes with identical receptive fields (of size 291x291), which is particularly simple to implement as a *multi-channel* heatmap prediction problem (where each scale-specific channel and pixel position has its own binary loss). In Fig. 2.5, we compare between descriptors with and without foveal structure, which shows that high-resolution components of our foveal descriptors are crucial for accurate detection on small instances.

2.3.2 Resolution

We now explore a rather strange question. What if we train a template whose size intentionally differs from the target object to be detected? In theory, one can use a "medium"-size template (50x40) to find small faces (25x20) on a 2X upsampled (interpolated) test image. Fig. 2.7 actually shows the surprising result that this noticeably boosts performance, from 69% to 75%! We ask the reverse question for large faces: can one find large faces (250x200) by running a template tuned for "medium" faces (125x100) on test images downsampled by 2X? Once again, we see a noticeable increase in performance, from 89% to 94%!

One explanation is that we have different amounts of training data for different object sizes, and we expect better performance for those sizes with more training data. A recurring observation in "in-the-wild" datasets such as WIDER FACE and COCO [106] is that smaller

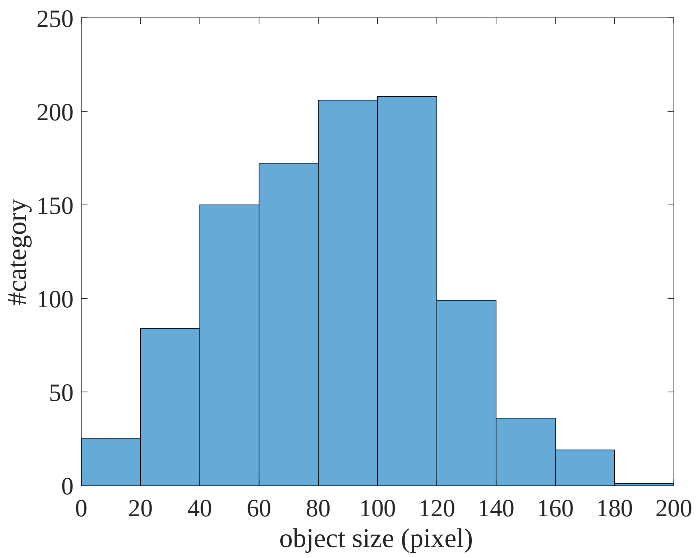


Figure 2.6: The distribution of average object scales in the ImageNet dataset (assuming images are normalized to 224x224). more than 80% categories have an average object size between 40 and 140 pixel. We hypothesize that pre-trained ImageNet models are optimized for objects in that range.

objects greatly outnumber larger objects, in part because more small things can be labeled in a fixed-size image. We verify this for WIDER FACE in Fig. 2.8 (gray curve). While imbalanced data may explain why detecting large faces is easier with medium templates (because there are more medium-sized faces for training), it does not explain the result for small faces. There exists *less* training examples of medium faces, yet performance is still much better using a medium-size template.

We find that the culprit lies in the distribution of object scales in the *pre-trained* dataset (ImageNet). Fig. 2.6 reveals that 80% of the training examples in ImageNet contain objects of a "medium" size, between 40 to 140px. Specifically, we hypothesize that the pre-trained ImageNet model (used for fine-tuning our scale-specific detectors) is optimized for objects in that range, and that one should bias canonical-size template sizes to lie in that range when possible. We verify this hypothesis in the next section, where we describe a pipeline for building scale-specific detectors with varying canonical resolutions.

2.4 Approach: scale-specific detection

It is natural to ask a follow-up question: is there a general strategy for selecting template resolutions for particular object sizes? We demonstrate that one can make use of multi-task

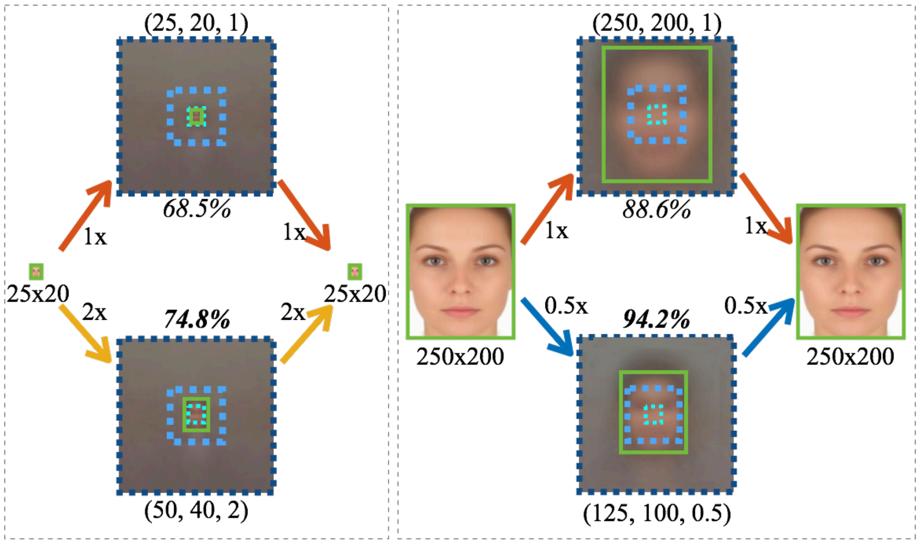


Figure 2.7: Building templates at original resolution is not optimal. For finding small (25x20) faces, building templates at 2x resolution improves overall accuracy by 6.3%; while for finding large (250x200) faces, building templates at 0.5x resolution improves overall accuracy by 5.6%.

learning to "brute-force" train several templates at different resolution, and greedily select the ones that do the best. As it turns out, there appears to be a general strategy consistent with our analysis in the previous section.

First, let us define some notation. We use $t(h, w, \sigma)$ to represent a template. Such a template is tuned to detect objects of size $(h/\sigma, w/\sigma)$ at resolution σ. For example, the right-hand-side Fig 2.7 uses both $t(250, 200, 1)$ (top) and $t(125, 100, 0.5)$ (bottom) to find 250x200 faces.

Given a training dataset of images and bounding boxes, we can define a set of canonical bounding box shapes that roughly covers the bounding box shape space. In this paper, we define such canonical shapes by clustering, which is derived based on Jaccard distance d(Eq. (2.1)):

$$d(s_i, s_j) = 1 - \mathrm{J}(s_i, s_j) \tag{2.1}$$

where, $s_i = (h_i, w_i)$ and $s_j = (h_j, w_j)$ are a pair of bounding box shapes and J represents the standard Jaccard similarity (intersection over union overlap).

Now for each target object size $s_i = (h_i, w_i)$, we ask: *what σ_i will maximize performance of $t_i(\sigma_i h_i, \sigma_i w_i, \sigma_i)$?* To answer, we simply train separate multi-task models for each value of $\sigma \in \Sigma$ (some fixed set) and take the max for each object size. We plot the performance of each resolution-specific multi-task model as a colored curve in Fig. 2.8. With optimal σ_i

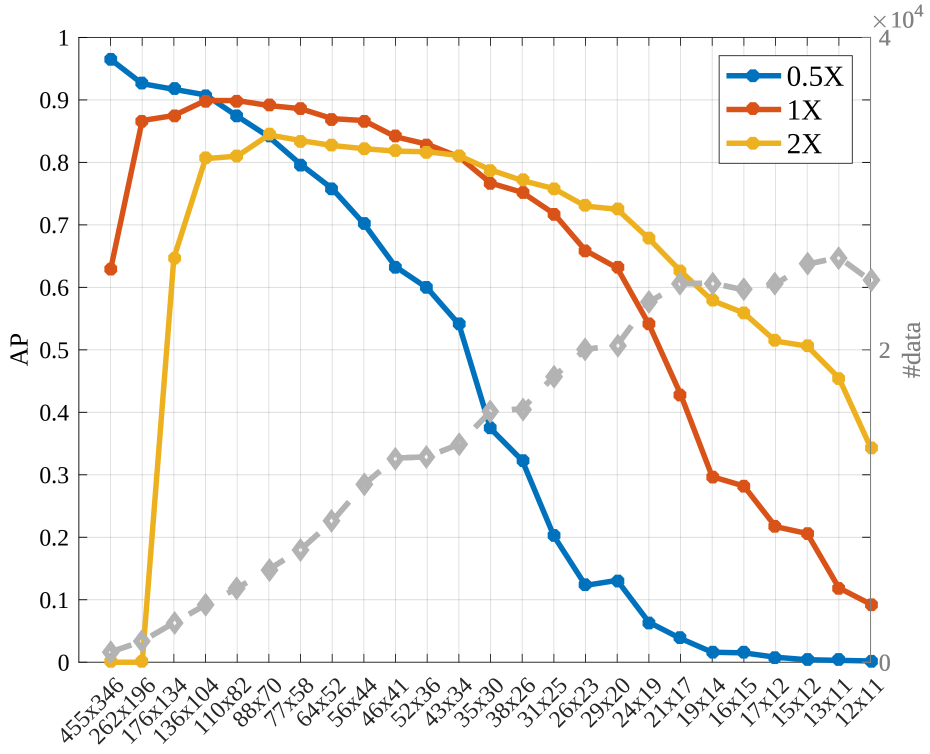


Figure 2.8: Template resolution analysis. X-axis represents target object sizes, derived by clustering. Left Y-axis shows AP around each target size (ignoring objects with more than 0.5 Jaccard distance). Natural regimes emerge in the figure: for finding large faces (more than 140px in height), build templates at 0.5 resolution; for finding smaller faces (less than 40px in height), build templates at 2X resolution. For sizes in between, build templates at 1X resolution. Right Y-axis along with the gray curve shows the number of data within 0.5 Jaccard distance for each object size, suggesting that more small faces are annotated.

for each (h_i, w_i), we retrain one multi-task model with "hybrid" resolutions (referred to as HR), which in practice follows the upper envelope of all the curves. Interestingly, there exist natural regimes for different strategies: to find large objects (greater than 140px in height), use 2X smaller canonical resolution. To find small objects (less than 40px in height), use 2X larger canonical template resolution. Otherwise, use the same (1X) resolution. Our results closely follow the statistics of ImageNet (Fig. 2.6), for which most objects fall into this range.

Pruning: The hybrid-resolution multitask model in the previous section is somewhat redundant. For example, template $(62, 50, 2)$, the optimal template for finding 31x25 faces, is redundant given the existence of template $(64, 50, 1)$, the optimal template for finding 64x50 faces. Can we prune away such redundancies? Yes! We refer the reader to the caption

Method	Easy	Medium	Hard
RPN	0.896	0.847	0.716
HR-ResNet101 (Full)	0.919	0.908	0.823
HR-ResNet101 (A+B)	**0.925**	**0.914**	**0.831**

Table 2.1: Pruning away redundant templates does not hurt performance. As a reference, we also plotted the performance of a vanilla RPN as mentioned in Sec. 6.4. Please refer to Fig. 2.10 for visualization of (Full) and (A+B).

in Fig. 2.10 for an intuitive description. As Table 2.1 shows, pruning away redundant templates led to some small improvement. Essentially, our model can be reduced to a small set of scale-specific templates (tuned for 40-140px tall faces) that can be run on a coarse image pyramid (including 2X interpolation), combined with a set of scale-specific templates designed for finding small faces (less than 20px in height) in 2X interpolated images.

2.4.1 Architecture

We visualize our proposed architecture in Fig. 3.4. We train binary multi-channel heatmap predictors to report object confidences for a range of face sizes (40-140px in height). We then find larger and smaller faces with a coarse image pyramid, which importantly includes a 2X upsampling stage with special-purpose heatmaps that are predicted only for this resolution (e.g., designed for tiny faces of shorter than 20 pixels). For the shared CNNs, we experimented with three different architectures: ResNet101, ResNet50, and VGG16. Though ResNet101 performs the best, we include a detailed comparison for all models in Table 2.2. Importantly, our results noticeably improve over prior art for *all* models on "hard" set.

Details: Given training images with ground-truth annotations of objects and templates, we define positive locations to be those where IOU overlap exceeds 70%, and negative locations to be those where the overlap is below 30% (all other locations are ignored by zero-ing out the gradient). Note that this implies that each large object instance generates many more positive training examples than small instances. Since this results in a highly imbalanced binary classification training set, we make use of balanced sampling [52] and hard-example mining [161] to ameliorate such effects. We find performance increased with a post-processing linear regressor that fine-tuned reported bounding-box locations. To ensure that we train on data similar to test conditions, we randomly resize training data to the range of Σ resolution that we consider at test-time (0.5x,1x,2x) and learn from a random fixed-size crop of 500x500 pixels per image (to take advantage of batch processing). We fine-tune pre-trained ImageNet models on the WIDER FACE training set with a fixed

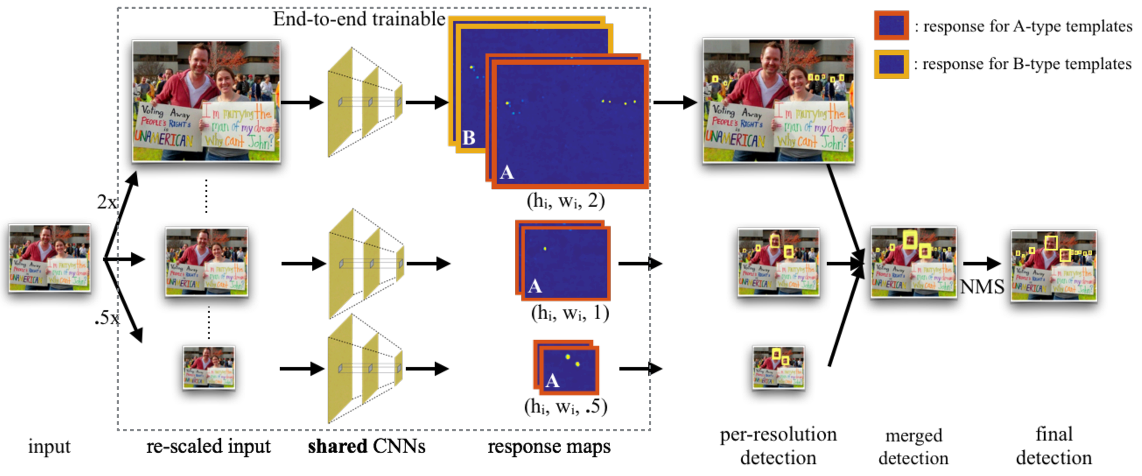


Figure 2.9: Overview of our detection pipeline. Starting with an input image, we first create a coarse-level image pyramid (including 2X interpolation). At each scale, we feed the re-scaled input into a CNN to extract hyper-column features. Based on hyper-column features, we predict response maps (for both detection and regression) of corresponding templates. Given response maps, we first extract detection bounding boxes for each scale and then merge them back in original scale. In the end, we apply non-maximum suppression (NMS) to get the final detection results. The dotted box represents the part that is trained end-to-end. We run A-type templates (tuned for 40-140px tall faces) on the coarse image pyramid (including 2X interpolation), while only run B-type (tuned for less than 20px tall faces) templates on only 2X interpolated images. Please refer to Fig. 2.10 for more details about two types of templates.

Method	Easy	Medium	Hard
ACF[194]	0.659	0.541	0.273
Two-stage CNN[198]	0.681	0.618	0.323
Multiscale Cascade CNN[197]	0.691	0.634	0.345
Faceness[197]	0.713	0.664	0.424
Multitask Cascade CNN[205]	0.848	0.825	0.598
CMS-RCNN[210]	0.899	0.874	0.624
HR-VGG16	0.862	0.844	0.749
HR-ResNet50	0.907	0.890	0.802
HR-ResNet101	**0.919**	**0.908**	**0.823**

Table 2.2: Performance of our approach on validation set featuring different architectures. ResNet101 performs slightly better than ResNet50 and much better than VGG16. Importantly, our VGG16-based model already outperforms prior art by a large margin on "hard" set.

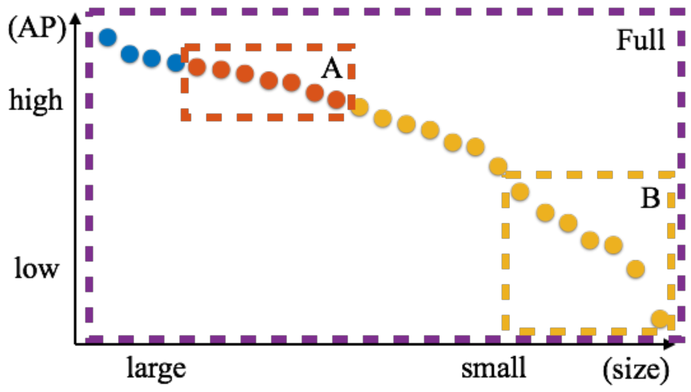


Figure 2.10: Pruning away redundant templates. Suppose we test templates built at 1X resolution (A) on a coarse image pyramid (including 2X interpolation). They will cover a larger range of scale except extremely small sizes, which are best detected using templates built at 2X, as shown in Fig. 2.8. Therefore, our final model is reduced to two small sets of scale-specific templates: (A) tuned for 40-140px tall faces and are run on a coarse image pyramid (including 2X interpolation) and (B) tuned for faces shorter than 20px and are only run in 2X interpolated images.

learning rate of 10^{-4}, and evaluate performance on the WIDER FACE validation set (for diagnostics) and held-out testset. To generate a final set of detections, we apply standard NMS to the detected heatmap with an overlap threshold of 30%. We discuss other details of our procedure in supplementary material. Our code will be released in the future.

2.5 Experiments

WIDER FACE: We train a hybrid-resolution model with 25 templates on WIDER FACE's training set and reported our performance on the held-out test set. As we see in Fig. 2.11, our hybrid-resolution model (HR) achieves state-of-the-art performance on all difficulty levels, but most importantly, it outperformed prior-art by 17% on the "hard" set. Note that "hard" set includes all faces taller than 10px, hence more accurately represents performance on the full testset. Qualitative results are shown in Fig. 2.13. Please see supplementary material for evaluation on "easy" and "medium", and more diagnosis[74] of our detectors. We only report the performance of full model (without pruning) on test set.

FDDB: We test our hybrid-resolution model trained on WIDER FACE directly on FDDB. Our out-of-the-box detector (HR) outperforms all published results on discrete score. Because WIDER FACE has bounding box annotation while FDDB has bounding ellipses, we train a post-hoc elliptical regressor to transform our predicted bounding boxes to bounding ellipses. With the post-hoc regressor, our detector achieves state-of-the-art

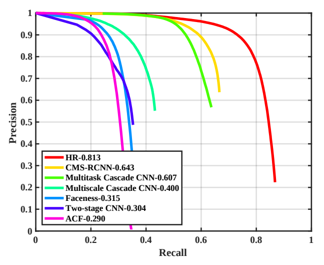


Figure 2.11: Precision recall curves on WIDER FACE test set, featuring "hard" set, where our approach (HR) outperforms state-of-the-art by 17%.

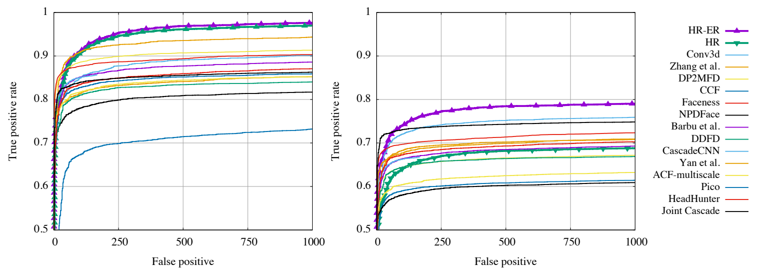


Figure 2.12: ROC curves on FDDB test set. Our out-of-the-box detector (HR) achieves state-of-the-art on discrete score(on the **left**). By learning a post-hoc elliptical regressor, our approach (HR-ER) achieves state-of-the-art on continuous score as well(on the **right**). Note that only published results are included here.

performance on continuous score as well. Our post-hoc regressor is trained following 10-fold cross validation. In Fig. 2.12, we plot the performance of our detector both with and without the elliptical regressor (ER). Qualitative results are shown in Fig. 2.14. Please see supplementary material for details on how we train the elliptical regressor.

Conclusion: We propose a simple yet effective framework for finding small objects, demonstrating that both large context and scale-variant representations are crucial. We specifically show that massively-large receptive fields can be effectively encoded as a foveal descriptor that captures both coarse context (necessary for detecting small objects) and high-resolution image features (helpful for localizing small objects). We also explore the encoding of scale in existing pre-trained deep networks, suggesting a simple way to extrapolate networks tuned for limited scales to more extreme scenarios in a scale-variant fashion. Finally, we use our detailed analysis of scale, resolution, and context to develop a state-of-the-art face detector that significantly outperforms prior work on standard benchmarks.

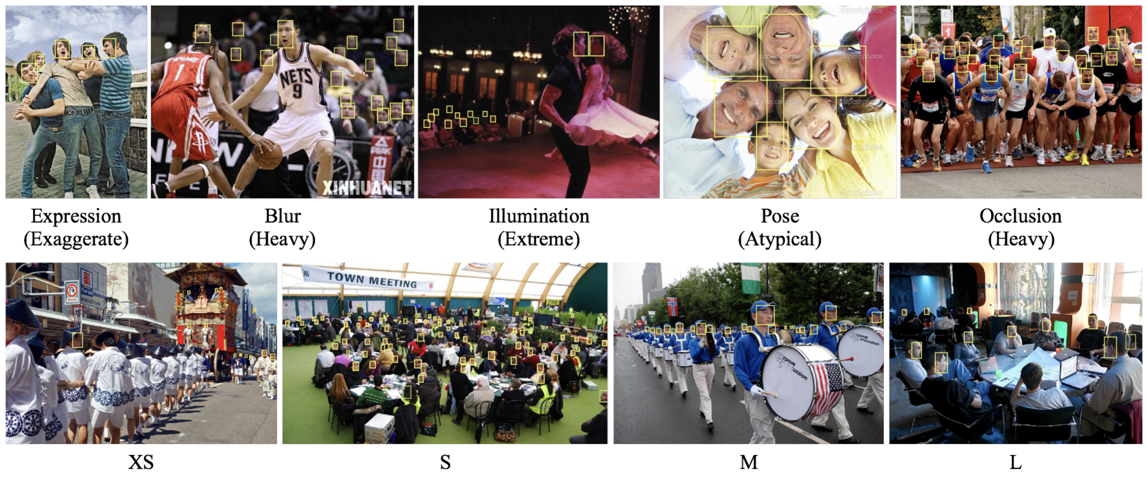


Figure 2.13: Qualitative results on WIDER Face. We visualize one example for each attribute and scale. Our proposed detector is able to detect faces at a continuous range of scales, while being robust to challenges such as expression, blur, illumination etc. Please zoom in to look for some very small detections.

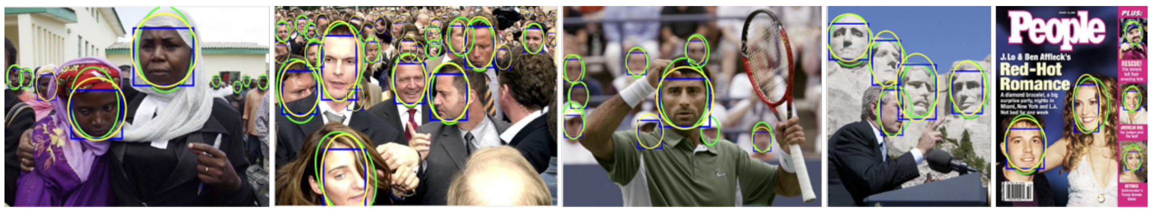


Figure 2.14: Qualitative results on FDDB. Our proposed detector is robust to heavy occlusion, heavy blur, large appearance and scale variance. Interestingly, many faces under such challenges are not even annotated (second example). Green ellipses are ground truth, blue bounding boxes are detection results, and yellow ellipses are regressed ellipses.

Chapter 3

Exploiting Top-down Feedback for Occlusion

3.1 Introduction

Hierarchical models of visual processing date back to the iconic work of Marr [116]. Convolutional neural nets (CNN's), pioneered by LeCun *et al.* [102], are hierarchical models that compute progressively more invariant representations of an image in a bottom-up, feedforward fashion. They have demonstrated remarkable progress in recent history for visual tasks such as classification [95, 163, 172], object detection [52], and image captioning [85], among others.

Feedback in biology: Biological evidence suggests that *vision at a glance* tasks, such as rapid scene categorization [184], can be effectively computed with feedforward hierarchical processing. However, *vision with scrutiny* tasks, such as fine-grained categorization [93] or detailed spatial manipulations [78], appear to require feedback along a "reverse hierarchy" [72]. Indeed, most neural connections in the visual cortex are believed to be feedback rather than feedforward [37, 96].

Feedback in computer vision: Feedback has also played a central role in many classic computer vision models. Hierarchical probabilistic models [79, 104, 211], allow random variables in one layer to be naturally influenced by those above and below. For example, lower layer variables may encode edges, middle layer variables may encode parts, while higher layers encode objects. Part models [43] allow a face object to influence the activation of an eye part through top-down feedback, which is particularly vital for occluded parts that receive misleading bottom-up signals. Interestingly, feed-forward inference on

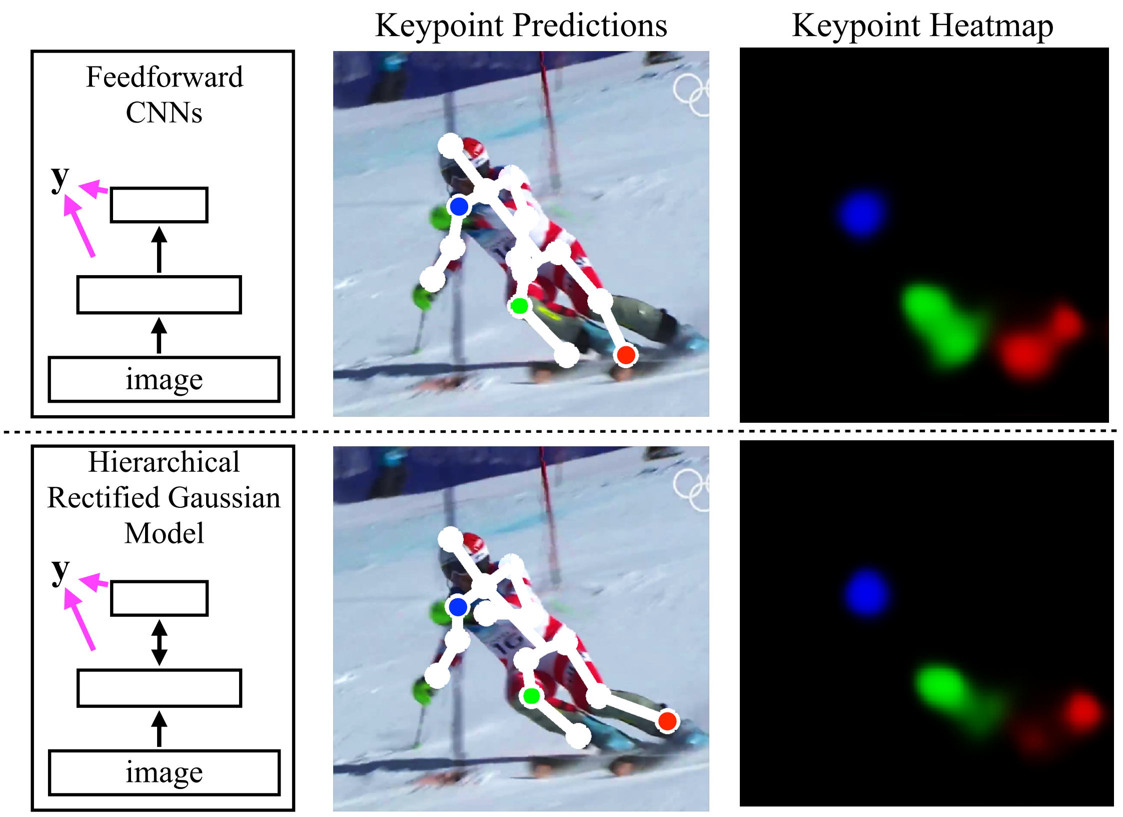


Figure 3.1: On the **top**, we show a state-of-the-art multi-scale feedforward net, trained for keypoint heatmap prediction, where the blue keypoint (the right shoulder) is visualized in the blue plane of the RGB heatmap. The ankle keypoint (red) is confused between left and right legs, and the knee (green) is poorly localized along the leg. We believe this confusion arises from bottom-up computations of neural activations in a feedforward network. On the **bottom**, we introduce hierarchical Rectified Gaussian (RG) models that incorporate top-down feedback by treating neural units as latent variables in a quadratic energy function. Inference on RGs can be unrolled into recurrent nets with rectified activations. Such architectures produce better features for “vision-with-scrutiny” tasks [72] (such as keypoint prediction) because lower-layers receive top-down feedback from above. Leg keypoints are much better localized with top-down knowledge (that may capture global constraints such as kinematic consistency).

part models can be written as a CNN [53], but the proposed mapping does not hold for feedback inference.

Overview: To endow CNNs with feedback, we treat neural units as nonnegative latent variables in a quadratic energy function. When probabilistically normalized, our quadratic energy function corresponds to a Rectified Gaussian (RG) distribution, for which inference can be cast as a quadratic program (QP) [164]. We demonstrate that coordinate descent optimization steps of the QP can be "unrolled" into a recurrent neural net with rectified linear units. This observation allows us to discriminatively-tune RGs with neural network toolboxes: *we tune Gaussian parameters such that, when latent variables are inferred from an image, the variables act as good features for discriminative tasks.* From a theoretical perspective, RGs help establish a connection between CNNs and hierarchical probabilistic models. From a practical perspective, we introduce RG variants of state-of-the-art deep models (such as VGG16 [163]) that require no additional parameters, but consistently improve performance due to the integration of top-down knowledge.

3.2 Hierarchical Rectified Gaussians

In this section, we describe the Rectified Gaussian models of Socci and Seung [164] and their relationship with rectified neural nets. Because we will focus on convolutional nets, it will help to think of variables $z = [z_i]$ as organized into layers, spatial locations, and channels (much like the neural activations of a CNN). We begin by defining a quadratic energy over variables z:

$$\begin{aligned} S(z) &= \frac{1}{2} z^T W z + b^T z \qquad (3.1) \\ P(z) &\propto e^{S(z)} \\ \text{Boltzmann:} \quad & z_i \in \{0,1\}, w_{ii} = 0 \\ \text{Gaussian:} \quad & z_i \in R, -W \text{ is PSD} \\ \text{Rect. Gaussian:} \quad & z_i \in R^+, -W \text{ is copositive} \end{aligned}$$

where $W = [w_{ij}]$, $b = [b_i]$. The symmetric matrix W captures bidirectional interactions between low-level features (e.g., edges) and high-level features (e.g., objects). Probabilistic models such as Boltzmann machines, Gaussians, and Rectified Gaussians differ simply in restrictions on the latent variable - binary, continuous, or nonnegative. Hierarchical models, such as deep Boltzmann machines [151], can be written as a special case of a block-sparse matrix W that ensures that only neighboring layers have direct interactions.

Normalization: To ensure that the scoring function can be probabilistically normalized, Gaussian models require that $(-W)$ be positive semidefinite (PSD) $(-z^T W z \geq 0, \forall z)$ Socci and Seung [164] show that Rectified Gaussians require the matrix $(-W)$ to only be *copositive* ($-z^T W z \geq 0, \forall z \geq 0$), which is a strictly weaker condition. Intuitively, copositivity ensures that the maximum of $S(z)$ is still finite, allowing one to compute the partition function.

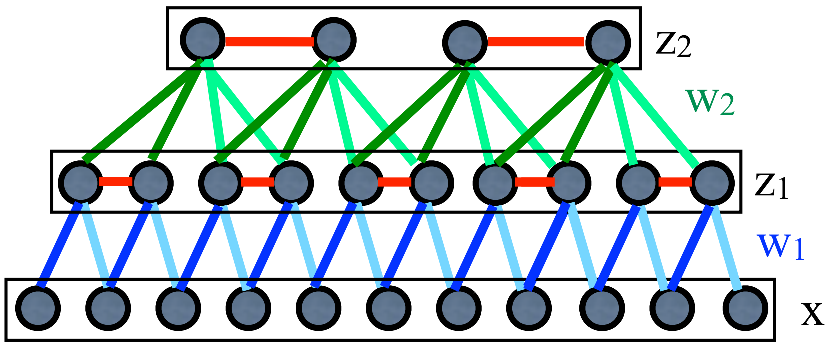


Figure 3.2: A hierarchical Rectified Gaussian model where latent variables z_i are denoted by circles, and arranged into layers and spatial locations. We write x for the input image and w_i for convolutional weights connecting layer $i-1$ to i. Lateral inhibitory connections between latent variables are drawn in red. Layer-wise coordinate updates are computed by filtering, rectification, and non-maximal suppression.

This relaxation significantly increases the expressive power of a Rectified Gaussian, allowing for multimodal distributions. We refer the reader to the excellent discussion in [164] for further details.

Comparison: Given observations (the image) in the lowest layer, we will infer the latent states (the features) from the above layers. Gaussian models are limited in that features will always be linear functions of the image. Boltzmann machines produce nonlinear features, but may be limited in that they pass only binary information across layers [126]. Rectified Gaussians are nonlinear, but pass continuous information across layers: z_i encodes the presence or absence of a feature, and if present, the strength of this activation (possibly emulating the firing rate of a neuron [83]).

Inference: Socci and Seung point out that MAP estimation of Rectified Gaussians can be formulated as a quadratic program (QP) with nonnegativity constraints [164]:

$$\max_{z\geq 0} \frac{1}{2} z^T W z + b^T z \tag{3.2}$$

However, rather than using projected gradient descent (as proposed by [164]), we show that coordinate descent is particularly effective in exploiting the sparsity of W. Specifically, let us optimize a single z_i holding all others fixed. Maximizing a 1-d quadratic function subject to non-negative constraints is easily done by solving for the optimum and clipping:

$$\begin{aligned}
\max_{z_i \geq 0} f(z_i) \quad \text{where} \quad f(z_i) &= \frac{1}{2} w_{ii} z_i^2 + (b_i + \sum_{j \neq i} w_{ij} z_j) z_i \\
\frac{\partial f}{\partial z_i} &= w_{ii} z_i + b_i + \sum_{j \neq i} w_{ij} z_j = 0 \\
z_i &= -\frac{1}{w_{ii}} \max(0, b_i + \sum_{j \neq i} w_{ij} z_j) \qquad (3.3) \\
&= \max(0, b_i + \sum_{j \neq i} w_{ij} z_j) \quad \text{for} \quad w_{ii} = -1
\end{aligned}$$

By fixing $w_{ii} = -1$ (which we do for all our experiments), the above maximization can solved with a rectified dot-product operation.

Layerwise-updates: The above updates can be performed for all latent variables in a layer in parallel. With a slight abuse of notation, let us define the input image to be the (observed) bottom-most layer $x = z_0$, and the variable at layer i and spatial position u is written as $z_i[u]$. The weight connecting $z_{i-1}[v]$ to $z_i[u]$ is given by $w_i[\tau]$, where $\tau = u - v$ depends only on the relative offset between u and v (visualized in Fig. 3.2):

$$\begin{aligned}
z_i[u] &= \max(0, b_i + top_i[u] + bot_i[u]) \quad \text{where} \qquad (3.4) \\
top_i[u] &= \sum_{\tau} w_{i+1}[\tau] z_{i+1}[u - \tau] \\
bot_i[u] &= \sum_{\tau} w_i[\tau] z_{i-1}[u + \tau]
\end{aligned}$$

where we assume that layers have a single one-dimensional channel of a fixed length to simplify notation. By tying together weights such that they only depend on relative locations, bottom-up signals can be computed with cross-correlational filtering, while top-down signals can be computed with convolution. In the existing literature, these are sometimes referred to as deconvolutional and convolutional filters (related through a 180° rotation) [201]. It is natural to start coordinate updates from the bottom layer z_1, initializing all variables to 0. During the initial bottom-up coordinate pass, top_i will always be 0. This means that the bottom-up coordinate updates can be computed with simple filtering and thresholding. *Hence a single bottom-up pass of layer-wise coordinate optimization of a Rectified Gaussian model can be implemented with a CNN.*

Top-down feedback: We add top-down feedback simply by applying additional coordinate updates (3.4) in a top-down fashion, from the top-most layer to the bottom. Fig. 3.3 shows that such a sequence of bottom-up and top-down updates can be "unrolled"

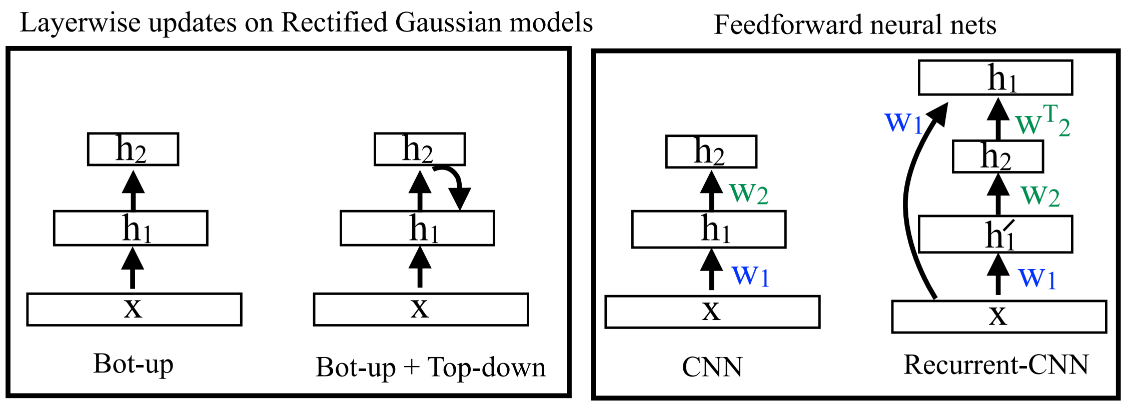


Figure 3.3: On the **left**, we visualize two sequences of layer-wise coordinate updates on our latent-variable model. The first is a bottom-up pass, while the second is a bottom-up + top-down pass. On the **right**, we show that bottom-up updates can be computed with a feed-forward CNN, and bottom-up-and-top-down updates can be computed with an "unrolled" CNN with additional skip connections and tied weights (which we define as a recurrent CNN). We use ^{T} to denote a 180° rotation of filters that maps correlation to convolution. We follow the color scheme from Fig. 3.2.

into a feed-forward CNN with "skip" connections between layers and tied weights. One can interpret such a model as a recurrent CNN that is capable of feedback, since lower-layer variables (capturing say, edges) can now be influenced by the activations of high-layer variables (capturing say, objects). Note that we make use of recurrence along the depth of the hierarchy, rather than along time or spacial dimensions as is typically done [63]. When the associated weight matrix W is copositive, an infinitely-deep recurrent CNN *must* converge to the solution of the QP from (3.2).

Non-maximal suppression (NMS): To encourage sparse activations, we add lateral inhibitory connections between variables from same groups in a layer. Specifically, we write the weight connecting $z_i[u]$ and $z_i[v]$ for $(u, v) \in$ group as $w_i[u, v] = -\infty$. Such connections are shown as red edges in Fig. 3.2. For disjoint groups (say, non-overlapping 2x2 windows), *layer-wise updates correspond to filtering, rectification* (3.4)*, and non-maximal suppression (NMS) within each group.*

Unlike max-pooling, NMS encodes the spatial location of the max by returning 0 values for non-maximal locations. Standard max-pooling can be obtained as a special case by replicating filter weights w_{i+1} across variables z_i within the same group (as shown in Fig. 3.2). This makes NMS independent of the top-down signal top_i. However, our approach is more general in that NMS can be guided by top-down feedback: high-level variables (e.g., car detections) influence the spatial location of low-level variables (e.g., wheels), which is particularly helpful when parsing occluded wheels. Interestingly, top-down feedback seems to encode spatial information without requiring additional "capsule" variables [71].

Approximate inference: Given the above global scoring function and an image x, inference corresponds to $\text{argmax}_z\, S(x, z)$. As argued above, this can be implemented with an infinitely-deep unrolled recurrent CNN. However, rather than optimizing the latent variables to completion, we perform a fixed number (k) of layer-wise coordinate descent updates. This is guaranteed to report back finite variables z^* for any weight matrix W (even when not copositive):

$$z^* = \mathbf{QP}_k(x, W, b), \quad z^* \in R^N \tag{3.5}$$

We write $\mathbf{QP}_k$ in bold to emphasize that it is a *vector-valued function* implementing k passes of layer-wise coordinate descent on the QP from (3.2), returning a vector of all N latent variables. We set $k = 1$ for a single bottom-up pass (corresponding to a standard feed-forward CNN) and $k = 2$ for an additional top-down pass. We visualize examples of recurrent CNNs that implement $\mathbf{QP}_1$ and $\mathbf{QP}_2$ in Fig. 3.4.

Output prediction: We will use these N variables as features for M recognition tasks. In our experiments, we consider the task of predicting heatmaps for M keypoints. Because our latent variables serve as rich, multi-scale description of image features, we assume that simple linear predictors built on them will suffice:

$$y = V^T z^*, \quad y \in R^M, V \in R^{N\times M} \tag{3.6}$$

Training: Our overall model is parameterized by (W, V, b). Assume we are given training data pairs of images and output label vectors $\{x_i, y_i\}$. We define a training objective as follows

$$\min_{W,V,b} R(W) + R(V) + \sum_i \text{loss}(y_i, V^T \mathbf{QP}_k(x_i, W, b)) \tag{3.7}$$

where R are regularizer functions (we use the Frobenius matrix norm) and "loss" sums the loss of our M prediction tasks (where each is scored with log or softmax loss). We optimize the above by stochastic gradient descent. Because $\mathbf{QP}_k$ is a deterministic function, its gradient with respect to (W, b) can be computed by backprop on the k-times unrolled recurrent CNN (Fig. 3.3). We choose to separate V from W to ensure that feature extraction does not scale with the number of output tasks ($\mathbf{QP}_k$ is independent of M). During learning, we fix diagonal weights ($w_i[u, u] = -1$) and lateral inhibition weights ($w_i[u, v] = -\infty$ for $(u, v) \in$ group).

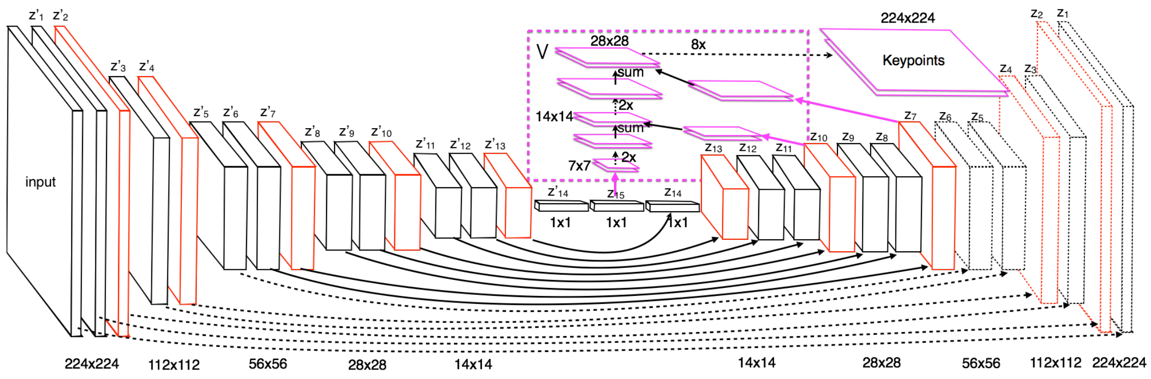


Figure 3.4: We show the architecture of $\mathbf{QP}_2$ implemented in our experiments. $\mathbf{QP}_1$ corresponds to the left half of $\mathbf{QP}_2$, which essentially resembles the state-of-the-art VGG-16 CNN [163]. $\mathbf{QP}_2$ is implemented with an 2X "unrolled" recurrent CNN with transposed weights, skip connections, and zero-interlaced upsampling (as shown in Fig. 3.5). Importantly, $\mathbf{QP}_2$ does not require any additional parameters. Red layers include lateral inhibitory connections enforced with NMS. Purple layers denote multi-scale convolutional filters that (linearly) predict keypoint heatmaps given activations from different layers. Multi-scale filters are efficiently implemented with coarse-to-fine upsampling [108], visualized in the purple dotted rectangle (to reduce clutter, we visualize only 3 of the 4 multiscale layers). Dotted layers are not implemented to reduce memory.

Related work (learning): The use of gradient-based backpropagation to learn an unrolled model dates back to 'backprop-through-structure' algorithms [56, 165] and graph transducer networks [102]. More recently, such approaches were explored general graphical models [169] and Boltzmann machines [57]. Our work uses such ideas to learn CNNs with top-down feedback using an unrolled latent-variable model.

Related work (top-down): Prior work has explored networks that reconstruct images given top-down cues. This is often cast as unsupervised learning with autoencoders [70, 118, 185] or deconvolutional networks [201], though supervised variants also exist [108, 128]. Our network differs in that all nonlinear operations (rectification and max-pooling) are influenced by both bottom-up and top-down knowledge (3.4), which is justified from a latent-variable perspective.

3.3 Implementation

In this section, we provide details for implementing $\mathbf{QP}_1$ and $\mathbf{QP}_2$ with existing CNN toolboxes. We visualize our specific architecture in Fig. 3.4, which closely follows the state-of-the-art VGG-16 network [163]. We use 3x3 filters and 2x2 non-overlapping pooling

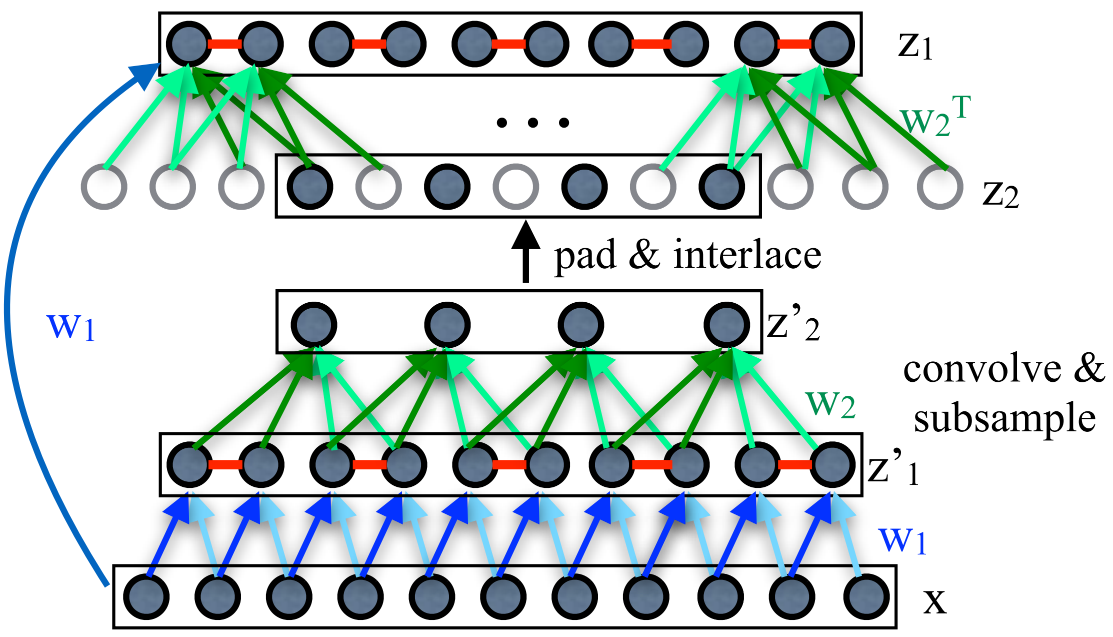


Figure 3.5: Two-pass layer-wise coordinate descent for a two-layer Rectified Gaussian model can be implemented with modified CNN operations. White circles denote 0's used for interlacing and border padding. We omit rectification operations to reduce clutter. We follow the color scheme from Fig. 3.2.

windows (for NMS). Note that, when processing NMS-layers, we conceptually use 6x6 filters with replication after NMS, which in practice can be implemented with standard max-pooling and 3x3 filters (as argued in the previous section). Hence $\mathbf{QP}_1$ *is* essentially a re-implementation of VGG-16.

$\mathbf{QP}_2$**:** Fig. 3.5 illustrates top-down coordinate updates, which require additional feedforward layers, skip connections, and tied weights. Even though $\mathbf{QP}_2$ is twice as deep as $\mathbf{QP}_1$ (and [163]), *it requires no additional parameters.* Hence top-down reasoning "comes for free". There is a small notational inconvenience at layers that decrease in size. In typical CNNs, this decrease arises from a previous pooling operation. Our model requires an explicit $2\times$ subsampling step (sometimes known as strided filtering) because it employs NMS instead of max-pooling. When this subsampled layer is later used to produce a top-down signal for a future coordinate update, variables must be zero-interlaced before applying the $180°$ rotated convolutional filters (as shown by hollow circles in Fig. 3.5). Note that is *not* an approximation, but the mathematically-correct application of coordinate descent given subsampled weight connections.

Supervision y: The target label for a single keypoint is a sparse 2D heat map with a '1' at the keypoint location (or all '0's if that keypoint is not visible on a particular training image). We score this heatmap with a per-pixel log-loss. In practice, we assign '1's to a circular neighborhood that implicitly adds jittered keypoints to the set of positive examples.

Multi-scale classifiers V: We implement our output classifiers (3.7) as multi-scale convolutional filters defined over different layers of our model. We use upsampling to enable efficient coarse-to-fine computations, as described for fully-convolutional networks (FCNs) [108] (and shown in Fig. 3.4). Specifically, our multi-scale filters are implemented as 1×1 filters over 4 layers (referred to as fc7, pool4, pool3, and pool2 in [163]). Because our top (fc7) layer is limited in spatial resolution (1x1x4096), we define our coarse-scale filter to be "spatially-varying", which can alternatively be thought of as a linear "fully-connected" layer that is reshaped to predict a coarse (7x7) heatmap of keypoint predictions given fc7 features. Our intuition is that spatially-coarse global features can still encode global constraints (such as viewpoints) that can produce coarse keypoint predictions. This coarse predictions are upsampled and added to the prediction from pool4, and so on (as in [108]).

Multi-scale training: We initialize parameters of both $\mathbf{QP}_1$ and $\mathbf{QP}_2$ to the pre-trained VGG-16 model[163], and follow the coarse-to-fine training scheme for learning FCNs [108]. Specifically, we first train coarse-scale filters, defined on high-level (fc7) variables. Note that $\mathbf{QP}_1$ and $\mathbf{QP}_2$ are equivalent in this setting. This coarse-scale model is later used to initialize a two-scale predictor, where now $\mathbf{QP}_1$ and $\mathbf{QP}_2$ differ. The process is repeated up until the full multi-scale model is learned. To save memory during various stages of learning, we only instantiate $\mathbf{QP}_2$ up to the last layer used by the multi-scale predictor (not suitable for $\mathbf{QP}_k$ when $k > 2$). We use a batch size of 40 images, a fixed learning rate of 10^{-6}, momentum of 0.9 and weight decay of 0.0005. We also decrease learning rates of parameters built on lower scales [108] by a factor of 10. Batch normalization[77] is used before each non-linearity. Both our models and code are available online [1].

Prior work: We briefly compare our approach to recent work on keypoint prediction that make use of deep architectures. Many approaches incorporate multi-scale cues by evaluating a deep network over an image pyramid [176, 177, 180]. Our model processes only a single image scale, extracting multi-scale features from multiple layers of a single network, where importantly, fine-scale features are refined through top-down feedback. Other approaches cast the problem as one of regression, where (x,y) keypoint locations are predicted [207] and often iteratively refined [18, 171]. Our models predict heatmaps, which can be thought of as *marginal distributions* over the (x,y) location of a keypoint, capturing uncertainty. We show that by thresholding the heatmap value (certainty), one can also

[1]https://github.com/peiyunh/rg-mpii

produce *keypoint visibility* estimates "for free". Our comments hold for our bottom-up model $\mathbf{QP}_1$, which can be thought of as a FCN tuned for keypoint heatmap prediction, rather than semantic pixel labeling. Indeed, we find such an approach to be a surprisingly simple but effective baseline that outperforms much prior work.

3.4 Experiment Results

We evaluated fine-scale keypoint localization on several benchmark datasets of human faces and bodies. To better illustrate the benefit of top-down feedback, we focus on datasets with significant occlusions, where bottom-up cues will be less reliable. All datasets provide a rough detection window for the face/body of interest. We crop and resize detection windows to 224x224 before feeding into our model. Recall that $\mathbf{QP}_1$ is essentially a re-implementation of a FCN [108] defined on a VGG-16 network [163], and so represents quite a strong baseline. Also recall that $\mathbf{QP}_2$ adds top-down reasoning *without any increase in the number of parameters*. We will show this consistently improves performance, sometimes considerably. Unless otherwise stated, results are presented for a 4-scale multi-scale model.

AFLW: The AFLW dataset [94] is a large-scale real-world collection of 25,993 faces in 21,997 real-world images, annotated with facial keypoints. Notably, these faces are not limited to be responses from an existing face detector, and so this dataset contains more pose variation than other landmark datasets. We hypothesized that such pose variation might illustrate the benefit of bidirectional reasoning. Due to a lack of standard splits, we randomly split the dataset into training (60%), validation (20%) and test (20%). As this is not a standard benchmark dataset, we compare to ourselves for exploring the best practices to build multi-scale predictors for keypoint localization (Fig. 3.7). We include qualitative visualizations in Fig. 3.6.

COFW: Caltech Occluded Faces-in-the-Wild (COFW) [16] is dataset of 1007 face images with severe occlusions. We present qualitative results in Fig. 3.8 and Fig. 3.9, and quantitative results in Table 3.1 and Fig. 3.10. Our bottom-up $\mathbf{QP}_1$ already performs near the state-of-the-art, while the $\mathbf{QP}_2$ significantly improves in accuracy of visible landmark localization and occlusion prediction. In terms of the latter, our model even approaches upper bounds that make use of ground-truth segmentation labels [51]. Our models are not quite state-of-the-art in localizing occluded points. We believe this may point to a limitation in the underlying benchmark. Consider an image of a face mostly occluded by the hand (Fig. 3.8). In such cases, humans may not even agree on keypoint locations, indicating that a keypoint *distribution* may be a more reasonable target output. Our models provide such uncertainty estimates, while most keypoint architectures based on regression cannot.

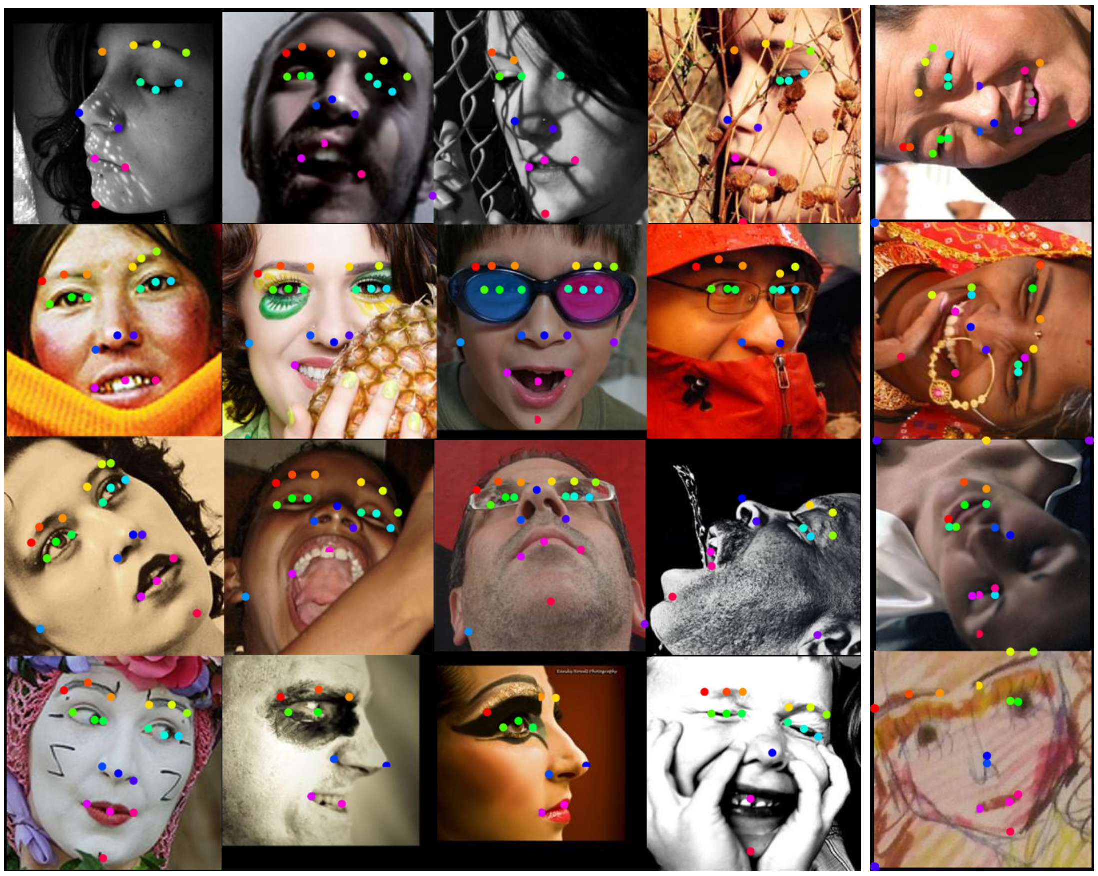


Figure 3.6: Facial landmark localization results of $\mathbf{QP}_2$ on AFLW, where landmark ids are denoted by color. We only plot landmarks annotated visible. Our bidirectional model is able to deal with large variations in illumination, appearance and pose (**a**). We show images with multiple challenges present in (**b**).

Pascal Person: The Pascal 2011 Person dataset [60] consists of 11,599 person instances, each annotated with a bounding box around the visible region and up to 23 human keypoints per person. This dataset contains significant occlusions. We follow the evaluation protocol of [110] and present results for localization of visible keypoints on a standard testset in Table 3.2. Our bottom-up $\mathbf{QP}_1$ model already significantly improves upon the state-of-the-art (including prior work making use of deep features), while our top-down models $\mathbf{QP}_2$ further improve accuracy by 2% without any increase in model complexity (as measured by the number of parameters). Note that the standard evaluation protocols evaluate only visible keypoints. In Fig. 3.11, we demonstrate that our model can also accurately predict keypoint visibility "for free".

	Visible Points	All Points
RCPR[16]	-	8.5
RPP[196]	-	7.52
HPM[50]	-	7.46
SAPM[51]	5.77	6.89
FLD-Full[191]	5.18	**5.93**
$\mathbf{QP}_1$	5.26	10.06
$\mathbf{QP}_2$	**4.67**	7.87

Table 3.1: Average keypoint localization error (as a fraction of inter-ocular distance) on COFW. When adding top-down feedback ($\mathbf{QP}_2$), our accuracy on visible keypoints significantly improves upon prior work. In the text, we argue that such localization results are more meaningful than those for occluded keypoints. In Fig. 3.10, we show that our models significantly outperform all prior work in terms of keypoint visibility prediction.

α	0.10	0.20
CNN+prior[110]	47.1	-
$\mathbf{QP}_1$	66.5	78.9
$\mathbf{QP}_2$	**68.8**	**80.8**

Table 3.2: We show human keypoint localization performance on PASCAL VOC 2011 Person following the evaluation protocol in [110]. PCK refers to the fraction of keypoints that were localized within some distance (measured with respect to the instance's bounding box). Our bottom-up models already significantly improve results across all distance thresholds ($\alpha = 10, 20\%$). Our top-down models add a 2% improvement without increasing the number of parameters.

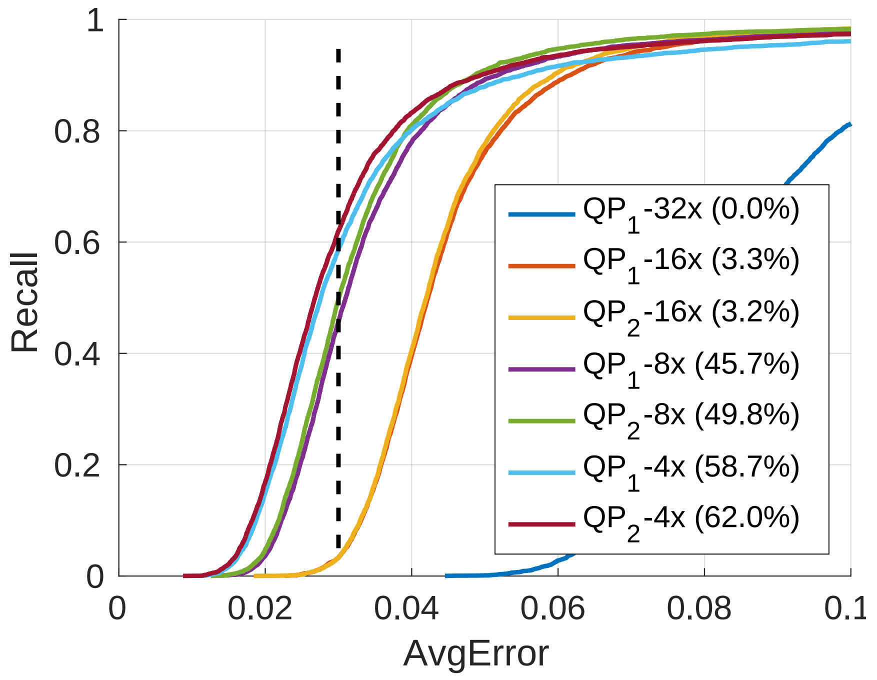


Figure 3.7: We plot the fraction of recalled face images whose average pixel localization error in AFLW (normalized by face size [212]) is below a threshold (x-axis). We compare our $\mathbf{QP}_1$ and $\mathbf{QP}_2$ with varying numbers of scales used for multi-scale prediction, following the naming convention of FCN [108] (where the Nx encodes the upsampling factor needed to resize the predicted heatmap to the original image resolution.) Single-scale models ($\mathbf{QP}_1$-32x and $\mathbf{QP}_2$-32x) are identical but perform quite poorly, not localizing any keypoints with 3.0% of the face size. Adding more scales dramatically improves performance, and moreover, as we add additional scales, the relative improvement of $\mathbf{QP}_2$ also increases (as finer-scale features benefit the most from feedback). We visualize such models in Fig. 3.12.

MPII: MPII is (to our knowledge) the largest available articulated human pose dataset [4], consisting of 40,000 people instances annotated with keypoints, visibility flags, and activity labels. We present qualitative results in Fig. 3.14 and quantitative results in Table 3.3. Our top-down model $\mathbf{QP}_2$ appears to outperform all prior work on full-body keypoints. Note that this dataset also includes visibility labels for keypoints, even though these are not part of the standard evaluation protocol. In Fig. 3.13, we demonstrate that visibility prediction on MPII also benefits from top-down feedback.

TB: It is worth contrasting our results with TB [178], which implicitly models feedback by (1) using a MRF to post-process CNN outputs to ensure kinematic consistency between keypoints and (2) using high-level predictions from a coarse CNN to adaptively crop high-res features for a fine CNN. Our single CNN endowed with top-down feedback is slightly more

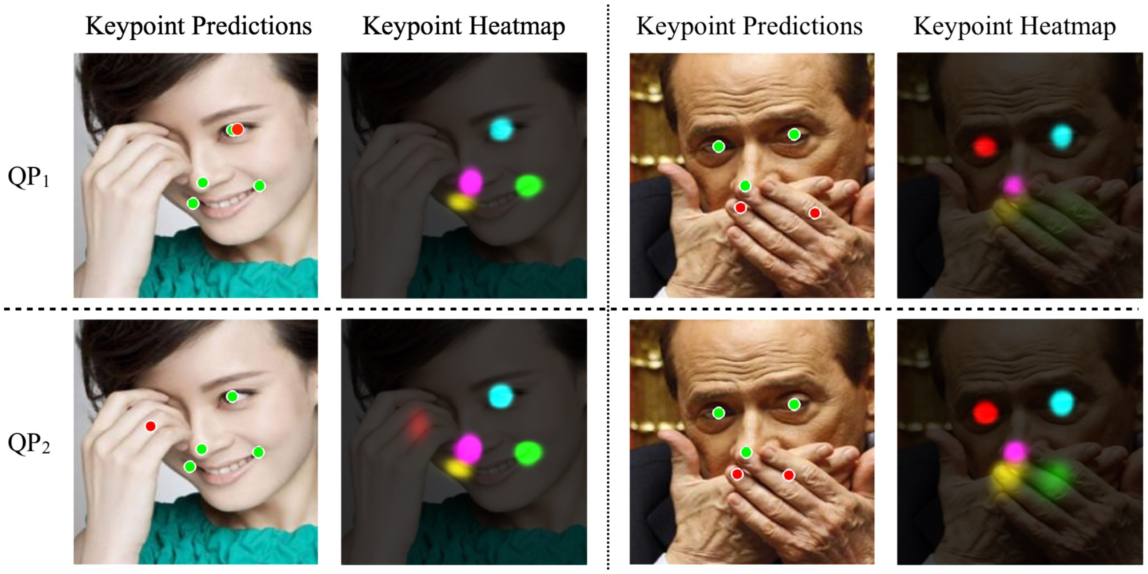


Figure 3.8: Visualization of keypoint predictions by $\mathbf{QP}_1$ and $\mathbf{QP}_2$ on two example COFW images. Both our models predict both keypoint locations and their visibility (produced by thresholding the value of the heatmap confidence at the predicted location). We denote (in)visible keypoint predictions with (red)green dots, and also plot the raw heatmap prediction as a colored distribution overlayed on a darkened image. Both our models correctly estimate keypoint visibility, but our bottom-up models $\mathbf{QP}_1$ misestimate their locations (because bottom-up evidence is misleading during occlusions). By integrating top-down knowledge (perhaps encoding spatial constraints on configurations of keypoints), $\mathbf{QP}_2$ is able to correctly estimate their locations.

	Head	Shou	Elb	Wri	Hip	Kne	Ank	Upp	Full
GM [54]	-	36.3	26.1	15.3	-	-	-	25.9	-
ST [152]	-	38.0	26.3	19.3	-	-	-	27.9	-
YR [199]	73.2	56.2	41.3	32.1	36.2	33.2	34.5	43.2	44.5
PS [136]	74.2	49.0	40.8	34.1	36.5	34.4	35.1	41.3	44.0
TB [178]	96.1	91.9	83.9	77.8	80.9	72.3	64.8	**84.5**	82.0
$\mathbf{QP}_1$	94.3	90.4	81.6	75.2	80.1	73.0	68.3	82.4	81.1
$\mathbf{QP}_2$	95.0	91.6	83.0	76.6	81.9	74.5	69.5	83.8	**82.4**

Table 3.3: We show PCKh-0.5 keypoint localization results on MPII using the recommended benchmark protocol [4].

accurate without requiring any additional parameters, while being 2X faster (86.5 ms vs TB's 157.2 ms). These results suggest that top-down reasoning may elegantly capture structured outputs and attention, two active areas of research in deep learning.

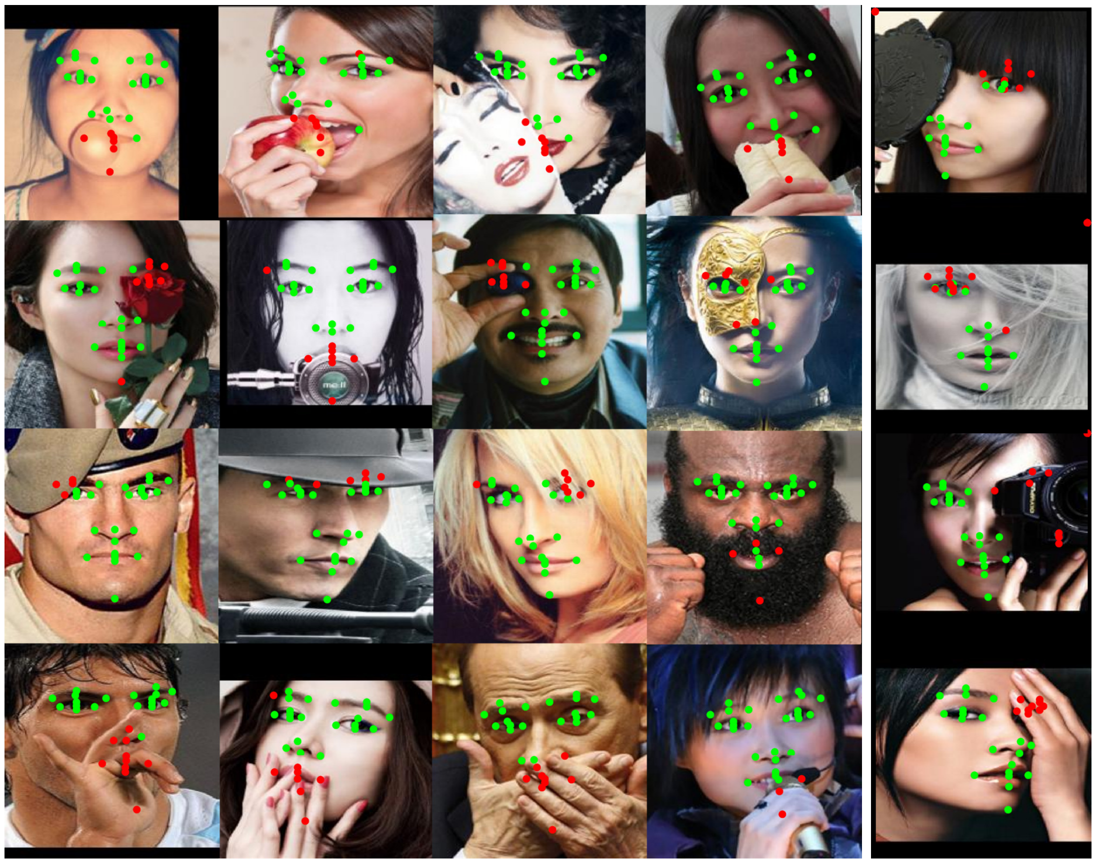


Figure 3.9: Facial landmark localization and occlusion prediction results of $\mathbf{QP}_2$ on COFW, where red means occluded. Our bidirectional model is robust to occlusions caused by objects, hair, and skin. We also show cases where the model correctly predicts visibility but fails to accurately localize occluded landmarks (**b**).

K	1	2	3	4	5	6
Upper Body	57.8	59.6	58.7	**61.4**	58.7	60.9
Full Body	59.8	62.3	61.0	**63.1**	61.2	62.6

Table 3.4: PCKh(.5) on MPII-Val for a smaller network

More recurrence iterations: To explore $\mathbf{QP}_K$'s performance as a function of K without exceeding memory limits, we trained a smaller network from scratch on 56X56 sized inputs for 100 epochs. As shown in Table 3.4, we conclude: (1) all recurrent models outperform the bottom-up baseline $\mathbf{QP}_1$; (2) additional iterations generally helps, but performance maxes out at $\mathbf{QP}_4$. A two-pass model ($\mathbf{QP}_2$) is surprisingly effective at capturing top-down info while being fast and easy to train.

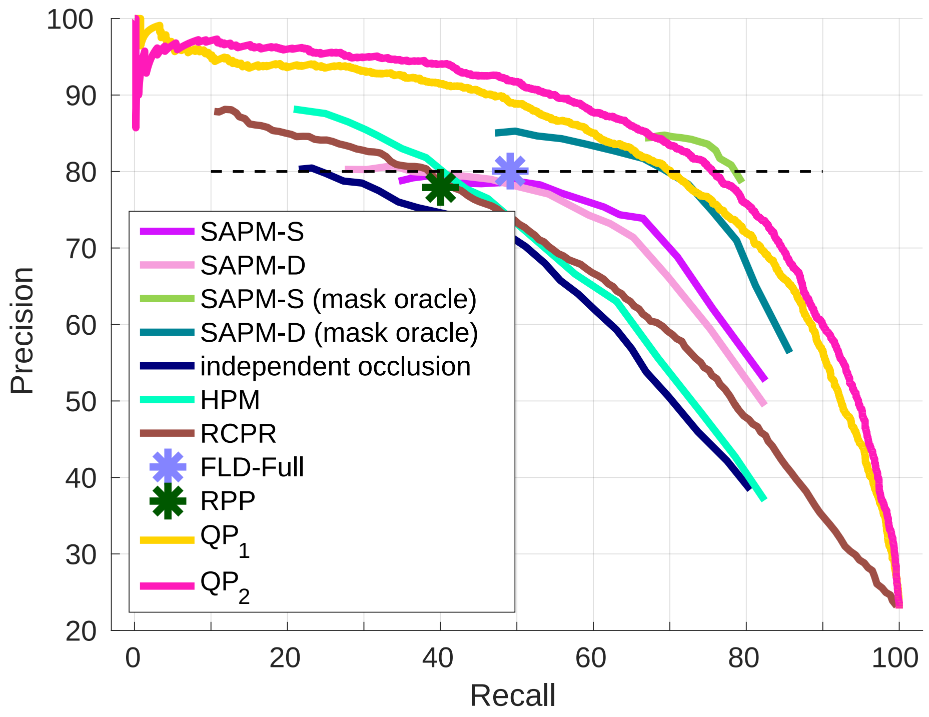


Figure 3.10: Keypoint visibility prediction on COFW, measured by precision-recall. Our bottom-up model $\mathbf{QP}_1$ already outperforms all past work that does not make use of ground-truth segmentation masks (where acronyms correspond those in Table 3.1). Our top-down model $\mathbf{QP}_2$ even approaches the accuracy of such upper bounds. Following standard protocol, we evaluate and visualize accuracy in Fig. 3.9 at a precision of 80%. At such a level, our recall (76%) significantly outperform the best previously-published recall of FLD [191] (49%).

Conclusion: We show that hierarchical Rectified Gaussian models can be optimized with rectified neural networks. From a modeling perspective, this observation allows one to discriminatively-train such probabilistic models with neural toolboxes. From a neural net perspective, this observation provides a theoretically-elegant approach for endowing CNNs with top-down feedback – *without any increase in the number of parameters.* To thoroughly evaluate our models, we focus on "vision-with-scrutiny" tasks such as keypoint localization, making use of well-known benchmark datasets. We introduce (near) state-of-the-art bottom-up baselines based on multi-scale prediction, and consistently improve upon those results with top-down feedback (particularly during occlusions when bottom-up evidence may be ambiguous).

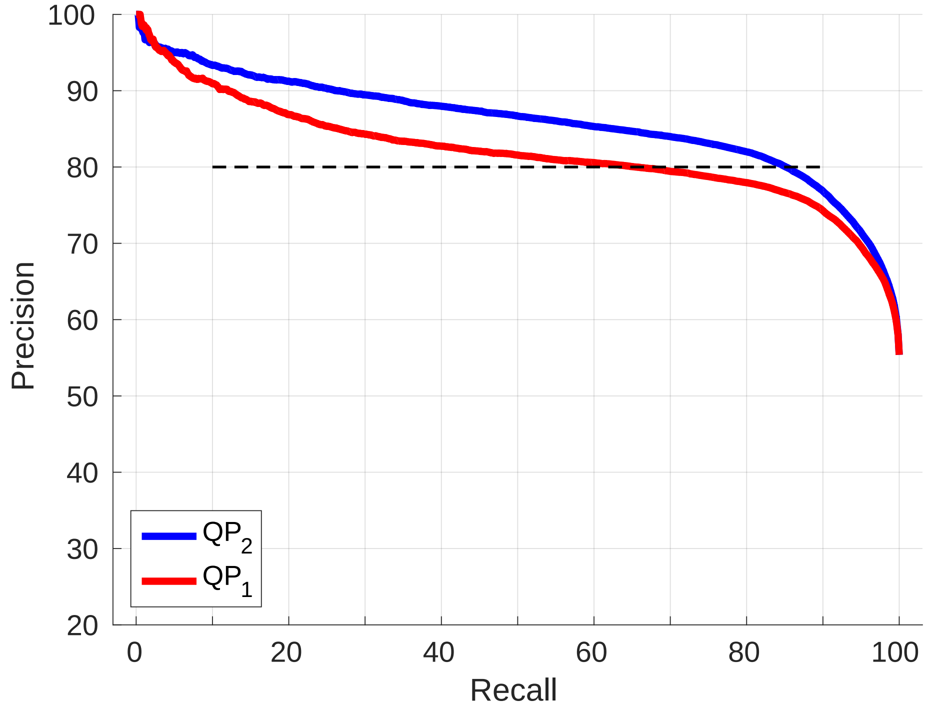


Figure 3.11: Keypoint visibility prediction on Pascal Person (a dataset with significant occlusion and truncation), measured by precision-recall curves. At 80% precision, our top-down model (QP_2) significantly improves recall from 65% to 85%.

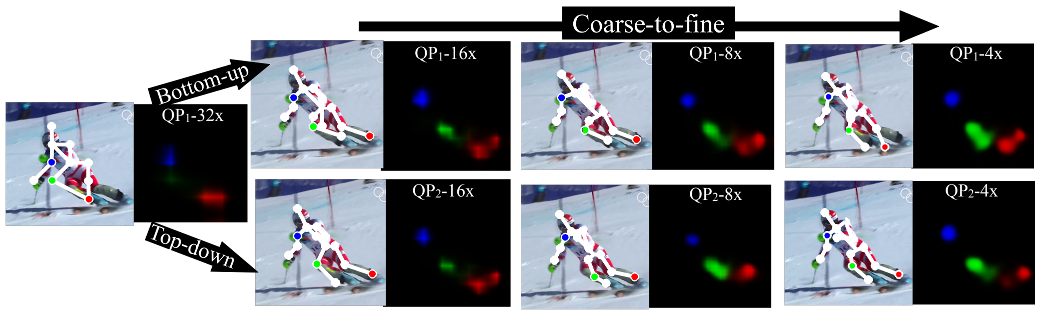


Figure 3.12: We visualize bottom-up and top-down models trained for human pose estimation, using the naming convention of Fig. 3.7. Top-down feedback ($\mathbf{QP}_2$) more accurately guides finer-scale predictions, resolving left-right ambiguities in the ankle (red) and poor localization of the knee (green) in the bottom-up model ($\mathbf{QP}_1$).

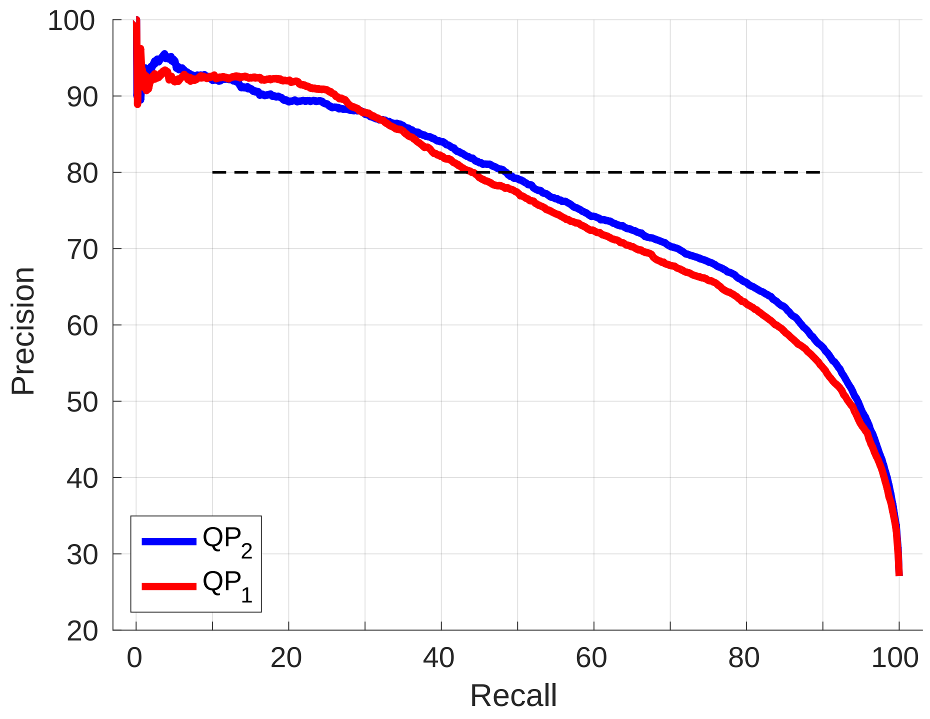


Figure 3.13: Keypoint visibility prediction on MPII, measured by precision-recall curves. At 80% precision, our top-down model (QP_2) improves recall from 44% to 49%.

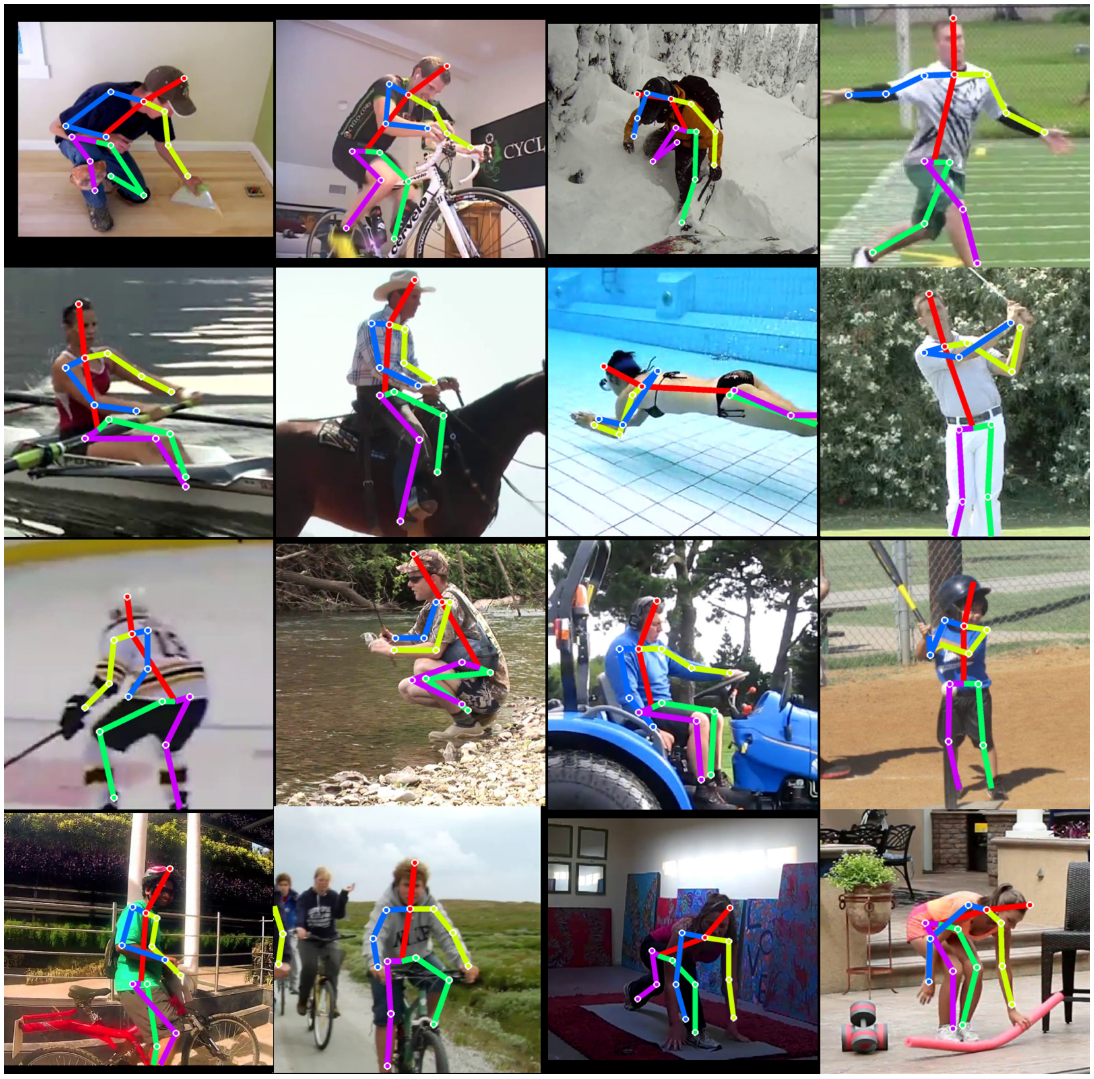


Figure 3.14: Keypoint localization results of $\mathbf{QP}_2$ on the MPII Human Pose testset. We quantitatively evaluate results on the validation set in Table 3.2. Our models are able to localize keypoints even under significant occlusions. Recall that our models can also predict visibility labels "for free", as shown in Fig. 3.13.

Chapter 4

Exploiting Visibility For Occlusion

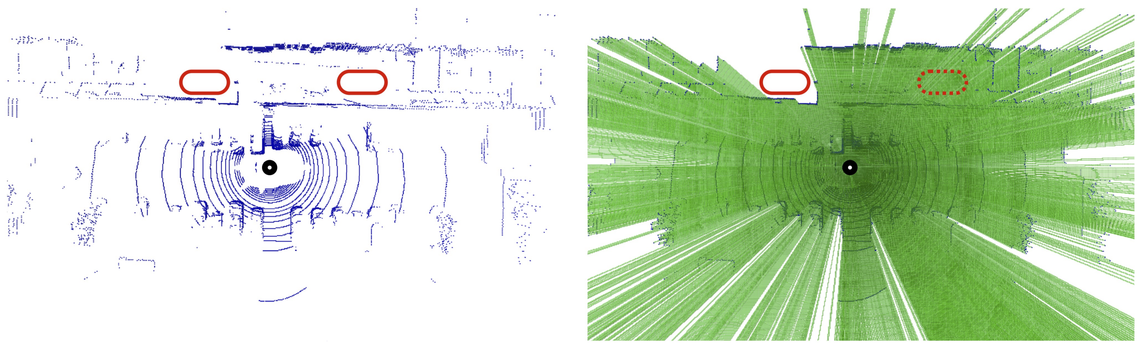

(a) (b)

Figure 4.1: What is a good representation for 3D sensor data? We visualize a bird's-eye-view LiDAR scene and highlight two regions that may contain an object. Many contemporary deep networks process 3D point clouds, making it hard to distinguish the two regions (**left**). But depth sensors provide more than 3D points - they provide estimates of freespace in between the sensor and the measured 3D point. We visualize freespace by raycasting (**right**), where green is free and white is unknown. In this paper, we introduce deep 3D networks that leverage freespace to significantly improve 3D object detection accuracy.

4.1 Introduction

What is a good representation for processing 3D sensor data? While this is a fundamental challenge in machine vision dating back to stereoscopic processing, it has recently been explored in the context of deep neural processing of 3D sensors such as LiDARs. Various representations have been proposed, including graphical meshes [14], point clouds [138], voxel grids [209], and range images [120], to name a few.

Visibility: We revisit this question by pointing out that 3D sensored data, is infact, not fully 3D! Instantaneous depth measurements captured from a stereo pair, structured light sensor, or LiDAR undeniably suffer from occlusions: once a particular scene element is measured at a particular depth, visibility ensures that all other scene elements behind it along its line-of-sight are occluded. Indeed, this loss of information is one of the fundamental reasons why 3D sensor readings can often be represented with 2D data structures - e.g., 2D range image. From this perspective, such 3D sensored data might be better characterized as "2.5D" [117].

3D Representations: We argue that representations for processing LiDAR data should embrace visibility, particularly for applications that require instantaneous understanding of freespace (such as autonomous navigation). However, most popular representations are based on 3D point clouds (such as PointNet [100, 138]). Because these were often proposed in the context of truly 3D processing (e.g., of 3D mesh models), they do not exploit visibility constraints implicit in the sensored data (Fig. 7.1). Indeed, representing a LiDAR sweep as a collection of (x, y, z) points fundamentally *destroys* such visibility information if normalized (e.g., when centering point clouds).

Occupancy: By no means are we the first to point out the importance of visibility. In the context of LiDAR processing, visibility is well studied for the tasks of map-building and occupancy reasoning [75, 174]. However, it is not well-explored for object detection, with one notable exception: [200] builds a probabilistic occupancy grid and performs template matching to directly estimate the probability of an object appearing at each discretized location. However, this approach requires knowing surface shape of object instances beforehand, therefore it is not scalable. In this paper, we demonstrate that deep architectures can be simply augmented to exploit visibility and freespace cues.

Range images: Given our arguments above, one solution might be defining a deep network on 2D range image input, which *implicitly* encodes such visibility information. Indeed, this representation is popular for structured light "RGBD" processing [40, 89], and has also been proposed for LiDAR [120]. However, such representations do not seem to produce state-of-the-art accuracy for 3D object understanding, compared to 3D voxel-based or top-down, bird's-eye-view (BEV) projected grids. We posit that convolutional layers that operate along a depth dimension can reason about uncertainty in depth. To maintain this property, we introduce simple but novel approaches that directly augment state-of-the-art 3D voxel representations with visibility cues.

Our approach: We propose a deep learning approach that efficiently augments point clouds with visibility. Our specific constributions are three-fold; (1) We first (re)introduce raycasting algorithms that effciently compute on-the-fly visibility for a voxel grid. We demonstrate that these can be incorporated into batch-based gradient learning. (2) Next, we describe a simple approach to augmenting voxel-based networks with visibility: we add a voxelized visibility map as an additional input stream, exploring alternatives for early and late fusion; (3) Finally, we show that visibility can be combined with two crucial modifications common to state-of-the-art networks: synthetic data augmentation of virtual objects, and temporal aggregation of LiDAR sweeps over multiple time frames. We show that visibility cues can be used to better place virtual objects. We also demonstrate that visibility reasoning over multiple time frames is akin to online occupancy mapping.

4.2 Related Work

4.2.1 3D Representations

Point representation: Most classic works on point representation employ *hand-crafted* descriptors and require robust estimates of local surface normals, such as spin-images [80] and Viewpoint Feature Histograms (VFH) [149]. Since PointNet [138], there has been a line of work focuses on learning better point representation, including PointNet++[140], Kd-networks [92], PointCNN [105], EdgeConv [187], and PointConv [190] to name a few. Recent works on point-wise representation tend not to distinguish between *reconstructed* and *measured* point clouds. We argue that when the input is a *measured* point cloud, e.g. a LiDAR sweep, we need to look beyond points and reason about visibility that is hidden within points.

Visibility representation: Most research on visibility representation has been done in the context of robotic mapping. For example, Buhmann et al. [15] estimates a 2D probabilistic occupancy map from sonar readings to navigate the mobile robot and more recently Hornung et al. [75] have developed Octomap for general purpose 3D occupancy mapping. Visibility through raycasting is at the heart of developing such occupancy maps. Despite the popularity, such visibility reasoning has not been widely studied in the context of object detection, except a notable exception of [200], which develops a probabilistic framework based on occupancy maps to detect objects with known surface models.

4.2.2 LiDAR-based 3D Object Detection

Initial representation: We have seen LiDAR-based object detectors built upon range images, bird's-eye-view feature maps, raw point clouds, and also voxelized point clouds. One example of a range image based detector is LaserNet [120], which treats each LiDAR sweep as a cylindrical range image. Examples of bird-eye-view detectors include AVOD [97],

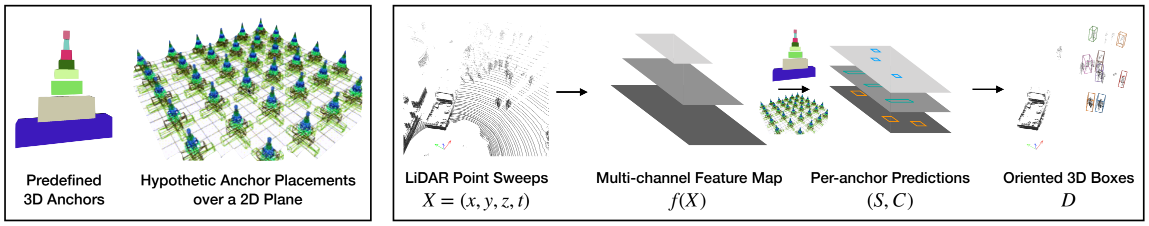

Predefined 3D Anchors
Hypothetic Anchor Placements over a 2D Plane
LiDAR Point Sweeps $X = (x, y, z, t)$
Multi-channel Feature Map $f(X)$
Per-anchor Predictions (S, C)
Oriented 3D Boxes D

Figure 4.2: Overview of a general 3D detection framework, designed to solve 3D detection as a bird's-eye-view (BEV) 2D detection problem. The framework consists of two parts: anchors (**left**) and network (**right**). We first define a set of 3D anchor boxes that match the average box shape of different object classes. Then we hypothesize placing each anchor at different spatial locations over a ground plane. We learn a convolutional network to predict confidence and adjustments for each anchor placement. Such predictions are made based on 2D multi-channel feature maps, extracted from the input 3D point cloud. The predictions for each anchor consist of a confidence score S and a set of coefficients C for adjusting the anchor box. Eventually, the framework produces a set of 3D detections with oriented 3D boxes.

HDNet [195], and Complex-YOLO [162]. One example that builds upon raw point clouds is PointRCNN [160]. Examples of voxelized point clouds include the initial VoxelNet[209], SECOND [193], and PointPillars [100]. Other than [200], we have not seen a detector that uses visibility as the initial representation.

Object augmentation: Yan et al. [193] propose a novel form of data augmentation, which we call *object augmentation*. It copy-pastes object point clouds from one scene into another, resulting in new training data. This augmentation technique improves both convergence speed and final performance and is adopted in all recent state-of-the-art 3D detectors, such as PointRCNN [160] and PointPillars [100]. For objects captured under the same sensor setup, simple copy-paste preserves the relative pose between the sensor and the object, resulting in approximately correct return patterns. However, such practice often inserts objects regardless of whether it violates the scene visibility. In this paper, we propose to use visibility reasoning to maintain correct visibility while augmenting objects across scenes.

Temporal aggregation: When learning 3D object detectors over a series of LiDAR sweeps, it is proven helpful to aggregate information across time. Luo et al. [114] develop a recurrent architecture for detecting, tracking, and fore-casting objects on LiDAR sweeps. Choy et al. [23] propose to learn spatial-temporal reasoning through 4D ConvNets. Another technique for temporal aggregation, first found in SECOND [193], is to simply aggregate point clouds from different sweeps while preserving their timestamps relative to the current one. These timestamps are treated as additional per-point input feature along with (x, y, z)

and fed into point-wise encoders such as PointNet. We explore temporal aggregation over visibility representations and point out that one can borrow ideas from classic robotic mapping to integrate visibility representation with learning.

4.3 Exploit Visibility for 3D Object Detection

Before we discuss how to integrate visibility reasoning into 3D detection, we first introduce a general 3D detection framework. Many 3D detectors have adopted this framework, including AVOD [97], HDNet [195], Complex-YOLO [162], VoxelNet [209], SECOND [193], and PointPillars [100]. Among the more recent ones, there are two crucial innovations: (1) object augmentation by inserting rarely seen (virtual) objects into training data and (2) temporal aggregation of LiDAR sweeps over multiple time frames.

We integrate visibility into the aforementioned 3D detection framework. First, we (re)introduce a raycasting algorithm that efficiently computes visibility. Then, we introduce a simple approach to integrate visibility into the existing framework. Finally, we discuss visibility reasoning within the context of object augmentation and temporal aggregation. For object augmentation, we modify the raycasting algorithm to make sure visibility remains valid while inserting virtual objects. For temporal aggregation, we point out that visibility reasoning over multiple frames is akin to online occupancy mapping.

4.3.1 A General Framework for 3D Detection

Overview: We illustrate the general 3D detection framework in Fig. 6.1. Please refer to the caption. We highlight the fact that once the input 3D point cloud is converted to a multi-channel BEV 2D representation, we can make use of standard 2D convolutional architectures. We later show that visibility can be naturally incorporated into this 3D detection framework.

Object augmentation: Data augmentation is a crucial ingredient of contemporary training protocols. Most augmentation strategies perturb coordinates through random transformations (e.g. translation, rotation, flipping) [97, 139]. We focus on *object augmentation* proposed by Yan et al. [193], which copy-pastes (virtual) objects of rarely-seen classes (such as buses) into LiDAR scenes. Our ablation studies (g$\rightarrow$i in Tab. 4.3) suggest that it dramatically improves vanilla PointPillars by an average of **+9.1%** on the augmented classes.

Temporal aggregation: In LiDAR-based 3D detection, researchers have explored various strategies for temporal reasoning. We adopt a simple method that aggregates (motion-compensated) points from different LiDAR sweeps into a single scene [17, 193].

Importantly, points are augmented with an additional channel that encodes the relative timestamp (x, y, z, t). Our ablation studies (g→j in Tab. 4.3) suggest that temporal aggregation dramatically improves the overall mAP of vanilla PointPillars model by **+8.6%**.

4.3.2 Compute Visibility through Raycasting

Physical raycasting in LiDAR: Each LiDAR point is generated through a physical raycasting process. To generate a point, the sensor emits a laser pulse in a certain direction. The pulse travels through air forward and back after hitting an obstacle. Upon its return, one can compute a 3D coordinate derived from the direction and the time-of-flight. However, coordinates are by no means the only information offered by such active sensing. Crucially, it also provides estimates of freespace along the ray of the pulse.

Simulated LiDAR raycasting: By exploiting the causal relationship between freespace and point returns - points lie along the ray where freespace ends, we can re-create the instantaneous visibility encountered at the time of LiDAR capture. We do so by drawing a line segment from the sensor origin to a 3D point. We would like to use this line segment to define freespace across a discretized volume, e.g. a 3D voxel grid. Specifically, we compute all voxels that intersect this line segment. Those that are encountered along the ray are marked as free, except the last voxel enclosing the 3D point is marked as occupied. This results in a visibility volume where all voxels are marked as occupied, free, or unknown (default). We will integrate the visibility volume into the general detection framework (Fig. 6.1) in the form of a *multi-channel 2D feature map* (e.g. a RGB image is an example with 3 channels) where visibility along the vertical dimension (z-axis) is treated as different channels.

Efficient voxel traversal: Visibility computation must be extremely efficient. Many detection networks exploit sparsity in LiDAR point clouds: PointPillars[100] process only non-empty pillars (about 3%) and SECOND [193] employs spatially sparse 3D ConvNets. Inspired by these approaches, we exploit sparsity through an efficient voxel traversal algorithm [3]. For any given ray, we need traverse only a sparse set of voxels along the ray. Intuitively, during the traversal, the algorithm enumerates over the six axis-aligned faces of the current voxel to determine which is intersected by the exiting ray (which is quite efficient). It then simply advances to the neighboring voxel with a shared face. The algorithm begins at the voxel at the origin and terminates when it encounters the (precomputed) voxel occupied by the 3D point. This algorithm is linear in the grid dimension, making it quite efficient. Given an *instantaneous* point cloud, where points are captured at the same timestamp, we perform raycasting from the origin to each point and aggregate voxels' visibility afterwards. To reduce discretization effects during aggregation, we follow best-practices outlined in

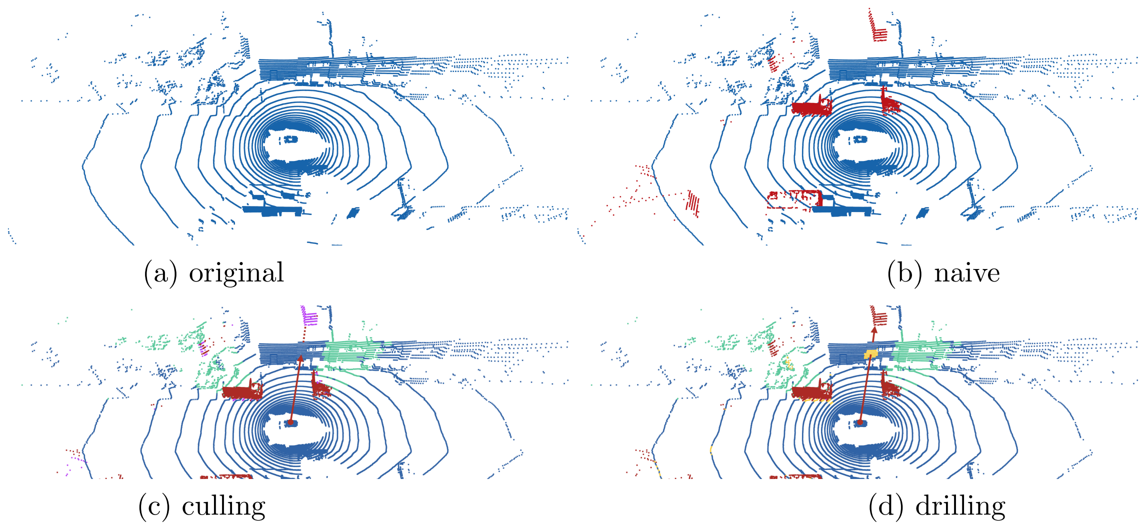


Figure 4.3: Different types of object augmentation we can do through visibility reasoning. In (a), we show the original LiDAR point cloud. In (b), we naively insert new objects (red) into the scene. Clearly, the naive strategy may result in inconsistent visibility. Here, a trailer is inserted behind a wall that should occlude it. We use raycasting as a tool to "rectify" the LiDAR sweep. In (c), we illustrate the culling strategy, where we remove virtual objects that are occluded (purple). In practice, this may excessively remove augmented objects. In (d), we visualize the drilling strategy, where we remove points from the original scene that occlude the virtual objects. Here, a small piece of wall is removed (yellow).

Octomap (Sec. 5.1 in [75]).

Raycasting with augmented objects: Prior work augments virtual objects while ignoring visibility constraints, producing LiDAR sweeps with inconsistent visibility (e.g., by inserting an object behind a wall that should occlude it - Fig. 4.3-(b)). We can use raycasting as a tool to "rectify" the LiDAR sweep. Specifically, we might wish to remove virtual objects that are occluded (a strategy we term *culling* - Fig. 4.3-(c)). Because this might excessively decrease the number of augmented objects, another option is to remove points from the original scene that occlude the inserted objects (a strategy we term *drilling* - Fig. 4.3-(d)).

Fortunately, as we show in Alg. 1, both strategies are efficient to implement with simple modifications to the vanilla voxel traversal algorithm. We only have to change the terminating condition of raycasting from arriving at the end point of the ray to hitting a voxel that is `BLOCKED`. For *culling*, when casting rays from the original scene, we set voxels occupied by virtual objects as `BLOCKED`; when casting rays from the virtual objects, we set voxels occupied in original scenes as `BLOCKED`. As a result, points that should be occluded will be removed. For *drilling*, we allow rays from virtual objects to pass through voxels

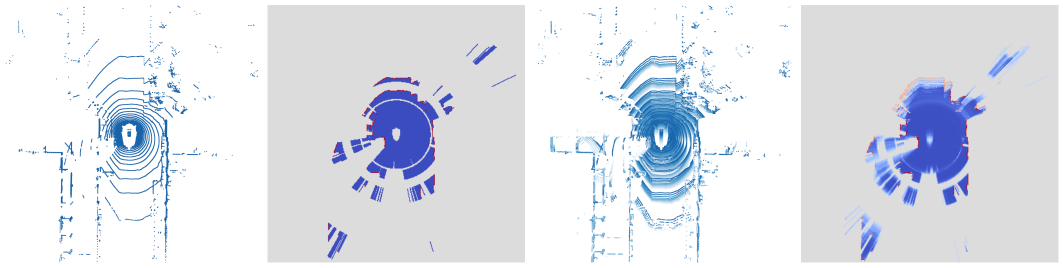


Figure 4.4: We visualize instantaneous visibility vs. temporal occupancy. We choose one xy-slice in the middle to visualize. Each pixel represents a voxel on the slice. On the **left**, we visualize a single LiDAR sweep and the instantaneous visibility, which consists of three discrete values: occupied (red), unknown (gray), and free (blue). On the **right**, we visualize aggregated LiDAR sweeps plus temporal occupancy, computed through Bayesian Filtering [75]. Here, the color encodes the probability of the corresponding voxel being occupied: redder means more occupied.

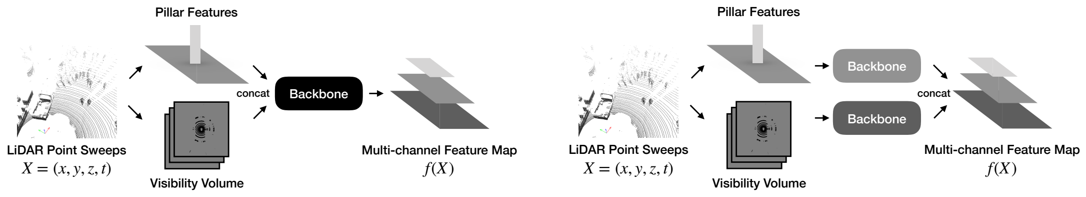


Figure 4.5: We explore both early fusion and late fusion when integrating visibility into the PointPillars model. In the early fusion (a), we concatenate visibility volume with pillar features before applying a backbone network for further encoding. For late fusion, we build one separate backbone network for each stream and concatenate the output of each stream into a final multi-channel feature map. We compare these two alternatives in ablation studies (Tab. 4.3).

occupied in the original scene.

Online occupancy mapping: How do we extend instantaneous visibility into a temporal context? Assume knowing the sensor origin at each timestamp, we can compute instantaneous visibility over every sweep, resulting in 4D spatial-temporal visibility. If we directly integrate a 4D volume into the detection framework, it would be too expensive. We seek out online occupancy mapping [75, 175] and apply Bayesian filtering to turn a 4D spatial-temporal visibility into a 3D posterior probability of occupancy. In Fig. 4.4, we plot a visual comparison between instantaneous visibility and temporal occupancy. We follow Octomap [75]'s formulation and use their off-the-shelf hyper-parameters, e.g. the log-odds of observing freespace and occupied space.

Table 4.1: 3D detection mAP on the NuScenes test set.

	car	pedes.	barri.	traff.	truck	bus	trail.	const.	motor.	bicyc.	mAP
PointPillars [17]	68.4	59.7	**38.9**	**30.8**	23.0	28.2	23.4	4.1	**27.4**	**1.1**	30.5
Ours	**79.1**	**65.0**	34.7	28.8	**30.4**	**46.6**	**40.1**	**7.1**	18.2	0.1	**35.0**

4.3.3 Approach: A Two-stream Network

Now that we have discussed raycasting approaches for computing visibility, we introduce a novel two-stream network for 3D object detection. We incorporate visibility into a state-of-the-art 3D detector, i.e. PointPillars, as an additional stream. The two-stream approach leverages both the point cloud and the visibility representation and fuses them into a multi-channel representation. We explore both early and late fusion strategies, as illustrated in Fig. 4.5. This is a part of the overall architecture illustrated in Fig. 6.1.

Implementation: We implement our two-stream approach by adding an additional input stream to PointPillars. We adopt PointPillars's resolution for discretization in order to improve ease of integration. As a result, our visibility volume has the same 2D spatial size as the pillar feature maps. A simple strategy is to concatenate and feed them into a backbone network. We refer to this strategy as early fusion (Fig. 4.5-(a)). Another strategy is to feed each into a separate backbone network, which we refer to as late fusion (Fig. 4.5-(b)). We discuss more training details in the Appendix **??** . Our code is available online[1].

4.4 Experiments

We present both qualitative (Fig. 5.4) and quantitative results on the NuScenes 3D detection benchmark. We first introduce the setup and baselines, before we present main results on the test benchmark. Afterwards, we perform diagnostic evaluation and ablation studies to pinpoint where improvements come from. Finally, we discuss the efficiency of computing visibility through raycasting on-the-fly.

Setup: We benchmark our approach on the NuScenes 3D detection dataset. The dataset contains 1,000 scenes captured in two cities. We follow the official protocol for NuScenes detection benchmark. The training set contains 700 scenes (28,130 annotated frames). The validation set contains 150 scenes (6,019 annotated frames). Each annotated frame comes with one LiDAR point cloud captured by a 32-beam LiDAR, as well as up to 10

[1]`https://www.cs.cmu.edu/~peiyunh/wysiwyg`

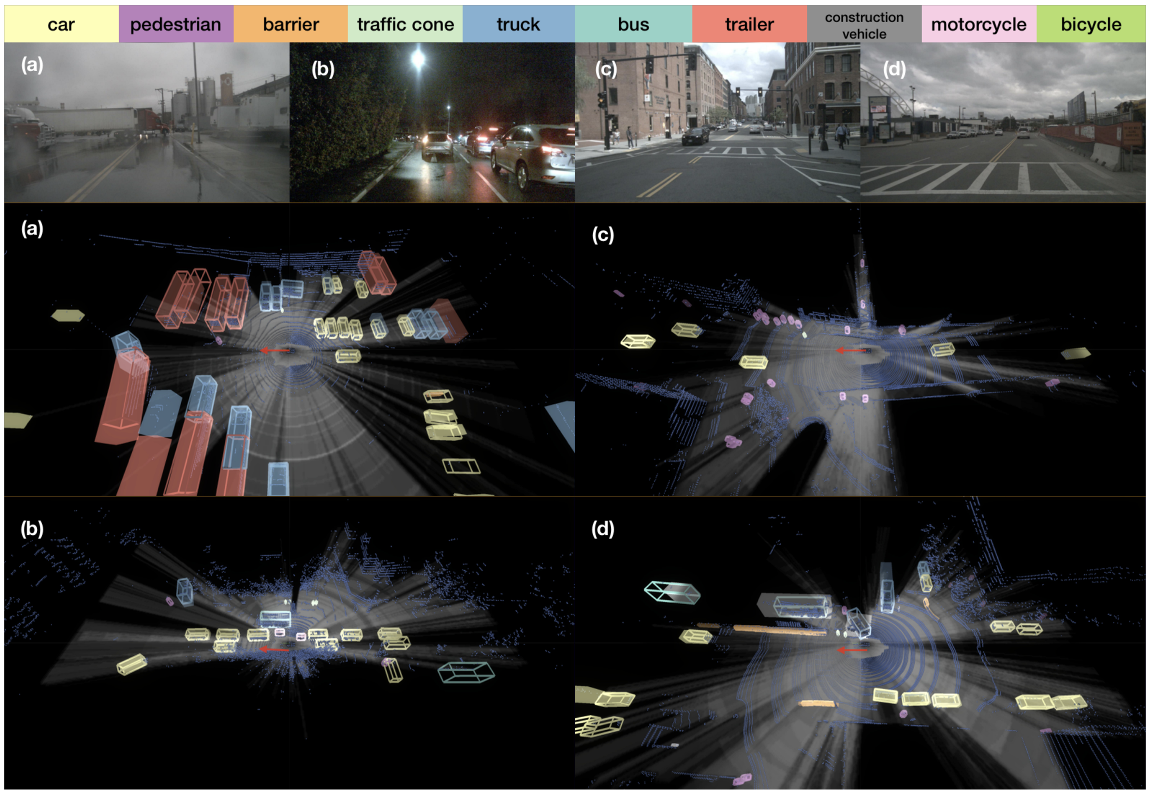


Figure 4.6: We visualize qualitative results of our two-stream approach on the NuScenes test set. We assign each class a different color (top). We use solid cuboids to represent ground truth objects and wireframe boxes to represent predictions. To provide context, we also include an image captured by the front camera in each scenario. Note the image is **not** used as part of input for our approach. In (a), our approach successfully detects most vehicles in the scene on a rainy day, including cars, trucks, and trailers. In (b), our model manages to detect all the cars around and also two motorcycles on the right side. In (c), we visualize a scene with many pedestrians on the sidewalk and our model is able to detect most of them. Finally, we demonstrate a failure case in (d), where our model fails to detect objects from rare classes. In this scenario, our model fails to detect two construction vehicles on the car's right side, reporting one as a truck and the other as a bus.

frames of (motion-compensated) point cloud. We follow the official evaluation protocol for 3D detection [17] and evaluate average mAP over different classes and distance threshold.

Baseline: PointPillars [100] achieves the best accuracy on the NuScenes detection leaderboard among all published methods that have released source code. The official PointPillars codebase[2] only contains an implementation for KITTI [49]. To reproduce PointPillars's results on NuScenes, the authors of PointPillars recommend a third-party implementation[3].Using an off-the-shelf configuration provided by the third-part implementation, we train a PointPillars model for 20 epochs from scratch on the full training set and use it as our baseline. This model achieves an overall mAP of 31.5% on the validation set, which is 2% higher than the official PointPillars mAP (29.5%) [17] (Tab. 4.2). As suggested by [17], the official implementation of PointPillars employ pretraining (ImageNet/KITTI). There is no pretraining in our re-implementation.

Main results: We submitted the results of our two-stream approach to the NuScenes test server. In Tab. 4.1, we compare our test-set performance to PointPillars on the official leaderboard [17]. By augmenting visibility, our proposed approach achieves a significant improvement over PointPillars in overall mAP by a margin of 4.5%. Specifically, our approach outperforms PointPillars by 10.7% on cars, 5.3% on pedestrians, 7.4% on trucks, 18.4% on buses, and 16.7% on trailers. Our model underperforms official PointPillars on motorcycles by a large margin. We hypothesize this might be due to us (1) using a coarser xy-resolution or (2) not pretraining on ImageNet/KITTI.

Improvement at different levels of visibility: We compare our two-stream approach to PointPillars on the validation set, where visibility improves overall mAP by 4%. We also evaluate each object class at different levels of visibility. Here, we focus on the two most common classes: car and pedestrian. Interestingly, we observe the biggest improvement over heavily occluded cars (0-40% visible) and the smallest improvement over fully-visible cars (80-100% visible). For pedestrian, we also find the smallest improvement is over fully-visible pedestrians (3.2%), which is 1-3% less than the improvement over pedestrians with heavier occlusion.

Ablation studies: To understand how much improvement each component provides, we perform ablation studies by starting from our final model and removing one component at a time. Key observations from Tab. 4.3 are:

- **Early fusion** (**a**,b): Replacing early fusion (a) with late fusion (b) results in a 1.4% drop in overall mAP.
- **Drilling** (**b**,c,d): Replacing drilling (b) with culling (c) results in a 11.4% drop on

[2]`https://github.com/nutonomy/second.pytorch`
[3]`https://github.com/traveller59/second.pytorch`

Table 4.2: 3D detection mAP on the NuScenes validation set.
†: reproduced based on an author-recommended third-party implementation.

	car	pedes.	barri.	traff.	truck	bus	trail.	const.	motor.	bicyc.	mAP
PointPillars [17]	70.5	59.9	33.2	**29.6**	25.0	34.4	16.7	4.5	**20.0**	**1.6**	29.5
PointPillars†	76.9	62.6	29.2	20.4	32.6	49.6	27.9	3.8	11.7	0.0	31.5
Ours	**80.0**	**66.9**	**34.5**	27.9	**35.8**	**54.1**	**28.5**	**7.5**	18.5	0.0	**35.4**

car	0-40%	40-60%	60-80%	80-100%
Proportion	20%	12%	15%	54%
PointPillars†	27.2	40.0	57.2	84.3
Ours	32.1	42.6	60.6	86.3
Improvement	**4.9**	2.6	3.4	2.0

pedestrian	0-40%	40-60%	60-80%	80-100%
Proportion	20%	12%	15%	54%
PointPillars†	17.3	23.4	28.0	68.3
Ours	22.1	27.8	34.2	71.5
Improvement	4.8	4.4	**6.2**	3.2

bus and a 4.9% drop on trailer. In practice, most augmented trucks and trailers tend to be severely occluded and are removed if the culling strategy is applied. Replacing drilling (b) with naive augmentation (d) results in a 1.9% drop on bus and 3.1% drop on trailer, likely due to inconsistent visibility when naively augmenting objects.

- **Object augmentation** (**b**,e): Removing object augmentation (b→e) leads to significant drops in mAP on classes affected by object augmentation, including in a 2.5% drop on truck, 13.7% on bus, and 7.9% on trailer.
- **Temporal aggregation** (**e**,f): Removing temporal aggregation (e→f) leads to worse performance for every class and a 9.4% drop in overall mAP.
- **Visibility stream** (**f**,g,h): Removing the visibility stream off a *vanilla* two-stream approach (f→g) drops overall mAP by 1.4%. Interestingly, the most dramatic drops are over pedestrian (+7.5%), barrier(+3.3%), and traffic cone (+3.7%). Shape-wise, these objects are all "skinny" and tend to have less LiDAR points on them. This suggests visibility helps especially when having less points. The network with only a visibility stream (h) underperforms a *vanilla* PointPillars (g) by 4%.
- **Vanilla PointPillars** (**g**,i,j,k): On top of vanilla PointPillars, object augmentation

Table 4.3: Ablation studies on the NuScenes validation set. We *italicize* classes for which we perform object augmentation. OA stands for object augmentation and TA stands for temporal aggregation.

	Fusion	OA	TA	car	pedes.	barri.	traff.	*truck*	*bus*	*trail.*	const.	motor.	bicyc.	avg
(a)	Early	Drilling	Multi-frame	80.0	66.9	34.5	27.9	35.8	54.1	28.5	7.5	18.5	0.0	35.4
(b)	Late	Drilling	Multi-frame	77.8	65.8	32.2	24.2	33.7	53.0	30.6	4.1	18.8	0.0	34.0
(c)	Late	Culling	Multi-frame	78.3	66.4	33.2	27.3	33.4	41.6	25.7	5.6	17.0	0.1	32.9
(d)	Late	Naive	Multi-frame	78.2	66.0	32.7	25.6	33.6	51.1	27.5	4.7	15.0	0.1	33.5
(e)	Late	N/A	Multi-frame	77.9	66.8	31.3	22.3	31.2	39.3	22.7	5.2	15.5	0.6	31.3
(f)	Late	N/A	Single-frame	67.9	45.7	24.0	12.4	22.6	29.9	8.5	1.3	7.1	0.0	21.9
(g)	No V	N/A	Single-frame	68.0	38.2	20.7	8.7	23.7	28.7	11.0	0.6	5.6	0.0	20.5
(h)	Only V	N/A	Single-frame	66.7	28.6	15.8	4.4	17.0	25.4	6.7	0.0	1.3	0.0	16.6
(i)	No V	Naive	Single-frame	69.7	38.7	22.5	11.5	28.1	40.7	21.8	1.9	4.7	0.0	24.0
(j)	No V	N/A	Multi-frame	77.7	61.6	26.4	17.2	31.2	38.5	24.2	3.1	11.5	0.0	29.1
(k)	No V	Naive	Multi-frame	76.9	62.6	29.2	20.4	32.6	49.6	27.9	3.8	11.7	0.0	31.5

(g$\rightarrow$i) improves mAP over augmented classes by 9.1%; temporal aggregation (g$\rightarrow$j) improves overall mAP by 8.6%. Adding both (g$\rightarrow$k) improves overall mAP by 11.0%.

Run-time speed: We implement visibility computation in C++ and integrate it into PyTorch training as part of (parallel) data loading. On an Intel i9-9980XE CPU, it takes 24.4$\pm$3.5ms on average to compute visibility for a 32-beam LiDAR point cloud when running on a single CPU thread.

Conclusions: We revisit the problem of finding a good representation for 3D data. We point out that contemporary representations are designed for true 3D data (e.g. sampled from mesh models). In fact, 3D sensored data such as a LiDAR sweep is 2.5D. By processing such data as a collection of *normalized* points (x, y, z), important visibility information is fundementally destroyed. In this paper, we augment visibility into 3D object detection. We first demonstrate that visibility can be efficiently re-created through 3D raycasting. We introduce a simple two-stream approach that adds visibility as a separate stream to an existing state-of-the-art 3D detector. We also discuss the role of visibility in placing virtual objects for data augmentation and explore visibility in a temporal context - building a local occupancy map in an online fashion. Finally, on the NuScenes detection benchmark, we demonstrate that the proposed network outperforms state-of-the-art detectors by a significant margin.

Algorithm 1 Raycasting with Augmented Objects

```
Input: mode $\mathbf{m}$, sensor origin $\mathbf{s}$, original points $\mathbf{P}$, augmented points $\mathbf{Q}$
Output: occupancy grid $\mathbf{O}$
Initialize: $\mathbf{O}[:] \leftarrow$ UNKNOWN  /* Raycast P with Q as a ray stopper */
Compute $\mathbf{B}$ such that $\forall \mathbf{q}$ in $\mathbf{Q}, \mathbf{B}[v_\mathbf{q}] \leftarrow$ BLOCKED  for $\mathbf{p}$ in $\mathbf{P}$ do
    $v \leftarrow v_\mathbf{s}$;                                   /* $v_s$: sensor voxel */
    while $v \neq v_\mathbf{p}$ do
        $v \leftarrow$ next_voxel$(v, \mathbf{p} - \mathbf{s})$  if $\mathbf{B}[v] =$ BLOCKED then
            break;                                        /* stop the ray */
        end
        if $v = v_\mathbf{p}$ then
            $\mathbf{O}[v] \leftarrow$ OCCUPIED
        else
            $\mathbf{O}[v] \leftarrow$ FREE
        end
    end
end
/* Raycast Q with P as a ray stopper */
Compute $\mathbf{B}$ such that $\forall \mathbf{q}$ in $\mathbf{Q}, \mathbf{B}[v_\mathbf{q}] \leftarrow$ BLOCKED  for $\mathbf{q}$ in $\mathbf{Q}$ do
    $v \leftarrow v_\mathbf{s}$;                                   /* $v_s$: sensor voxel */
    while $v \neq v_\mathbf{q}$ do
        $v \leftarrow$ next_voxel$(v, \mathbf{q} - \mathbf{s})$  if $\mathbf{B}[v] =$ BLOCKED then
            if $\mathbf{m} =$ CULLING then
                break;                                    /* stop the ray */
            else if $\mathbf{m} =$ DRILLING then
                $\mathbf{O}[v] \leftarrow$ FREE;                  /* let ray through */
            /* Do nothing under the naïve mode */
        end
        if $v = v_\mathbf{q}$ then
            $\mathbf{O}[v] \leftarrow$ OCCUPIED
        else
            $\mathbf{O}[v] \leftarrow$ FREE
        end
    end
end
```

Chapter 5

Exploiting Geometric Priors for Novel Objects

5.1 Introduction

Perception for autonomous robots presents a collection of compelling challenges for computer vision. We focus on the application of autonomous vehicles. This domain has three notable properties that tend not to surface in traditional vision applications: (1) 3D sensing in the form of LiDAR technology, which exhibits different properties than traditional 3D vision captured through stereo or structured light. Despite significant work in this area, the right representation for such sparse 3D signals still remains an open question. (2) Contemporary approaches to object detection and scene understanding tend to be closed-world, where the task is predicting 1-of-N possible labels. But autonomous systems require the ability to recognize all possible obstacles and movers - e.g., a piece of road debris must be avoided regardless of what *name* it has. Such understanding is crucial from a safety perspective. Historically, this has been formulated as a perceptual grouping or bottom-up segmentation task, which is typically addressed with different approaches. (3) Finally, practical autonomous robotics makes heavy use of perceptual priors in the forms of geometric maps and assumptions on LiDAR geometry. Indeed, prior map was a crucial component among finishing entries in the DARPA Urban Grand Challenge [124, 181].

Motivation: In this work, we focus on the problem of class-agnostic instance segmentation of LiDAR point clouds (Figure 7.1) in an open-world setting. We carefully mix graph-theoretic algorithms with data-driven learning. Data-driven learning has made an undeniable impact on computer vision, but it is difficult to make guarantees about perfor-

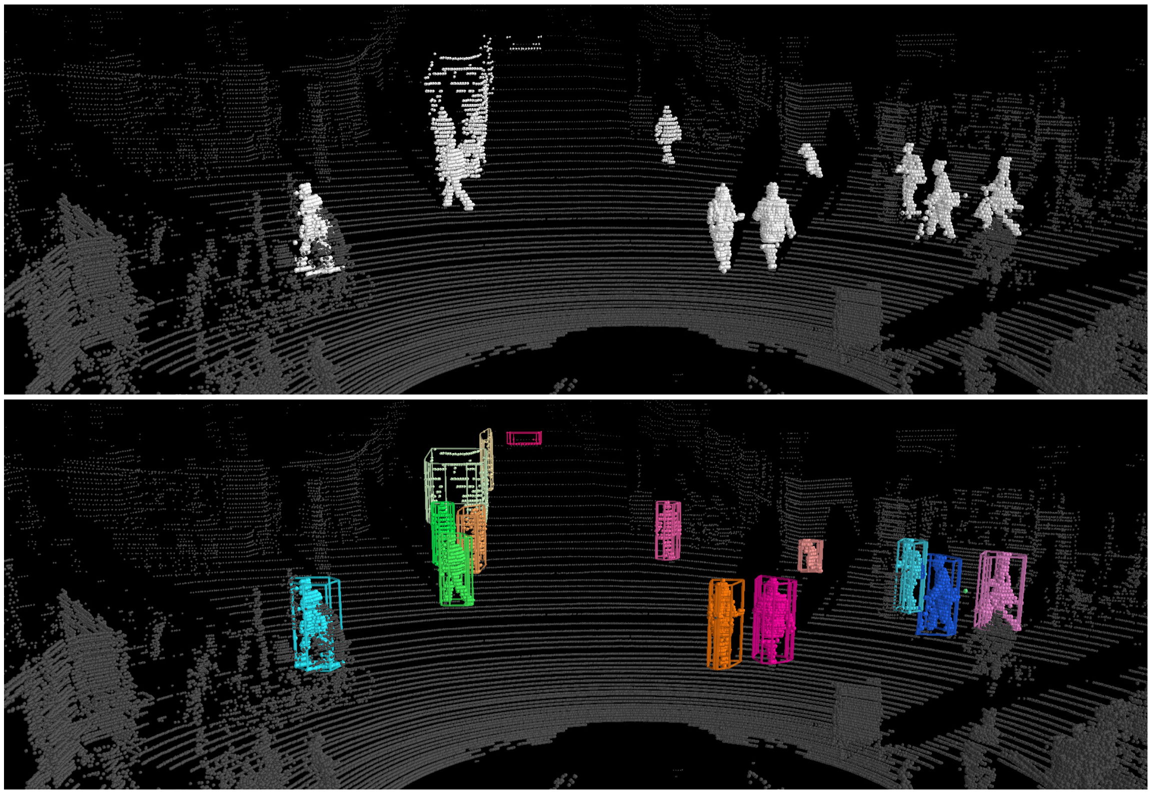


Figure 5.1: Our proposed algorithm takes a pre-processed LiDAR point cloud with background removed (top) and produces a class-agnostic instance-level segmentation over all foreground points (bottom). For visualization, we use a different color for each segment and plot an extruded polygon to show the spatial extent.

mance when processing out-of-sample data from an open world. Geometric graph-based approaches for segmentation tend not to require training and so are less-like to overfit, but also tend to be brittle.

Approach: Our approach searches over an exponentially-large space of candidate segmentations and returns one where individual segments score well according to a data-driven point-based model of "objectness" [2]. We demonstrate that one can repurpose existing closed-world point networks [140] for bottom-up perceptual grouping tasks that generalize to objects rarely seen during training.

Optimality: We prove that our approach produces *optimal* segmentations according to a specific definition. First, we restrict the search into a subset of segmentations that are consistent with a hierarchical grouping of a point cloud sweep. Such hierarchical groups can be readily produced with agglomerative clustering [208], HDBSCAN [119], or hierarchical graph-based algorithms [170].

Naive methods for producing a segmentation might apply a global threshold over the whole hierarchy. It turns out that one can produce an exponentially-large set of segmentations by applying different thresholds at different branches. We introduce efficient algorithms that search over this space of tree-consistent segmentations (Figure 5.2) and return the one that maximizes a global segmentation score that is computed by aggregating local objectness scores of individual segments.

Evaluation: We demonstrate empirical results on KITTI, a benchmark originally designed for closed-world object detection. Following past work, we repurpose it for open-world 3D segmentation [68]. We compare to existing bottom-up approaches [148] and state-of-the-art LiDAR-based object detectors after converting their output 3D bounding boxes to a point cloud segmentation. We demonstrate that our approaches outperform both baselines on less common classes.

5.2 Related work

Robust 3D object detection is crucial for downstream applications such as semantic understanding [59] and tracking [188]. Comparing to monocular 3D detection [98], we focus on LiDAR-based solutions in this paper.

LiDAR segmentation: Classic LiDAR segmentation algorithms use bottom-up grouping such as flood-filling [38], connected components [91], or density-based clustering [119]. Bottom-up strategies can also be applied on LiDAR sequences, allowing for motion as an additional cue [5, 67, 173]. Oftentimes such approaches are tuned for particular object categories such as cars. Our work differs in its use of static, single-frame cues that are not object-specific.

LiDAR object detection: There is an ever-increasing literature on data-driven object detection with LiDAR point clouds. Early approaches include fusion-based models that combine LiDAR and imagery [97], tracking-based detectors [114] and voxel-based classifiers [99, 193, 209]. We have seen approaches built upon raw point clouds such as PointRCNN [159]. Our approach is most related to Frustum PointNet [139] in the way we use pooled point cloud representation [140]. Our work differs in that we do not make use of camera input, and most notably, focus on *all* possible objects in an open world. Specifically, we compare to [97, 99, 160, 193] as a representative sample of the literature.

Perceptual grouping: Our graph-based approach is inspired by a long line of classic work on graph-theoretic perceptual grouping, dating back to normalized cuts [159], graph cuts [13], and spanning-tree approaches [42]. Such methods are typically used with hand-designed features, while we make use of data-driven techniques for learning a shape-based

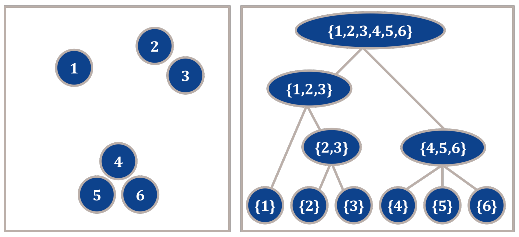


Figure 5.2: On the left, we visualize a set with 6 points. According to Bell number, one will find 203 unique segmentations (partitions). Most of these are arbitrary and do not respect local geometry, e.g. $\{\{1,2,5\},\{3,4,6\}\}$. On the right, we implement geometric constraints with a tree formed by hierarchical grouping. Every vertex cut of this tree is automatically a segmentation that respects local geometry encoded by the tree, e.g. $\{\{1\},\{2,3\},\{4,5,6\}\}$.

segment classifier.

Image segmentation: The idea of searching for an optimal image segmentation given a hierarchical image segmentation tree has been explored. [182] formulates neuron segmentation on electron microscopy images as a *maximum a posteriori* (MAP) labeling task on a tree-structured graph. It can be made equivalent to our search under certain conditions. [132] tackles the problem of class-agnostic instance segmentation in image space by exploiting visual appearance and motion. We discuss more in Section 5.3 and 5.4.2.

5.3 Approach

For 3D object point segmentation, the input is a 3D point cloud, which contains an *unknown* number of objects. The goal is to produce a point segmentation, in which every segment contains points from one and only one object.

Segmentation: A global *segmentation* P_X is a partition of a set of points $X = \{x_i\}_{i=1}^N$ into subsets of points, i.e. $P_X = \{C_i\}_{i=1}^M$, where M denotes the number of segments and $C_i \subset X$. We refer to each C_i as a local *segment*. Importantly, every point exists in one and only one segment, meaning $\cup_{i=1}^M C_i = X$ and $\forall i \neq j, C_i \cap C_j = \emptyset$.

Tree-consistent segmentations: Let us use S_X to denote the set of all possible global segmentations on X, i.e. all possible P_X. Without constraints, the size of S_X is exponential in N (i.e. the Bell number). In practice, we can reduce the number of candidates by enforcing geometric constraints. In this work, we implement the constraints by grouping all points hierarchically into a tree structure T_X. We will discuss how to build such a tree structure based on local geometric cues in Section 5.3.4. For now let us assume the tree is given.

Once we specify the tree, we can focus on a strictly smaller set of segmentations that respect local geometry. We denote such set as $S_{X,T}$ and call them *tree-consistent* segmentations. As a reference, the size of $S_{X,T}$ is still exponential in N, when T_X is a balanced binary tree[1]. We further illustrate the relationship between S_X and $S_{X,T}$ with an example in Figure 5.2. Any tree-consistent segmentation from $S_{X,T}$ corresponds to a vertex cut set of the tree T, i.e. a set of tree nodes, which satisfy the following constraints: (1) for each node in the vertex cut, its ancestor and itself cannot both be in the cut and (2) each leaf node must have itself or its ancestor in the cut. Such relationship allows us to design efficient tree searching algorithms, as we will see later.

Segment score: Before we discuss how to score a *global* segmentation, we first introduce how to score a *local* segment. Given a local segment $C \subset X$, we define a function $f(C;\theta) : C \mapsto [0,1]$ that predicts a given segment's "objectness", where θ represents the parameters. One can implement such a function with a PointNet++, where θ would represent weights of the PointNet++. We will discuss how to learn this function in Section 7.3.4. For now let us assume it is given.

Segmentation score: We now introduce how to score a *global* segmentation. Given a global segmentation $P_X = \{C_i\}_{i=1}^{M}$, we define its score $F(P_X;\theta) : P_X \mapsto [0,1]$ by aggregating over local objectness of its individual segments. Specifically, we introduce *worst-case* segmentation and *average-case* segmentation. Note that our objective can be made equivalent to [182] if we score a segmentation as the *sum* of its local segment scores. As we see in Section 5.4.2, this objective produces much larger oversegmentation error.

5.3.1 Worst-case segmentation

Worst-case segmentation scores a global segmentation as the *worst* objectness among its local segments:

$$F_{\min}(P_X;\theta) = \min_i f(C_i;\theta), i \in 1 \dots M \tag{5.1}$$

where $P_X \in S_{X,T}$, $P_X = \{C_i\}_{i=1}^{M}$, and $C_i \subset X$. We define $P^*_{X,min}$ as the *optimal worst-case segmentation* if

$$P^*_{X,\min} = \operatorname*{argmax}_{P_X \in S_{X,T}} F_{\min}(P_X;\theta) \tag{5.2}$$

It turns out the problem of finding optimal worst-case segmentation has optimal substructure (Theorem 1), allowing us to find the global optimum efficiently with dynamic programming (Algorithm **??**).

We briefly describe how the algorithm works. Given a set of points X and a tree T_X,

[1]One can derive recurrence on the number of segmentations between depth d+1 and d as $K_{d+1} = K_d^2 + 1$ with $K_1 = 2$. Since $K_d > 2^{2(d-1)}$, $K_d/N_d > 2^{d-2}$, where $N_d = 2^d$ represents the number of leaves, it suggests the number of segmentations *at least* outgrow the number of leaves exponentially.

OptMinSeg$(X,\ T_X)$ (Algorithm ??) produces an optimal worst-case segmentation $P^*_{X,\min}$ with score $F^*_{\min}(P^*_{X,\min}; \theta)$. For simplicity, we refer to a node in the tree by the set of points it is associated with. The algorithm starts from the root node X and chooses between a coarse segmentation ($\{X\}$) and a fine one. The fine segmentation will be the union of all X's children's optimal worst-case segmentation, which can be computed recursively. The algorithm would first traverse down to the leaf nodes, representing the finest segmentation. Then it will make its way up, during which it finalizes optimal segmentations for each intermediate node by making local coarse vs. fine decisions. Eventually, it returns to the root node and produces an optimal worst-case global segmentation.

Lemma 1 *Given pairs of non-empty sets that contain real numbers* $(X_1, Y_1), \ldots, (X_n, Y_n)$,

$$\forall i, \min_{x \in X_i} x \leq \min_{y \in Y_i} y \Rightarrow \min_{x \in \cup_i X_i} x \leq \min_{y \in \cup_i Y_i} y \tag{5.3}$$

Theorem 1 *Given C and T_C, Algorithm ?? finds the optimal segmentation* $P^*_{C,min} = \operatorname{argmax}_{P_C \in S_{C,T}} F_{min}(P_C; \theta)$.

Proof. Proof by structural induction.
Base: When $N_C = \emptyset$, meaning C corresponds to a leaf node in T_C, the algorithm returns $\{C\}$, which is the only segmentation in $S_{C,T}$ and obviously is optimal.
Induction: When $N_C \neq \emptyset$, we need to show that the algorithm will produce the optimal segmentation, i.e. P^*_C and F^*_C, if it has access to the optimal segmentation for each of C's child C_i, i.e. $P^*_{C_i}$ and $F^*_{C_i}$ (optimal substructure).

Let P_C be the segmentation that the algorithm produces for C and let F_C be its score. If P_C were not optimal, there must exist a different segmentation P''_C with score F'_C, s.t. $P'_C \neq P_C$ and $F'_C > F_C$. Moreover, P'_C is either a trivial segmentation, i.e. $P'_C = \{C\}$ or the union of segmentations over each of C's children nodes, i.e. $P'_C = \cup_i \{P'_{C_i}\}$.

First, P'_C is not a trivial segmentation. If we assume $P'_C = \{C\}$, we will have $F'_C = f(C; \theta)$. Since $P_C \neq P'_C$, the algorithm chooses P_C over $\{C\}$, therefore, $F_C > f(C; \theta)$. This clearly contradicts with $F'_C > F_C$.

Thus, P'_C has to be the union of segmentations over each of C's children node. According to the inductive hypothesis, the algorithm has the optimal segmentation over each of C's children node, meaning $\forall i, F'_{C_i} \leq F^*_{C_i}$ or concretely

$$\forall i, \min_{z \in P'_{C_i}} f(z; \theta) \leq \min_{z \in P^*_{C_i}} f(z; \theta) \tag{5.4}$$

Here, z represents an arbitrary local segment from a segmentation over C_i. By applying

Algorithm 1 Optimal worst-case segmentation

```
1: function OptMinSeg($C$, $T_C$)
   return a segmentation $P_C$ with a score of $F_C$
2:     $P_C \leftarrow \{C\}$
3:     $F_C \leftarrow f(C;\theta)$
4:     $N_C \leftarrow$ set of $C$'s children nodes in $T_C$
5:     if $N_C \neq \emptyset$ then
6:         for $C_i$ in $N_C$ do
7:             $T_{C_i} \leftarrow$ subtree of $T_C$ rooted at $C_i$
8:             $P_{C_i}, F_{C_i}$ = OptMinSeg($C_i$, $T_{C_i}$)
9:             if $F_{C_i} \leq F_C$ then return $P_C, F_C$
10:        if $\min_i F_{C_i} > F_C$ then
11:            $P_C \leftarrow \cup_i P_{C_i}$
12:            $F_C \leftarrow \min_i F_{C_i}$
       return $P_C, F_C$
```

Lemma 1, we have

$$\min_{z \in \cup_i P'_{C_i}} f(z;\theta) \leq \min_{z \in \cup_i P^*_{C_i}} f(z;\theta) \tag{5.5}$$

On one hand, $P'_C = \cup_i \{P'_{C_i}\}$ has a score of $F'_C = \min_{z \in \cup_i P'_{C_i}} f(z;\theta)$. On the other hand, the algorithm by design chooses the higher scoring one between $P_C = \{C\}$ with a score of $F_C = f(C;\theta)$ and $P_C = \cup_i P^*_{C_i}$ with a score of $F_C = \min_{z \in \cup_i P^*_{C_i}} f(z;\theta)$, ensuring that $F_C \geq \min_{z \in \cup_i P^*_{C_i}} f(z;\theta)$. With these and (5.5), we conclude $F_C \geq F'_C$, which contradicts the assumption $F'_C > F_C$. ■

Generality: Our analysis makes no assumptions about the objectness function $f(C;\theta)$ except the fact that it cannot be affected by the partitioning of other segments. In particular, this would allow objectness to depend on contextual arrangement of surrounding points outside C - e.g., $f(C, X;\theta)$.

Efficiency: Given points X and a tree T_X with N leaf nodes, Algorithm ?? guarantees to return the optimal worst-case segmentation after visiting every node in the tree. In practice, it might not visit all nodes. Instead, it skips the rest of sub-trees whenever one sub-tree exhibits lower score than the coarse segmentation (line 9 in Algorithm ??). The algorithm's complexity is linear in N despite the fact that the search space is exponential in N.

5.3.2 Average-case segmentation

Average-case segmentation scores a global segmentation as the *average* objectness among its local segments:

$$F_{\text{avg}}(P_X;\theta) = \frac{1}{M}\sum_{i=1}^{M} f(C_i;\theta) \tag{5.6}$$

where $P_X \in S_{X,T}$, $P_X = \{C_1, \ldots, C_M\}$, and $C_i \subset X$. We define $P^*_{X,\text{avg}}$ as an *optimal average-case segmentation* if

$$P^*_{X,\text{avg}} = \underset{P_X \in S_{X,T}}{\operatorname{argmax}} F_{\text{avg}}(P_X;\theta) \tag{5.7}$$

It turns out that the problem of finding the optimal *average-case* segmentation does not have optimal substructure, unlike *worst-case* segmentation, meaning a locally optimal partitioning might no longer be optimal when considering global partitioning. Formally speaking, Lemma 1 no longer holds once min is changed to avg.

Despite without optimal substructure, we apply a similar greedy searching algorithm. The main difference is how we aggregate local scores. Though greedily averaging local scores might lead to myopic decisions in certain situations (Figure 5.3), it performs well in practice (Section 7.4).

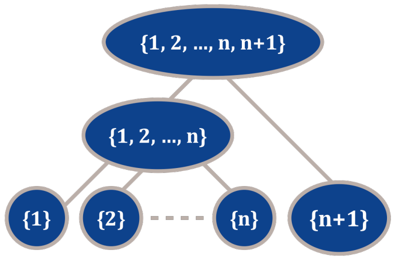
5.3.3 Learning the objectness function

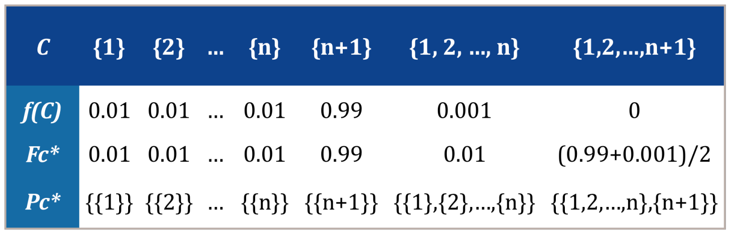
We have discussed segmentation algorithms under the assumption that we already have access to an objectness function $f(C;\theta)$, which predicts an objectness score for a given point cloud. We now introduce how to learn this function. Despite there has been a line of work that focuses on learning better representation, including Kd-networks [92], PointCNN [105], EdgeConv [187], PointConv [190], just to name a few, we choose a simple PointNet++ to parameterize such an objectness function as a proof of concept. Below, we talk about how to learn a PointNet++ model as a regressor to predict objectness score.

Ground truth objectness: First, we must define regression target, i.e. ground truth objectness, of a given segment C. Suppose we have ground truth segmentation $P^{gt} = \{C_1^{gt}, \ldots, C_L^{gt}\}$, where L is the number of ground truth segments. We can define C's target objectness as the largest point IoU between itself and any ground truth segment Eq. (5.8).

$$Objectness(C, P^{gt}) = \max_{l=1,\ldots,L} \frac{|C \cap C_l^{gt}|}{|C \cup C_l^{gt}|} \tag{5.8}$$

Such a definition of objectness is only reasonable if points are uniformly distributed in





C	{1}	{2}	...	{n}	{n+1}	{1, 2, ..., n}	{1,2,...,n+1}
f(C)	0.01	0.01	...	0.01	0.99	0.001	0
*Fc**	0.01	0.01	...	0.01	0.99	0.01	(0.99+0.001)/2
*Pc**	{{1}}	{{2}}	...	{{n}}	{{n+1}}	{{1},{2},...,{n}}	{{1,2,...,n},{n+1}}

Figure 5.3: We illustrate why average-case segmentation does not have optimal substructure. We plot a tree on the left and show local objectness scores on the right. In this case, the optimal average-case segmentation of the root node, i.e. {{1,2,. . . ,n},{n+1}} cannot be formed by the optimal average-case segmentations of its children nodes, i.e. {{1},{2},. . . ,{n}} and {{n+1}}.

space. In practice, 3D sensors (e.g. LiDAR) tend to produce denser points near the sensor. In consequence, the objectness will be heavily influenced by the partitioning of points closer to the sensor. For example, imagine two objects are segmented into one segment. Suppose one object has n_1 points and the other has n_2. If we use vanilla IoU as objectness, this segment would score $\frac{\max(n_1,n_2)}{n_1+n_2}$. When $n_1 \gg n_2$, the score could be really close to 1 despite it clearly introduces an under-segmentation error. To compensate such bias towards nearby objects, we propose a simple modification to IoU as in Eq. (5.9).

$$Objectness(C, P^{gt}) = \max_{l=1,\ldots,L} \frac{\sum_{x \in C \cap C_l^{gt}} x^T x}{\sum_{x \in C \cup C_l^{gt}} x^T x} \tag{5.9}$$

where $x^T x$ represents a point x's squared distance to sensor origin. Eq. (5.8) is a special case, where $x^T x$ is replaced with 1.

Implementation[2]**:** We train a PointNet++ w/ multi-scale grouping (MSG) [140] for learning the objectness function. Starting from the off-the-shelf architecture, we replaced the classifier with a regressor that produces a real-value given an input point cloud. We applied a sigmoid function to convert the regression output to numbers between [0,1]. Finally, we compute the mean-squared error between prediction and ground truth objectness and perform backprop. In terms of preprocessing, we follow [139] to make sure the input cloud is centered at origin and rotated based on the viewpoint. To facilitate batch processing, we follow the standard practice for PointNet++ and re-sample each segment to 1024 points.

[2]Code is available at: `https://www.github.com/peiyunh/3dseg`

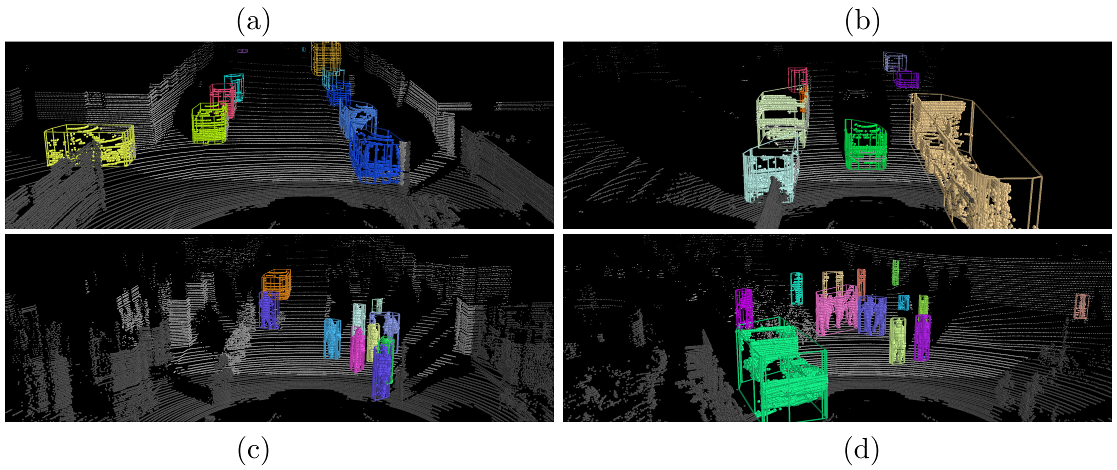


Figure 5.4: We visualize more qualitative results of the proposed algorithm Ours(avg) on KITTI. In (a), we show a common scenario where there are parked cars on both sides of the road. In (b), we show a rare scenario where there is an oversized tank truck in the right lane. In (c), we show a scenario where a group of pedestrians walking in front of the autonomous vehicle. In (d), we show a typical failure case where pedestrians walk closely side by side. For such cases, there is often no perfect solution within the search space generated by EC.

5.3.4 Building tree hierarchies

We have discussed segmentation algorithms under the assumption that we have access to a tree hierarchy. Now we introduce how to build such a tree hierarchy given a set of points X. One natural approach is agglomerative clustering. After we define a metric (i.e. pairwise distance between two points) and a linkage criteria (i.e. pairwise distance between two sets of points), we can start from $\{\{x_1\}, \ldots, \{x_N\}\}$ and keep merging the closest pair of point sets by taking the union over them, until all points are merged into one set. Such an approach produces a tree in a bottom-up fashion.

This approach tends to create tree hierarchies with very fine granularity, e.g. one node may differ from another with only one point of difference. As we have mentioned, our segmentation algorithms need to evaluate the objectness of every node in the tree. From an efficiency point of view, we would like to build a coarser tree whose leaf nodes are segments rather than individual points. Moreover, adjacent nodes should differ from each other much more.

Implementation: We build tree hierarchies by applying Euclidean Clustering [148] recursively in a top-down fashion with a list of decreasing ϵ. Since Euclidean Clustering finds connected components w.r.t. a distance threshold ϵ, we start with the largest ϵ that defines the most coarse connected components. Then, we apply Euclidean Clustering with a smaller

ϵ within each connected component. This produces a multiple-tree top-down hierarchy. In our experiments, we use $\epsilon \in \{2m, 1m, 0.5m, 0.25m\}$ to build tree hierarchies for both training and testing. During training, we extract segments out of tree hierarchies built with the same parameters to form our training set for learning the objectness function. During testing, we apply the same learned objectness function in both worst-case semgentation and average-case segmentation.

5.4 Experiments

For evaluation, we repurpose the KITTI object detection benchmark for point cloud segmentation following the setup in [68]. In our case, 3D objects do not physically overlap with one another. Therefore, we use ground truth 3D bounding boxes to produce ground truth segmentation. To do so, we first remove all points outside ground truth 3D bounding boxes (Figure 7.1). Then we treat points within one ground truth 3D bounding box as the ground truth segment for the object. On KITTI, there exist ground truth 3D bounding boxes that overlap with each other. We ignore such segments during evaluation, since it is not clear how to define the ground-truth for the points in such bounding boxes [68]. We follow [22] for splitting data into training and validation.

Evaluation protocol We follow evaluation metrics introduced by Held et al. [68], which consists of two errors, under-segmentation error and over-segmentation error. Given ground truth segmentation $P^{gt} = \{C_1^{gt}, \ldots, C_L^{gt}\}$, we compute under-segmentation error U and over-segmentation error O given an output segmentation $P = \{C_1, \ldots, C_M\}$ as:

$$U = \frac{1}{L} \sum_{l=1}^{L} \mathbf{1}\left(\frac{|C_{i^*} \cap C_l^{gt}|}{|C_{i^*}|} < \tau_U\right) \tag{5.10}$$

$$O = \frac{1}{L} \sum_{l=1}^{L} \mathbf{1}\left(\frac{|C_{i^*} \cap C_l^{gt}|}{|C_l^{gt}|} < \tau_O\right) \tag{5.11}$$

with

$$i^* = \operatorname*{argmax}_{i=1}^{M} |C_i \cap C_l^{gt}| \tag{5.12}$$

where $\mathbf{1}(\cdot)$ is an indicator function and τ_U, τ_O are both constant thresholds. We set $\tau_U = 2/3$ and $\tau_O = 1$ following [68]. We ignore objects with 0 points inside their 3D boxes (about 1%). For objects with overlapping bounding boxes (about 2.5%), we ignore points that fall into the overlapped region. Other than these, we compute segmentation errors over objects at all distance and also errors that focus on nearby objects (15m).

Table 5.1: Segmentation errors on KITTI Val. Left shows under-, over-segmentation, and total error. Right shows total error on a per-class basis.

Method	under		over		**total**		car		van		truck		pedestrian		person sitting		cyclist		*tram*		misc		**mean**	
	all	15m	all	15m	all	15m	all	15m	all	15m	all	15m	all	15m	all	15m	all	15m	all	15m	all	15m	all	15m
EC(2m)	24.89	46.21	5.24	0.43	30.1	46.6	24.4	37.3	18.2	21.1	**29.3**	**18.5**	63.1	75.2	79.2	78.5	28.7	53.0	94.9	55.0	31.0	36.2	46.1	46.9
EC(1m)	11.26	26.31	24.89	7.09	36.1	33.4	31.4	21.2	44.5	22.1	51.6	59.3	48.9	60.7	74.0	74.3	17.4	35.6	121.2	100.0	39.5	36.2	53.6	51.2
EC(0.5m)	5.54	12.89	63.70	48.32	69.2	61.2	74.6	67.9	81.0	73.2	79.3	92.6	39.0	47.6	63.0	62.5	25.5	16.1	121.2	100.0	64.9	43.3	68.6	62.9
EC(0.25m)	3.02	7.47	89.61	78.70	92.6	86.2	97.1	98.8	98.8	99.1	98.7	100.0	58.9	54.1	74.7	75.0	87.5	56.4	119.1	100.0	94.1	82.7	91.1	83.3
EC(all)*	*9.72*	*17.16*	*5.24*	*0.43*	*15.0*	*17.6*	*10.9*	*11.6*	*13.6*	*5.6*	*29.3*	*18.5*	*26.9*	*33.6*	*48.7*	*48.6*	*10.4*	*14.1*	*94.5*	*55.0*	*15.8*	*7.1*	*31.3*	*24.3*
AVOD	-	-	-	-	-	-	81.8	85.5	-	-	-	-	85.4	92.0	-	-	88.3	87.9	-	-	-	-	-	-
AVOD++	-	-	-	-	-	-	12.5	10.7	-	-	-	-	36.6	46.4	-	-	13.1	18.8	-	-	-	-	-	-
PointPillars++	-	-	-	-	-	-	22.4	22.7	-	-	-	-	58.6	63.6	-	-	44.8	36.2	-	-	-	-	-	-
PointRCNN++	-	-	-	-	-	-	7.6	5.2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SECOND++(4)	-	-	-	-	-	-	8.1	4.2	22.5	7.5	-	-	34.4	41.6	-	-	10.6	14.1	-	-	-	-	-	-
+ Ext. Range	-	-	-	-	-	-	**7.2**	4.5	18.4	7.0	-	-	34.2	41.8	-	-	10.0	**10.1**	-	-	-	-	-	-
+ BG Removal	-	-	-	-	-	-	9.9	**4.0**	25.9	8.9	-	-	33.2	39.1	-	-	11.9	15.4	-	-	-	-	-	-
+ Both	-	-	-	-	-	-	9.3	**4.0**	20.6	8.0	-	-	32.9	**38.7**	-	-	12.0	14.1	-	-	-	-	-	-
SECOND++(8)	4.07	7.42	11.79	9.25	15.9	16.7	8.1	5.1	23.2	8.0	43.2	51.9	33.2	39.5	70.8	70.8	9.6	13.4	119.5	100.0	28.7	18.9	42.0	38.5
+ Ext. Range	4.28	7.85	10.48	9.10	**14.8**	17.0	7.4	4.9	17.6	8.5	34.0	48.1	34.3	40.8	72.7	72.9	**9.5**	12.1	115.3	100.0	27.8	22.8	39.8	38.8
+ BG Removal	3.89	6.74	13.34	9.74	17.2	16.5	10.0	5.0	24.7	5.6	45.8	63.0	**32.8**	40.0	63.0	62.5	11.7	14.8	116.1	95.0	26.6	21.3	41.3	38.4
+ Both	3.84	6.66	12.28	9.66	16.1	**16.3**	9.4	4.8	20.1	6.1	38.2	63.0	33.2	40.1	63.0	62.5	11.5	13.4	112.7	95.0	23.4	21.3	38.9	38.3
Ours(min)	15.09	25.70	5.57	0.58	20.7	26.3	15.9	17.9	15.4	11.3	**29.3**	**18.5**	39.2	48.0	68.8	67.4	17.9	26.2	94.9	55.0	23.2	**15.0**	38.1	32.4
Ours(avg)	10.54	17.41	7.87	4.60	18.4	22.0	13.8	14.6	14.8	7.0	30.1	29.6	33.4	40.6	61.7	61.1	16.4	19.5	94.9	55.0	22.3	**15.0**	35.9	30.3
Ours(avg) w/																								
(2.7, 0.9, 0.3)m	13.41	19.65	6.16	5.13	19.6	24.8	15.8	17.1	13.9	13.6	23.6	25.9	36.2	43.3	63.6	63.2	18.3	26.2	72.0	20.0	20.2	19.7	33.0	28.6
(2.4, 1.2, 0.6, 0.3)m	11.26	16.65	5.94	5.69	17.2	22.3	12.9	15.0	11.5	7.5	27.2	29.6	34.7	41.2	**59.7**	**59.0**	15.7	19.5	80.9	25.0	**18.4**	19.7	32.6	27.1
(3.2, 1.6, 0.8, 0.4, 0.2)m	12.36	15.10	4.67	5.36	17.0	20.5	12.8	12.7	**10.7**	**4.2**	**21.5**	22.2	35.8	41.2	**59.7**	**59.0**	17.4	16.8	**67.8**	**10.0**	19.0	17.3	**30.6**	**22.9**

5.4.1 Baselines

Euclidean clustering: We use Euclidean clustering with 4 different distance threshold $\{2m, 1m, 0.5m, 0.25m\}$ to build trees of segments, which defines the space of possible segmentations for our approach. Therefore, we include them as baselines and see if a better solution can be found.

State-of-the-art 3D detectors: We compare our approach to AVOD [97], PointPillars [99], PointRCNN [160], and SECOND [193]. We follow the off-the-shelf training and testing setting as closely as possible. For AVOD, we re-train a car detector and a people detector (pedestrian and cyclist) with LiDAR as the only input following the official implementation[3]. For PointPillars, we re-train a detector that simultaneously detects cars and people (pedestrian and cyclist) following an author-endorsed implementation[4]. For PointRCNN, we evaluate the official pre-trained car model as there are no available models or training configurations for other classes within its official repository[5]. For SECOND, since it is our best performing baseline, besides re-training the off-the-shelf model, we also explore various ways to improve its performance.

By design, these detectors output class-specific bounding box detection. To produce class-agnostic segmentations, we ignore the class label and follow a greedy procedure: We

[3]https://github.com/kujason/avod
[4]https://github.com/traveller59/second.pytorch
[5]https://github.com/sshaoshuai/pointrcnn

start with the highest scoring bounding box and group all points within the box as one segment. We then remove those points and move onto the next highest scoring detection. We repeat until exhausting either detections or 3D points. In the end, we might still not have every point assigned to a segment. A simple fix is grouping leftover points as a new segment. We discuss a much better alternative approach below.

Detector++: A better approach to handling missed detection is to fall back to clustering. Specifically, we apply Euclidean Clustering (EC) with a fixed ϵ on all leftover points, producing a set of leftover segments. For each leftover segment, we check if it can merged into an existing detection segment, using the criteria of whether the smallest pairwise distance between two segments is smaller than the threshold ϵ. If so, we merge the leftover segment into the detection segment. We refer to such baselines as Detector++ (e.g. AVOD++ etc.).

SECOND++: To ensure an apples-to-apples comparison, we re-train and re-evaluate the best baseline, i.e. SECOND, with background removal. These baselines are marked with "+ BG Removal". In addition, we discover that, by extending SECOND's detection range from 50m to 80m, we significantly improve SECOND's performance. The affected baselines are marked with "+ Ext. Range". Finally, we re-train and re-evaluate SECOND on all 8 classes. The new baselines are labeled as "SECOND++(8)". In contrast, off-the-shelf SECOND baselines are labeled as "SECOND++(4)" as they are trained on 4 classes (car, pedestrian, cyclist, and van).

5.4.2 Results

We first present qualitative examples of our approach segmenting rare objects on KITTI Val, as shown in Figure 5.4. For quantitative evaluation, we present both per-class and overall segmentation errors in Table 5.1.

Ours(min) vs. Ours(avg): We label the optimal worst-case segmentation as Ours(min) and the average-case segmentation as Ours(avg). Ours(avg) consistently outperforms Ours(min) in terms of the total error. Ours(min) produces a much lower over-segmentation error but a much higher under-segmentation error, suggesting it makes more mistakes of grouping different objects into one segment and less mistakes of splitting points from one single object into multiple segments. The cause of such behavior might be due to the risk-averse objective of optimal worst-case segmentation. However, current evaluation does not emphasize the worst-case performance, instead, it measures the average performance over all objects. We observe that if we evaluate the worst-case objectness (Section 5.4.3), Ours(min) does outperform both Ours(avg) and AVOD++.

Ours vs. Euclidean Clustering: We label Euclidean Clustering as "EC(ϵ)", where ϵ represents the distance threshold (meter). All together, they define a segment hierarchy. We construct a pool of segments that contains every node (segment) in the hierarchy and call this "EC(all)*". This serves as a *unreachable* upper-bound, since segments from such a pool overlap with each other, which violates the non-disjoint constraint of a valid partition. Nonetheless, it shows that there gap between our proposed method and the upper bound is relatively small (3-4%), suggesting plenty of room left for improvement in creating better hierarchies.

Detector++ vs. Detector: We focus on AVOD to demonstrate the improvement of Detector++ over Detector. AVOD produces much larger oversegmentation errors, likely due to imprecisely localized 3D bounding boxes. For example, when a 3D bounding box is predicted smaller than it should be, the resultant segment might miss points on the edge, leading to oversegmentation. AVOD++ is designed to fix this issue and dramatically improves the oversegmentation error. The undersegmentation errors also improves significantly from AVOD to AVOD++, likely due to successfully segmenting objects that are completely missed by detections.

Ours vs. Detector++: SECOND++ performs the best among all Detector++ baselines and also achieves the lowest overall total error among all methods. However, if we break down total segmentation errors on a per-class basis, our approaches perform much better than SECOND++. Such difference is due to a skewed data distribution. For example, 68% objects are labeled as car while only 3% are labeled as misc. SECOND++ performs better on common classes such as car and ours perform better on rare ones such as misc.

Runtime analysis: Our algorithm requires running PointNet++ on every candidate segment in order to compute its objectness. In practice, one frame from KITTI Val, which contains $68(\sigma = 42)$ segments on average, takes about $0.19s(\sigma = 0.06s)$ to process on a single GTX 1080.

5.4.3 Additional evaluation protocols

Class-agnostic instance segmentation: The evaluation protocol we adopt comes from the robotics community [67]. It differs from the standard evaluation in computer vision, i.e. per-voxel instance segmentation in ScanNet [31]. One key difference is that 3D instance segmentation does not require the output segmentation to be a valid partition. Instead, it treats the task as retrieval and evaluates the tradeoff between precision and recall. Here we take a similar approach as ScanNet, but modify the evaluation protocol to be class-agnostic and per-point instead of per-voxel.

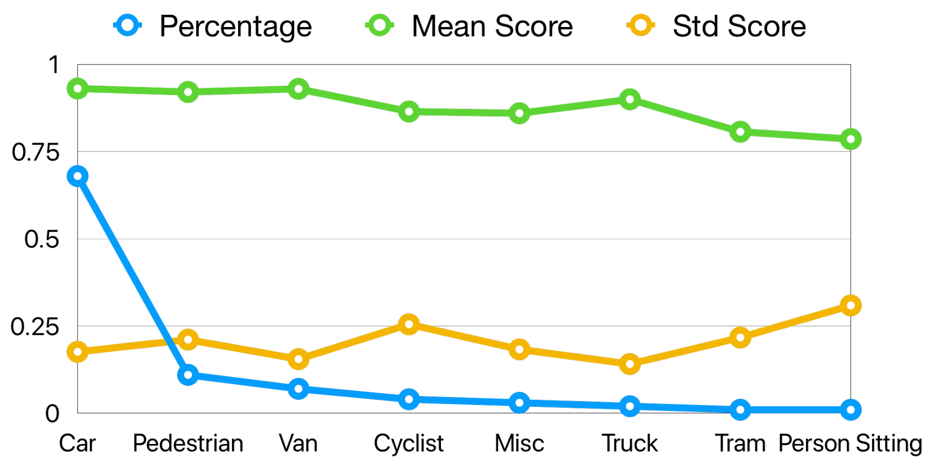


Figure 5.5: How the learned objectness model generalizes in the tail.

As we can see in Table 5.2, the observations are consistent with what we see in Table 5.1: SECOND++(8) with both modifications outperforms our segmentation approach on common classes such as *car*, but falls short on rarer classes (such as person sitting and tram) by a large margin. Overall, the best SECOND approach outperforms the best variant of our approach by 1.6% in mAP.

How objectness generalizes: To evaluate how well our learned objectness model generalizes, we apply it onto ground truth segments from the validation set. In Figure 5.5, we plot the average objectness score for each class and the standard deviation. We also show the percentage of objects for each class within the training set. As the number of training data decreases dramatically, the average score tends to drops slightly and the variance tends to rise slightly.

Worst-case evaluation: In Table 5.1 and 5.2, we see Ours(avg) outperforms Ours(min) despite the latter is provably optimal. We have briefly discussed the reason: current protocols do not evaluate worst-case performance. Here, we score the worst IoU between a set of local segments and the ground truths, as Eq. (5.13) shows, where $\{P_1 \dots P_N\}$ and $\{P_1^{gt} \dots P_N^{gt}\}$ represents predicted and ground truth segmentation in each of the N frames. We found Ours(min) scores a mean-worst IoU of 72.2%, 4.2% higher than Ours(avg).

$$\text{score} = \sum_{i=1}^{N} \frac{1}{N} \min_{C \in P_i} \max_{C^{gt} \in P_i^{gt}} \frac{|C \cap C^{gt}|}{|C \cup C^{gt}|} \tag{5.13}$$

Table 5.2: Instance segmentation AP[@.5:.95:.05] on KITTI Val.

	car	van	trk	ped	psit	cyc	tram	misc	**mean**
AVOD	64.4	-	-	31.1	-	15.5	-	-	-
AVOD++	91.6	-	-	51.6	-	41.6	-	-	-
PointPillars++	91.4	-	-	55.2	-	55.9	-	-	-
PointRCNN++	95.2	-	-	-	-	-	-	-	-
SECOND++(4)	95.1	68.9	-	68.6	-	65.9	-	-	-
+ Ext. Range	95.8	75.4	-	70.2	-	68.1	-	-	-
+ BG Removal	95.3	74.0	-	77.5	-	71.8	-	-	-
+ Both	**96.0**	**82.0**	-	**78.1**	-	**72.9**	-	-	-
SECOND++(8)	95.3	70.3	30.0	71.8	2.6	69.6	10.2	33.9	48.0
+ Ext. Range	95.9	78.3	**63.4**	71.0	2.9	71.9	13.4	39.6	54.5
+ BG Removal	95.1	73.5	30.3	76.2	9.0	71.4	13.1	47.3	52.0
+ Both	**96.0**	81.3	61.7	76.5	8.6	72.4	16.4	**55.4**	**58.5**
Ours(min)	86.0	80.4	61.6	62.3	12.9	66.3	**21.9**	53.0	55.6
Ours(avg)	89.8	81.1	58.6	69.2	**14.0**	68.2	19.8	51.0	56.5
Ours(avg) w/									
(2.7, 0.9, 0.3)m	87.5	78.6	57.6	66.7	**14.0**	66.9	20.8	49.7	55.2
(2.4, 1.2, 0.6, 0.3)m	89.6	81.9	59.4	67.9	13.7	69.2	21.5	52.1	56.9
(3.2, 1.6, 0.8, 0.4, 0.2)m	89.0	79.0	56.3	67.6	13.2	67.2	18.7	49.0	55.0

5.4.4 Additional diagnostics

Sensitivity analysis: Our objectness function is learned on segments from a EC hierarchy generated with 4 distance thresholds {2m, 1m, 0.5m, 0.25m}. To analyze how robust our algorithm is to change of hyper-parameters, we test the learned objectness function on different hierarchies. In Table 5.1 and 5.2, we find that having a deeper hierarchy significantly reduces segmentation errors. Comparing to hard-thresholded segmentation errors, there are only slight changes in multi-threshold instance segmentation mAP.

Weighted vs. vanilla IoU: Here, we empirically compare weighted IoU and vanilla IoU in terms of defining the training target for our objectness model. As we see in Table 5.3, for both worst-case and average-case segmentation, the objectness model trained with weighted IoU perform slightly better than the one trained with vanilla IoU. Note "Ours(min) - vanilla" and "Ours(avg) - vanilla" share the same objectness model.

Table 5.3: Segmentation errors on KITTI Val.

Method	Under (%)		Over (%)		Total (%)	
	all	15m	all	15m	all	15m
Ours(min) - vanilla	13.91	22.40	5.58	0.60	19.5	23.0
Ours(min) - weighted	13.13	21.42	5.65	0.60	**18.8**	**22.0**
Ours(avg) - vanilla	10.30	15.44	7.11	3.13	17.4	18.6
Ours(avg) - weighted	8.64	12.75	7.89	4.73	**16.5**	**17.5**

5.5 Conclusion

We present an approach for class-agnostic point cloud segmentation. The approach efficiently searches over an exponentially large space of candidate segmentations and return one where individual segments score well according to a data-driven point-based model of “objectness”. We prove that our algorithm is guaranteed to achieve optimality to a specific definition. On KITTI, we demonstrate our approach significantly outperforms past bottom-up approaches and top-down object-based algorithms for segmenting point clouds.

Part II

Scalability

Chapter 6

Learning with Active and Partial Feedback

6.1 Introduction

Given a large set of unlabeled images, and a budget to collect annotations, how can we learn an accurate image classifier most economically? Active Learning (AL) seeks to increase data efficiency by strategically choosing which examples to annotate. Typically, AL treats the labeling process as atomic: every annotation costs the same and produces a correct label. However, large-scale multi-class annotation is seldom atomic; we can't simply ask a crowd-worker to select one among 1000 classes if they aren't familiar with our ontology. Instead, annotation pipelines typically solicit feedback through simpler mechanisms such as yes/no questions. For example, to construct the 1000-class ImageNet dataset, researchers first filtered candidates for each class via Google Image Search, then asking crowd-workers questions like "Is there a Burmese cat in this image?" [32]. For tasks where the Google trick won't work, we might exploit class hierarchies to drill down to the *exact* label. Costs scale with the number of questions asked. Thus, real-world annotation costs can vary per example [156].

We propose *Active Learning with Partial Feedback* (ALPF), asking, *can we cut costs by actively choosing both which examples to annotate, and which questions to ask?* Say that for a new image, our current classifier places 99% of the predicted probability mass on various dog breeds. Why start at the top of the tree – *"is this an artificial object?"* – when we can cut costs by jumping straight to dog breeds (Figure 6.1)?

ALPF proceeds as follows: In addition to the class labels, the learner possesses a

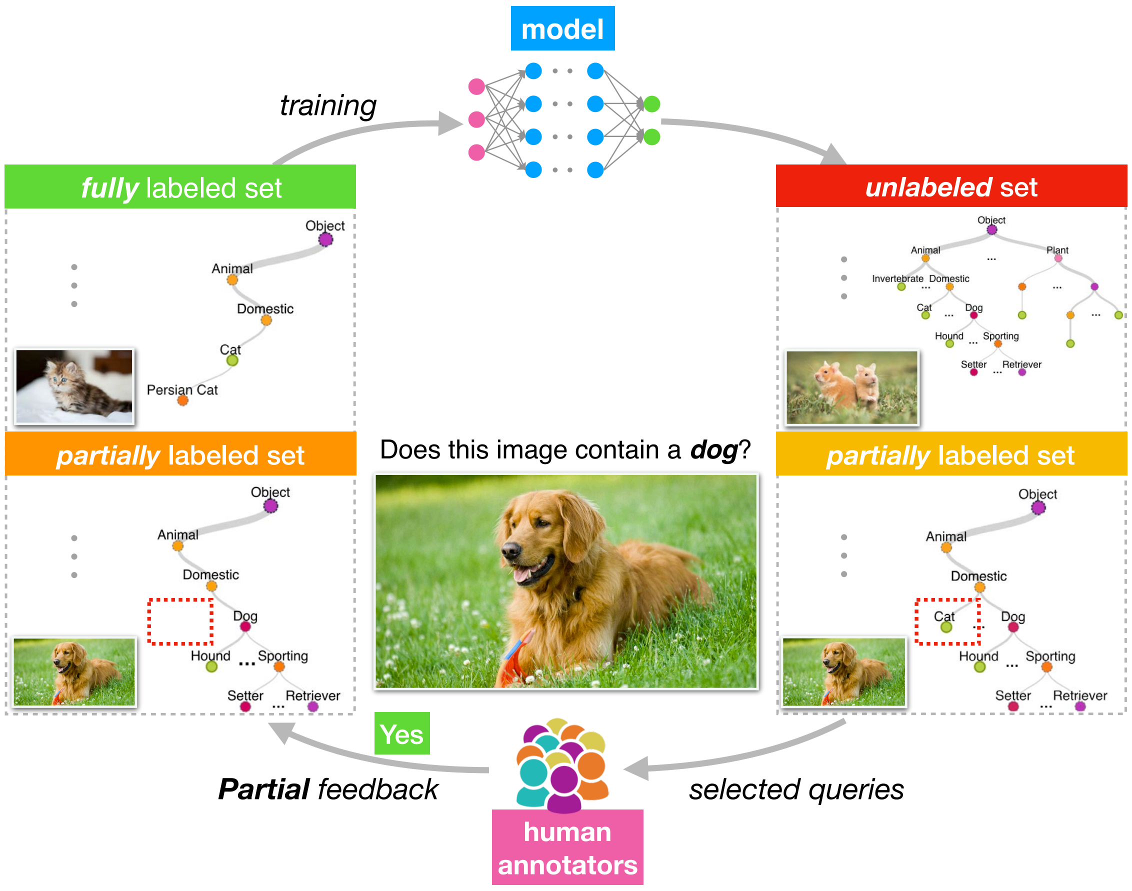


Figure 6.1: Workflow for an ALPF learner.

pre-defined collection of *composite classes*, e.g. *dog* ⊃ *bulldog*, *mastiff*, ... At each round, the learner selects an (example, class) pair. The annotator responds with binary feedback, leaving the learner with a *partial label*. If only the *atomic* class label remains, the learner has obtained an *exact label*. For simplicity, we focus on hierarchically-organized collections—trees with atomic classes as leaves and composite classes as internal nodes.

For this to work, we need a hierarchy of concepts *familiar* to the annotator. Imagine asking an annotator *"is this a **foo**?"* where *foo* represents a category comprised of 500 *random* ImageNet classes. Determining class membership would be onerous for the same reason that providing an exact label is: It requires the annotator be familiar with an enormous list of seemingly-unrelated options before answering. On the other hand, answering *"is this an animal?"* is easy despite *animal* being an extremely coarse-grained category —because most people already know what an animal is.

We use *active questions* in a few ways. To start, in the simplest setup, we can select samples at random but then once each sample is selected, choose questions actively until finding the label:

ML: "Is it a dog?"	Human: Yes!
ML: "Is it a poodle?"	Human: No!
ML: "Is it a hound?"	Human: Yes!
ML: "Is it a Rhodesian ?"	Human: No!
ML: "Is it a Dachsund?"	**Human: Yes!**

In ALPF, we go one step further. Since our goal is to produce accurate classifiers on tight budget, should we necessarily label each example to completion? After each question, ALPF learners have the option of choosing a *different example* for the next binary query. Efficient learning under ALPF requires (i) good strategies for choosing (example, class) pairs, and (ii) techniques for learning from the partially-labeled data that results when labeling examples to completion isn't required.

We first demonstrate an effective scheme for learning from partial labels. The predictive distribution is parameterized by a softmax over all classes. On a per-example basis, we convert the multiclass problem to a binary classification problem, where the two classes correspond to the subsets of *potential* and *eliminated* classes. We determine the total probability assigned to *potential* classes by summing over their softmax probabilities. For active learning with partial feedback, we introduce several acquisition functions for soliciting partial labels, selecting questions among all (example, class) pairs. One natural method, expected information gain (EIG) generalizes the classic maximum entropy heuristic to the ALPF setting. Our two other heuristics, EDC and ERC, select based on the number of labels that we *expect* to see *eliminated from* and *remaining in* a given partial label, respectively.

We evaluate ALPF learners on CIFAR10, CIFAR100, and Tiny ImageNet datasets. In all cases, we use WordNet to impose a hierarchy on our labels. Each of our experiments simulates rounds of active learning, starting with a small amount of i.i.d. data to warmstart the models, and proceeding until all examples are exactly labeled. We compare models by their test-set accuracy after various amounts of annotation. Experiments show that ERC sampling performs best. On *TinyImageNet*, with a budget of 250k binary questions, ALPF improves in accuracy by 26% (relative) and 8.1% (absolute) over the i.i.d. baseline. Additionally, ERC & EDC fully annotate the dataset with just $491k$ and $484k$ examples binary questions, respectively (vs 827k), a 42% reduction in annotation cost. Surprisingly, we observe that taking disparate annotation costs into account may alter the conventional wisdom that active learners should solicit labels for *hard* examples. In ALPF, *easy* examples might yield less information, but are cheaper to annotate.

6.2 Active Learning with Partial Feedback

By $\boldsymbol{x} \in \mathcal{R}^d$ and $y \in \mathcal{Y}$ for $\mathcal{Y} = \{\{1\}, ..., \{k\}\}$, we denote feature vectors and labels. Here d is the feature dimension and k is the number of *atomic* classes. By *atomic* class, we mean that they are indivisible. As in conventional AL, the agent starts off with an unlabeled training set $\mathcal{D} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$.

Composite classes We also consider a pre-specified collection of composite classes $\mathcal{C} = \{c_1, ..., c_m\}$, where each composite class $c_i \subset \{1, ..., k\}$ is a subset of labels such that $|c_i| \geq 1$. Note that $\mathcal{C}$ includes both the atomic and composite classes. In this paper's empirical section, we generate composite classes by imposing an existing lexical hierarchy on the class labels [121].

Partial labels For an example i, we use *partial label* to describe any element $\tilde{y}_i \subset \{1, ..., k\}$ such that $\tilde{y}_i \supset y_i$. We call $\tilde{y}_i$ a *partial label* because it may rule out some classes, but doesn't fully indicate underlying atomic class. For example, *dog* = $\{$*akita, beagle, bulldog, ...*$\}$ is a valid partial label when the true label is $\{$*bulldog*$\}$. An ALPF learner eliminates classes, obtaining successively smaller partial labels, until only one (the *exact label*) remains. To simplify notation, in this paper, by an example's *partial label*, we refer to the smallest partial label available based on the already-eliminated classes. At any step t and for any example i, we use $\tilde{y}_i^{(t)}$ to denote the current partial label. The initial partial label for every example is $\tilde{y}^0 = \{1, ..., k\}$ An *exact label* is achieved when the partial label $\tilde{y}_i = y_i$.

Partial Feedback The set of possible questions $\mathcal{Q} = \mathcal{X} \times \mathcal{C}$ includes all pairs of examples and composite classes. An ALPF learner interacts with annotators by choosing questions $q \in \mathcal{Q}$. Informally, we pick a question $q = (\boldsymbol{x}_i, c_j)$ and ask the annotator, *does* $\boldsymbol{x}_i$ *contain a* c_j*?* If the queried example's label belongs to the queried composite class $(y_i \subset c_j)$, the answer is 1, else 0.

Let α_q denote the binary answer to question $q \in \mathcal{Q}$. Based on the partial feedback, we can compute the new partial label $\tilde{y}^{(t+1)}$ according to Eq. (6.1),

$$\tilde{y}^{(t+1)} = \begin{cases} \tilde{y}^{(t)} \setminus c & \text{if } \alpha = 0 \\ \tilde{y}^{(t)} \setminus \overline{c} & \text{if } \alpha = 1 \end{cases} \tag{6.1}$$

Note that here $\tilde{y}^{(t)}$ and c are sets, α is a bit, $\overline{c}$ is a set complement, and that $\tilde{y}^{(t)} \setminus \overline{c}$ and $\tilde{y}^{(t)} \setminus c$ are set subtractions to eliminate classes from the partial label based on the answer.

Learning Process The learning process is simple: At each round t, the learner selects a pair $(\boldsymbol{x}, c)$ for labeling. Note that a rational agent will never select either (i) an example for which the exact label is known, or (ii) a pair $(\boldsymbol{x}, c)$ for which the answer is already known, e.g., if $c \supset \tilde{y}^{(t)}$ or $c \cap \tilde{y}^{(t)} = \emptyset$. After receiving binary feedback, the agent updates the corresponding partial label $\tilde{y}^{(t)} \rightarrow \tilde{y}^{(t+1)}$, using Equation 6.1. The agent then re-estimates its model, using all available non-trivial partial labels and selects another question q. In batch-mode, the ALPF learner re-estimates its model once per T queries which is necessary when training is expensive (e.g. deep learning). We summarize the workflow of a ALPF learner in Algorithm **??**.

Objectives We state two goals for ALPF learners. First, we want to learn predictors with low error (on exactly labeled i.i.d. holdout data), given a fixed annotation budget. Second, we want to fully annotate datasets at the lowest cost. In our experiments (Section 6.3), a ALPF strategy dominates on both tasks.

6.2.1 Learning from partial labels

We now address the task of learning a multiclass classifier from partial labels, a fundamental requirement of ALPF, regardless of the choice of sampling strategy. At time t, our model $\hat{y}(y, \boldsymbol{x}, \theta^{(t)})$ parameterised by parameters $\theta^{(t)}$ estimates the conditional probability of an atomic class y. For simplicity, when the context is clear, we will use $\hat{\boldsymbol{y}}$ to designate the full vector of predicted probabilities over all classes. The probability assigned to a partial label $\tilde{y}$ can be expressed by marginalizing over the atomic classes that it contains: $\hat{p}(\tilde{y}^{(t)}, \boldsymbol{x}, \theta^{(t)}) = \sum_{y \in \tilde{y}^{(t)}} \hat{y}(y, \boldsymbol{x}, \theta^{(t)})$. We optimize our model by minimizing the log loss:

$$\mathcal{L}(\theta^{(t)}) = -\frac{1}{n} \sum_{i=1}^{n} \log \left[\hat{p}(\tilde{y}_i^{(t)}, \boldsymbol{x}_i, \theta^{(t)}) \right] \tag{6.2}$$

Note that when every example is exactly labeled, our loss function simplifies to the standard cross entropy loss often used for multi-class classification. Also note that when every partial label contains the full set of classes, all partial labels have probability 1 and the update is a no-op. Finally, if the partial label indicates a composite class such as *dog*, and the predictive probability mass is exclusively allocated among various breeds of dog, our loss will be 0. Models are only updated when their predictions disagree (to some degree) with the current partial label.

Algorithm 1 Active Learning with Partial Feedback

```
Input: $\mathbf{X} \leftarrow (\mathbf{x}_1, \ldots, \mathbf{x}_N)$, $\mathbf{Q} \leftarrow (\mathbf{q}_1, \ldots, \mathbf{q}_M)$,
$K, T$.
Input: $\mathcal{D} \leftarrow [\boldsymbol{x}_i]_{i=1}^N$, $\mathcal{C} \leftarrow [c_j]_{j=1}^M$, $k$, $T$
Initialize: $\tilde{y}_i^{(0)} \leftarrow \{1, \ldots, k\}$, $\theta \leftarrow \theta^{(0)}$, $t \leftarrow 0$
repeat
    Score every $(\boldsymbol{x}_i, c_j)$ with $\theta$
    repeat
        Select $(\boldsymbol{x}_{i^*}, c_{j^*})$ with the best score
        Query $c_{j^*}$ on data $\boldsymbol{x}_{i^*}$
        Receive feedback $\alpha$
        Update $\tilde{y}_{i^*}^{(t+1)}$ according to $\alpha$
        $t \leftarrow t + 1$
    until ($t$ mod $T = 0$) or ($\forall i, |\tilde{y}_i^{(t)}| = 1$)
    $\theta \leftarrow \arg\min_\theta \mathcal{L}(\theta)$
until $\forall i, |\tilde{y}_i^{(t)}| = 1$ or $t$ exhausts budget
```

6.2.2 Sampling strategies

Expected Information Gain (EIG): Per classic uncertainty sampling, we can quantify a classifer's uncertainty via the entropy of the predictive distribution. In AL, each query returns an exact label, and thus the post-query entropy is always 0. In our case, each answer to the query yields a different partial label. We use the notation $\hat{\boldsymbol{y}}_0$, and $\hat{\boldsymbol{y}}_1$ to denote consequent predictive distributions for each answer (no or yes). We generalize maximum entropy to ALPF by selecting questions with greatest *expected reduction in entropy.*

$$EIG_{(\boldsymbol{x},c)} = S(\hat{\boldsymbol{y}}) - [\hat{p}(c, \boldsymbol{x}, \theta)S(\hat{\boldsymbol{y}}_1) + (1 - \hat{p}(c, \boldsymbol{x}, \theta))S(\hat{\boldsymbol{y}}_0)] \tag{6.3}$$

where $S(\cdot)$ is the entropy function. It's easy to prove that EIG is maximized when $\hat{p}(c, \boldsymbol{x}, \theta) = 0.5$.

Table 6.1: Learning from partial labels on Tiny ImageNet. These results demonstrate the usefulness of our training scheme absent the additional complications due to ALPF. In each row, γ% of examples are assigned labels at the *atomic class* (Level 0). Levels 1, 2, and 4 denote progressively coarser composite labels tracing through the WordNet hierarchy.

$\gamma(\%)$	γ	$(1-\gamma)$		
	Level 0	Level 1	Level 2	Level 4
20	0.285	**+0.113**	+0.086	+0.025
40	0.351	+0.079	+0.056	+0.016
60	0.391	+0.051	+0.036	+0.018
80	0.432	+0.015	+0.017	-0.009
100	0.441	-	-	-

Expected Remaining Classes (ERC): Next, we propose ERC, a heuristic that suggests arriving as quickly as possible at exactly-labeled examples. At each round, ERC selects those examples for which the expected number of remaining classes is fewest:

$$ERC_{(\boldsymbol{x},c)} = \hat{p}(c, \boldsymbol{x}, \theta)||\hat{\boldsymbol{y}}_1||_0 + (1 - \hat{p}(c, \boldsymbol{x}, \theta))||\hat{\boldsymbol{y}}_0||_0, \tag{6.4}$$

where $||\hat{\boldsymbol{y}}_\alpha||$ is the size of the partial label following given answer α. ERC is minimized when the result of the feedback will produce an exact label with probability 1. For a given example $\boldsymbol{x}_i$, if $||\hat{\boldsymbol{y}}_i||_0 = 2$ containing only the potential classes (e.g.) *dog* and *cat*, then with certainty, ERC will produce an exact label by querying the class $\{dog\}$ (or equivalently $\{cat\}$). This heuristic is inspired by [27], which shows that the partial classification loss (what we optimize with partial labels) is an upper bound of the true classification loss (as if true labels are available) with a linear factor of $\frac{1}{1-\varepsilon}$, where ε is ambiguity degree and $\varepsilon \propto |\tilde{y}|$. By selecting $q \in \mathcal{Q}$ that leads to the smallest $|\tilde{y}|$, we can tighten the bound to make optimization with partial labels more effective.

Expected Decrease in Classes (EDC): More in keeping with the traditional goal of minimizing uncertainty, we might choose EDC, the sampling strategy which we expect to result in the greatest reduction in the number of potential classes. We can express EDC as the difference between the number of potential labels (known) and the expected number of potential labels remaining: $EDC_{(\boldsymbol{x},c)} = |\tilde{y}^{(t)}| - ERC_{(\boldsymbol{x},c)}$.

6.3 Experiments

We evaluate ALPF algorithms on the CIFAR10, CIFAR100, and Tiny ImageNet datasets, with training sets of 50k, 50k, and 100k examples, and 10, 100, and 200 classes respectively. After imposing the Wordnet hierarchy on the label names, the size of the set of possible binary questions $|\mathcal{C}|$ for each dataset are 27, 261, and 304, respectively. The number of binary questions between re-trainings are 5k, 15k, and 30k, respectively. By default, we warm-start each learner with the same 5% of training examples selected i.i.d. and exactly labeled. Warm-starting has proven essential in other papers combining deep and active learning [158]. Our own analysis (Section 6.3.3) confirms the importance of warm-starting although the affect appears variable across acquisition strategies.

Model For each experiment, we adopt the widely-popular ResNet-18 architecture [65]. Because we are focused on active learning and thus seek fundamental understanding of this new problem formulation, we do not complicate the picture with any fine-tuning techniques. Note that some leaderboard scores circulating on the Internet appear to have far superior numbers. This owes to pre-training on the full ImageNet dataset (from which Tiny-ImageNet was subsampled and downsampled), constituting a target leak.

We initialize weights with the *Xavier* technique [55] and minimize our loss using the Adam [90] optimizer, finding that it outperforms SGD significantly when learning from partial labels. We use the same learning rate of 0.001 for all experiments, first-order momentum decay (β_1) of 0.9, and second-order momentum decay (β_2) of 0.999. Finally, we train with mini-batches of 200 examples and perform standard data augmentation techniques including random cropping, resizing, and mirror-flipping. We implement all models in MXNet and have posted our code publicly[1].

Re-training Ideally, we might update models after each query, but this is too costly. Instead, following [158] and others, we alternately query labels and update our models in rounds. We warm-start all experiments with 5% labeled data and iterate until every example is exactly labeled. At each round, we re-train our classifier from scratch with random initialization. While we could initialize the new classifier with the previous best one (as in [158]), preliminary experiments showed that this faster convergence comes at the cost of worse performance, perhaps owing to severe over-fitting to labels acquired early in training. In all experiments, for simplicity, we terminate the optimization after 75 epochs. Since $30k$ questions per re-training (for TinyImagenet) seems infrequent, we compared against 10x more frequent re-training More frequent training conferred no benefit (Appendix **??**).

[1]Our implementations of ALPF learners are available at: `https://github.com/peiyunh/alpf`

6.3.1 Learning from partial labels

Since the success of ALPF depends in part on learning from partial labels, we first demonstrate the efficacy of learning from partial labels with our loss function when the partial labels are given a priori. In these experiments we simulate a partially labeled dataset and show that the learner achieves significantly better accuracy when learning from partial labels than if it excluded the partial labels and focused only on exactly annotated examples. Using our WordNet-derived hierarchy, we conduct experiments with partial labels at different levels of granularity. Using partial labels from one level above the leaf, *German shepherd* becomes *dog*. Going up two levels, it becomes *animal*.

We first train a standard multi-class classifier with γ (%) *exactly labeled* training data and then another classifier with the remaining $(1-\gamma)\%$ *partially labeled* at a different granularity (level of hierarchy). We compare the classifier performance on holdout data both *with* and *without* adding *partial labels* in Table 6.1. We make two key observations: (i) additional coarse-grained partial labels improve model accuracy (ii) as expected, the improvement diminishes as partial label gets coarser. These observations suggest we can learn effectively given a mix of exact and partial labels.

6.3.2 Sampling strategies

Baseline This learner samples examples at random. Once an example is sampled, the learner applies top-down binary splitting—choosing the question that most evenly splits the probability mass, see Related Work for details— with a uniform prior over the classes until that example is exactly labeled.

AL To disentangle the effect of active sampling of questions and samples, we compare to conventional AL approaches selecting examples with uncertainty sampling but selecting questions as baseline.

AQ *Active questions* learners, choose examples at random but use partial feedback strategies to efficiently label those examples, moving on to the next example after finding an example's exact label.

ALPF ALPF learners are free to choose any (example, question) pair at each turn, Thus, unlike AL and AQ, ALPF learners commonly encounter partial labels during training.

Results We run all experiments until fully annotating the training set. We then evaluate each method from two perspectives: classification and annotation. We measure each classifiers' top-1 accuracy at each annotation budget. To quantify annotation performance, we count the number questions required to *exactly label* all training examples. We compile our results in Table 6.2, rounding costs to 10%, 20% etc. The budget includes the (5%)

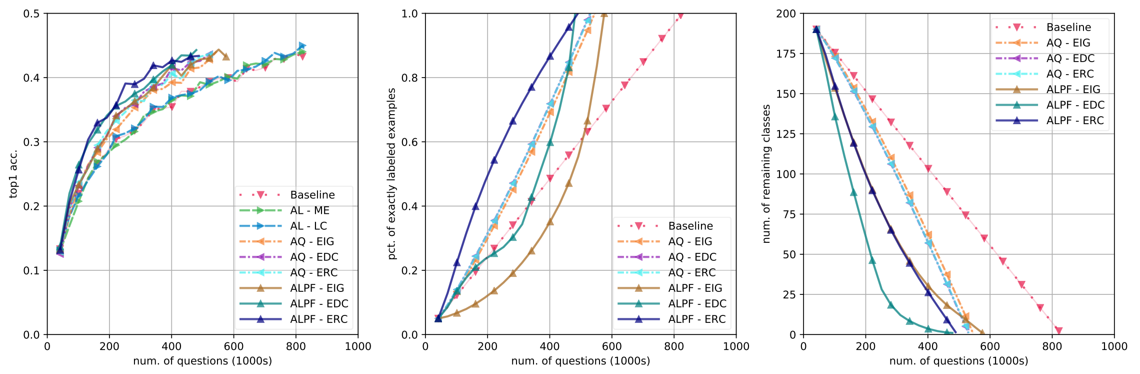


Figure 6.2: The progression of top1 classification accuracy (left), percentage of exactly labeled training examples (middle), and average number of remaining classes (right).

i.i.d. data for warm-starting. Some key results: (i) vanilla active learning does not improve over i.i.d. baselines, confirming similar observations on image classification by [153]; (ii) AQ provides a dramatic improvement over baseline. The advantage persists throughout training. These learners sample examples randomly and label to completion (until an exact label is produced) before moving on, differing only in how efficiently they annotate data. (iii) On Tiny ImageNet, at 30% of budget, ALPF-ERC outperforms AQ methods by 4.5% and outperforms the i.i.d. baseline by 8.1%.

6.3.3 Diagnostic analyses

First, we study how different amounts of warm-starting affects ALPF learners' performance with a small set of i.i.d. labels. Second, we compare the selections due to ERC and EDC to those produced through uncertainty sampling. Third, we note that while EDC and ERC appear to perform best on our problems, they may be vulnerable to excessively focusing on classes that are trivial to recognize. We examine this setting via an adversarial dataset intended to break the heuristics.

Warm-starting We compare the performance of each strategy under different percentages (0%, 5%, and 10%) of pre-labeled i.i.d. data (Figure 6.3). Results show that ERC works properly even without warm-starting, while EIG benefits from a 5% warm-start and EDC suffers badly without warm-starting. We observe that 10% warm-starting yields no further improvement.

Sample uncertainty Classic uncertainty sampling chooses data of high uncertainty. This question is worth re-examining in the context of ALPF. To analyze the behavior of ALPF learners vis-a-vis uncertainty we plot average prediction entropy of sampled data for ALPF learners with different sampling strategies (Figure 6.4). Note that ALPF

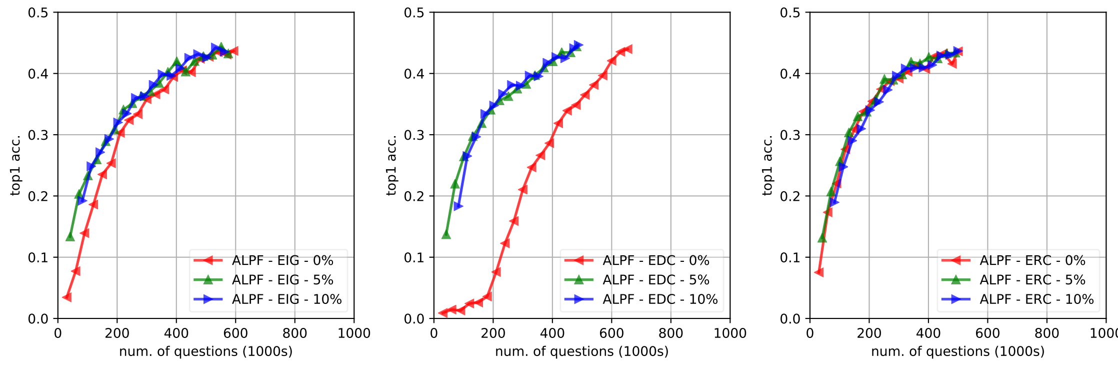


Figure 6.3: This plot compares our models under various amounts of warm-starting with pre-labeled i.i.d. data. We find that on the investigated datasets, ERC does benefit from warm-starting. However, absent warm-starting, EIG performs significantly worse and EDC suffers even more. We find that 5% warmstarting helps these two models and that for both, increasing warm-starting from 5% up to 10% does not lead to further improvements.

learners using EIG pick high-entropy data, while ALPF learners with EDC and ERC choose examples with lower entropy predictions. The (perhaps) surprising performance of EDC and ERC may owe to the cost structure of ALPF. While labels for examples with low-entropy predictions confer less information, they also come at lower cost.

Adversarial setting Because ERC goes after "easy" examples, we test its behavior on a simulated dataset where 2 of the *CIFAR10* classes (randomly chosen) are trivially easy. We set all pixels white for one class all pixels black for the other. We plot the label distribution among the selected data over rounds of selection in against that on the unperturbed *CIFAR10* in Figure 6.5. As we can see, in the normal case, EIG splits its budget among all classes roughly evenly while EDC and ERC focus more on different classes at different stages. In the adversarial case, EIG quickly learns the easy classes, thereafter focusing on the others until they are exhausted, while EDC and ERC concentrate on exhausting the easy ones first. Although EDC and ERC still manage to label all data with less total cost than EIG, this behavior might cost us when we have trivial classes, especially when our unlabeled dataset is enormous relative to our budget.

6.4 Related work

Binary identification: Efficiently finding answers with yes/no questions is a classic problem [47] dubbed *binary identification*. [76] proved that finding the optimal strategy given an arbitrary set of binary tests is NP-complete. A well-known greedy algorithm called

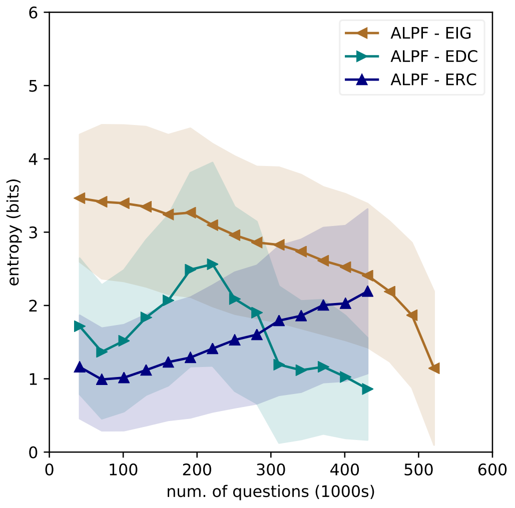
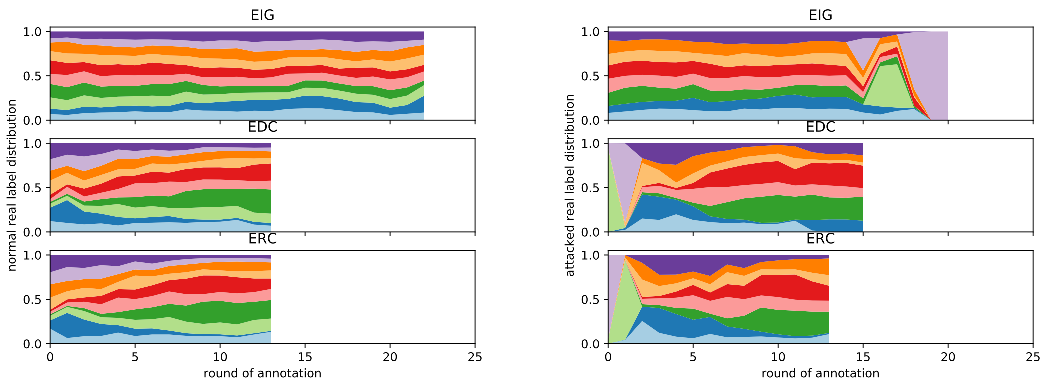


Figure 6.4: Classifier confidence (entropy of softmax layer) on *selected* examples.



(a) Normal CIFAR10 (b) Adversarial CIFAR10

Figure 6.5: Label distribution among selected examples for CIFAR 10 (left) and adversarially perturbed CIFAR 10 (right). Light green and light purple mark the two classes made artificially easy.

binary splitting [48, 111], chooses questions that most evenly split the probability mass.

Active learning: Our work builds upon the AL framework [12, 26, 155] (vs. i.i.d labeling). Classical AL methods select examples for which the current predictor is most uncertain, according to various notions of uncertainty: [29] selects examples with *maximum entropy* (ME) predictive distributions, while [28] uses the *least confidence* (LC) heuristic, sorting examples in ascending order by the probability assigned to the argmax. [157] notes that annotation costs may vary across data points suggesting cost-aware sampling heuristics but doesn't address the setting when costs change dynamically during training as a classifier grows stronger. [113] incorporates structure among outputs into an active learning scheme in the context of structured prediction. Mo et al. [122] addresses hierarchical label structure in active learning interestingly in a setting where subclasses are *easier* to learn. Thus they query classes more fine-grained than the targets, while we solicit feedback on more *general* categories.

Deep Active Learning Deep Active Learning (DAL) has recently emerged as an active research area. [186] explores a scheme that combines traditional heuristics with pseudo-labeling. [45] notes that the softmax outputs of neural networks do not capture epistemic uncertainty [87], proposing instead to use Monte Carlo samples from a dropout-regularized neural network to produce uncertainty estimates. DAL has demonstrated success on NLP tasks. [206] explores AL for sentiment classification, proposing a new sampling heuristic, choosing examples for which the expected update to the word embeddings is largest. Recently, [158] matched state of the art performance on named entity recognition, using just 25% of the training data. [82] and [88] explore other measures of uncertainty over neural network predictions.

Learning from partial labels Many papers on learning from partial labels [27, 58, 127] assume that partial labels are given a priori and fixed. [58] formalizes the partial labeling problem in the probabilistic framework and proposes a minimum entropy based solution. [127] proposes an efficient algorithm to learn classifiers from partial labels within the max-margin framework. [27] addresses desirable properties of partial labels that allow learning from them effectively. While these papers assume a fixed set of partial labels, we *actively* solicit partial feedback. This presents new algorithmic challenges: (i) the partial labels for each data point changes across training rounds; (ii) the partial labels result from active selection, which introduces bias; and (iii) our problem setup requires a sampling strategy to choose questions.

6.5 Conclusion

Our experiments validate the active learning with partial feedback framework on large-scale classification benchmarks. The best among our proposed ALPF learners fully labels the data with 42% fewer binary questions as compared to traditional active learners. Our diagnostic analysis suggests that in ALPF, it's sometimes more efficient to start with "easier" examples that can be cheaply annotated rather than with "harder" data as often suggested by traditional active learning.

Table 6.2: Results (N/A indicates data has been fully labeled)

	Annotation Budget (w.r.t. baseline labeling cost)						**Labeling Cost**
	10%	20%	30%	40%	50%	100%	
TinyImageNet							
Baseline	0.186	0.266	0.310	0.351	0.354	0.441	827k
AL - ME	0.169	0.269	0.303	0.347	0.365	-	827k
AL - LC	0.184	0.262	0.313	0.355	0.369	-	827k
AQ - EIG	0.186	0.283	0.336	0.381	0.393	-	545k
AQ - EDC	0.196	0.291	0.353	0.386	0.415	-	530k
AQ - ERC	0.194	0.295	0.346	0.394	0.406	-	531k
ALPF - EIG	0.203	0.289	0.351	0.384	0.420	-	575k
ALPF - EDC	**0.220**	0.319	0.363	0.397	0.420	-	482k
ALPF - ERC	0.207	**0.330**	**0.391**	**0.419**	**0.427**	-	491k
CIFAR100							
Baseline	0.252	0.340	0.412	0.437	0.469	0.537	337k
AL - ME	0.237	0.321	0.388	0.419	0.458	-	337k
AL - LC	0.247	0.332	0.398	0.432	0.468	-	337k
AQ - EIG	0.266	0.354	0.443	0.485	0.502	-	208k
AQ - EDC	0.264	0.366	0.439	0.483	0.508	-	215k
AQ - ERC	0.256	0.366	0.453	0.479	0.496	-	215k
ALPF - EIG	0.263	0.341	0.423	0.466	0.497	-	235k
ALPF - EDC	**0.281**	0.367	0.442	0.479	0.518	-	193k
ALPF - ERC	0.273	**0.379**	**0.464**	**0.502**	**0.526**	-	187k
CIFAR10							
Baseline	0.645	0.718	0.757	0.778	0.792	0.829	170k
AL - ME	0.663	0.709	0.759	0.763	0.800	-	170k
AL - LC	0.644	0.724	0.753	0.780	0.792	-	170k
AQ - EIG	0.654	0.747	0.791	0.806	0.823	-	89k
AQ - EDC	0.675	0.746	0.784	0.789	0.826	-	95k
AQ - ERC	0.682	0.750	0.771	0.811	0.822	-	96k
ALPF - EIG	0.673	0.741	0.786	0.815	0.813	-	124k
ALPF - EDC	**0.676**	**0.752**	**0.797**	0.832	N/A	-	74k
ALPF - ERC	0.670	0.743	**0.797**	**0.833**	N/A	-	74k

Chapter 7

Self-Supervised Freespace Forecasting

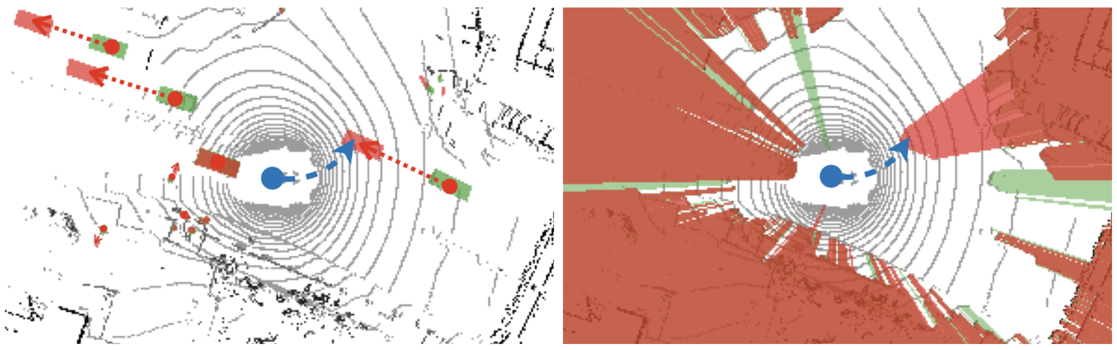


Figure 7.1: What are good 3D representations that support planning in dynamic environments? We visualize a typical urban motion planning scenario from a bird's-eye view, where an autonomous vehicle (AV) awaits an unprotected left turn. We highlight a candidate plan with a blue arrow, whose endpoint represents where the AV will be in 1s. An *object-centric* representation (left), as adopted by standard perception stacks, focuses on objects properties (their shape, orientation, position, etc.) both at the current time step and the future. Alternatively, a *freespace-centric* representation directly captures the freespace of the surrounding scene, and can be readily obtained by raycasting measurements from a depth (e.g., LiDAR) sensor. Forecasting a future version (in 1s) of either representation could help the AV identify a potential collision associated with the candidate plan, however at wildly different annotation costs. Forecasting future *object* trajectories requires a massive amount of object and track labels to train perceptual modules. Instead, we explore *future freespace*, whose forecasting can be naturally self-supervised by simply letting time move forward and raycasting future sensor measurements. We propose approaches to *planning with forecasted freespace* and *learning to plan with future freespace*.

7.1 Introduction

Motion planning in dynamic environments requires forecasting how the scene imminently evolves. What representation should we forecast to support planning? In practice, standard autonomy stacks forecast a semantic *object-centric* representation by building perceptual modules such as object detection, tracking, and prediction [181]. However, in the context of machine learning, training these modules comes at an enormous annotation cost, requiring massive amounts of data manually annotated with *object* labels, including both 3D trajectories and semantic categories (e.g., cars, pedestrians, bicyclists, etc). With autonomous fleets gathering petabytes of data, it's impossible to label data at a rate that keeps up with the rate of data collection.

To avoid the need for such costly annotations, and to enable learning at scale, we explore an alternative *freespace-centric* representation to support motion planning (Fig. 7.1). We believe this is effective for two primary reasons. First, freespace is a natural cue for safe planning - it is generally important to avoid straying into occupied space, regardless of what is occupying it. Second, gathering training data for freespace forecasting is *annotation-free* given LiDAR scans recorded from an autonomous vehicle.

In this work, we propose two approaches for using a *freespace-centric* representation to assist with planning. First, we explore *freespace forecasting* as a self-supervised learning task. We point out essential modeling choices for building an effective predictor that forecasts freespace. Then, given an off-the-shelf black-box motion planner, we demonstrate that self-supervised future freespace predictions can be used to identify candidate plans that are likely to collide with objects in the near future.

Lastly, we propose using *future freespace* as an additional source of supervision when learning to plan. Many planners learn from expert demonstrations, and for example, learn to imitate good habits like maintaining a wide safety margin when approaching pedestrians in the street. However, it is difficult for the learner to know which other actions are bad, since there may have been multiple reasonable actions that could have been taken. We use *future freespace* to identify a subset of other actions that are clearly poor because they collide with an obstacle. We empirically show that imitative learning-based planners with such additional supervision produce motion plans that are far more safe and less likely to induce collisions.

Contributions: We explore a self-supervised *freespace-centric* representation as an alternative to the predominantly supervised *object-centric* representation. We are the first to integrate self-supervised freespace predictions with an existing planner and demonstrate promising results. We also propose simple modifications to existing learning approaches to

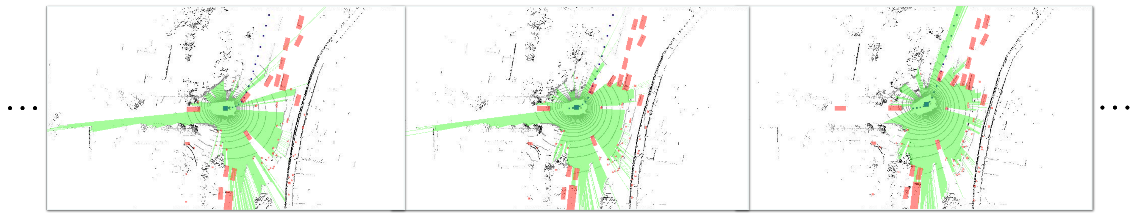


Figure 7.2: We illustrate a raw data log collected by an AV. In practice, such logs come in abundance. The crucial question is: *how do we use them to support local planning?* The widely-adopted approach tries to provide planners with knowledge of where objects are and will be (red). We try to provide knowledge of what future freespace looks like (green). We discuss two ways of how existing planners can use future freespace. First, we develop freespace forecasting models and demonstrate that their *predicted* future freespace can be used to improve off-the-shelf planners. Second, we show how to use *ground-truth* future freespace as additional supervision while learning to plan and demonstrate improvements by doing so.

planning that allow future freespace to be used as an additional source of self-supervision. Finally, we demonstrate promising results on planning benchmarks.

7.2 Related work

Geometric planning: Classic planning algorithms such as A* [62] D* [167], PRM [86], and RRT* [84] usually assume static scene geometry and focus on efficiently finding the shortest collision-free path within the navigable freespace. A common workaround for motion planning in dynamic environments is to replan at high frequencies and reactively avoid moving objects [8, 135, 168, 183]. To avoid reactive planning, one must be able to forecast future evolution of geometry. This is typically done by building a modular perception pipeline that contains components for object detection, tracking, and forecasting. However, massive amounts of training data and annotated labels are required to train perception modules for discrete object classes. Purely geometric planning approaches also commonly suffer from ambiguous interpretation of geometry (e.g. aggressively avoiding leaves blowing in the wind) or may not pick up on semantic cues (e.g. driving onto an empty opposite lane).

Behavior cloning: End-to-end learned approaches for autonomous driving have emerged as simple alternatives to modular autonomy stacks, with imitation learning methods showing particular promise [25, 145]. Imitation learning is generally split into two major classes: behavioral cloning and inverse optimal control (inverse reinforcement learning) [131]. Behavioral cloning refers to methods that learn a direct mapping from observations to actions using expert demonstrations. ALVINN [137] is a classic example of behavioral cloning for road following which uses a neural network to learn a mapping from image to steering angle. More recently, [11] used a deep convolutional network to demonstrate real-world vehicle control in a variety of driving scenarios. Another approach uses video game driving demonstrations to train a network that maps images to driving affordances that can be directly used for control [20]. [24] trains a network to produce steering and acceleration commands from input images while also conditioning on a high level-command. More recently, [21] proposes a new approach to behavior cloning. Their results suggest that a privileged imitative learner, despite performing worse than the expert it learns from, may serve as a better teacher to non-privileged imitative learners, by providing richer supervision.

Inverse optimal control: Inverse optimal control (or inverse reinforcement learning) attempts to recover an unknown cost function from a set of expert demonstrations which can then be used for planning. [1] developed a seminal approach that cast the cost/reward model as a linear function of state features whose feature weights could be learned from expert demonstrations. Maximum Margin Planning (MMP) [142] is another classic approach that used a structured margin loss to learn a cost map that can produce expert-like trajectories via dynamic programming. [213] use the maximum-entropy principle to select solutions that show the least commitment to the training data, avoiding ambiguities that may arise if expert demonstrations are imperfect. This approach has recently been improved by using neural networks to approximate the underlying cost model [192]. Similarly, [203] extends MMP by using a deep neural network trained end-to-end with a multi-task loss to produce cost maps for trajectory scoring.

Additional supervision: One well-known challenge for imitation learning is figuring out how to recover from mistakes - commonly referred to as the ”compounding error” problem [11, 137, 147]. [147] show that an effective solution is to have the expert interactively provide feedback by correcting the actions executed by a policy that is learning online. However, this is potentially dangerous to implement in the real world and may result in only sparse feedback. A more widely adopted strategy is to instead learn offline from historical (non-interactive) driving logs. Many researchers have used the CARLA simulator to record driving logs at scale using the built-in autopilot system [24, 25, 35, 145]. Simulation is particularly attractive because one has access to ground-truth labels of objects and the environment, which greatly simplifies learning [21]. However, transferring policies trained in simulation to the real world remains an active area of research. [7] perturb mid-level representations of real-world historical data to simulate nontrivial driving scenarios and were able to deploy their model on a real car. Our freespace forecasting approach is able to learn from both real historical data and simulated data, and works directly with raw sensor data instead of needing object labels.

Self-supervised learning: Self-supervised learning has recently emerged as an effective approach for many robotic manipulation tasks [39, 103, 125, 202]. However, its usage in mobile robotics is far less prevalent, with most methods aimed at solving perception tasks like road detection [30], aerial image analysis [154], and lidar/camera depth completion [115]. One early application of self-supervised learning to robot navigation developed by [166] learned mappings from both online and offline perceptual data to planning costs, demonstrating navigation on the Crusher robot. More recently, [81] showed that a navigation system based on a generalized computation graph trained with self-supervised deep reinforcement learning (DRL) was able to outperform standard DRL approaches in both simulation and real-world RC car experiments. In this work, we pose future freespace forecasting as a scalable source of self-supervision and show that it is effective for motion planning.

Freespace as a representation: A few works have explored the question of *estimating* freespace. [44] estimate a top-view probabilistic occupancy map to track people in an indoor setting with a multi-camera setup. [66] estimate indoor freespace from a single image by leveraging "boxy" object detectors. Thanks to the progress in 3D sensing, recent works have been building upon freespace *measured* through depth sensors (e.g. LiDAR). [33, 41, 73, 123, 130] pose occupancy grid maps (OGMs) prediction as a self-supervised learning task and explore effective neural net architectures for this task. Our work on forecasting freespace is similar to prior works on predicting OGMs in learning with *self-supervision*. However, our work extends beyond the forecasting task itself in three meaningful aspects. First, we demonstrate how off-the-shelf planners can use forecasted freespace. Second, we demonstrate how to learn planners with future freespace as additional supervision. Third, we demonstrate improvements in terms of planning performance. Most recently, [146] learn to predict semantic occupancy maps. [150] learn to predict *future* semantic occupancy maps as a representation that supports downstream planning and demonstrate improvement in planning performance. Our work differs in that our *freespace-centric* representation is annotation-free and therefore more scalable.

7.3 Method

Raw logs of autonomous fleets naturally provide an abundance of *aligned* sensor data sequences $\mathbf{x}$ and ego-vehicle trajectories $\mathbf{y}$, represented as collections of $\{(\mathbf{x}, \mathbf{y})\}$. We provide an example of such logs in Fig. 7.2. How do we make use of such data to learn representations that support planning? In the sections to follow, we first introduce the definition of freespace and how to compute it. Then we describe a self-supervised approach to forecasting freespace. Finally, we describe approaches to planning with forecasted freespace and learning to plan with future freespace.

7.3.1 Computing freespace

We define freespace as space free of obstacles as observed by a LiDAR sensor at a particular time instance. Given a sequence of *aligned* sensor data and ego-vehicle trajectory $(\mathbf{x}, \mathbf{y})$, let us write

$$\text{ray}(\mathbf{u}; \mathbf{x}, \mathbf{y}) \in \{0, -1, 1\}, \quad \mathbf{u} = (x, y, t), \quad \forall \mathbf{u} \in \mathbf{U} \tag{7.1}$$

to denote the freespace state of voxel $\mathbf{u}$ in the spacetime voxel grid $\mathbf{U}$, which can be *unknown* (0), *free* (-1), or *occupied* (1) respectively. Note that the spatial index of voxel $\mathbf{u}$ is 2D because we assume the local motion planners we work with operate on a ground plane and x, y represents a bird's-eye-view spatial location. When the freespace state of voxel $\mathbf{u}$ is *unknown*, the *true* state can be either *occupied* or *free* but is unobserved due to for example occlusion.

We compute freespace via raycasting. Given a 3D point cloud, we compute a 2D bird's-eye-view freespace following two steps. First, we identify LiDAR returns from the ground via a robust ground segmentation algorithm [69]. After we discard ground returns, we compute 2D freespace via a 2D visibility algorithm known as wall tracking [134]. We show example results in Fig. 7.2. This computation is automatic and does not require human annotators in the loop.

7.3.2 Forecasting freespace

Suppose we split each sensor trajectory pair $(\mathbf{x}, \mathbf{y})$ into a historical pair $(\mathbf{x_1}, \mathbf{y_1})$ and future pair $(\mathbf{x_2}, \mathbf{y_2})$, our goal is to learn a model that predicts freespace computed over $(\mathbf{x_2}, \mathbf{y_2})$ given freespace computed over $(\mathbf{x_1}, \mathbf{y_1})$. Crucially, we can compute ground-truth future freespace via raycasting for free (without human annotations)!

We train a convolutional neural network $f_\theta(\mathbf{u}; \mathbf{x_1}, \mathbf{y_1})$ with parameters θ to predict future freespace given the historical sequence $(\mathbf{x_1}, \mathbf{y_1})$ by minimizing the following loss:

$$\min_\theta \text{BCE}\Big(\sigma\big(f_\theta(\mathbf{u}; \mathbf{x_1}, \mathbf{y_1})\big), \text{ray}(\mathbf{u}; \mathbf{x_2}, \mathbf{y_2})\Big), \forall \mathbf{u} \in \mathbf{U} \tag{7.2}$$

where σ represents the sigmoid function and BCE stands for Binary Cross Entropy. Here, we use $\mathbf{U}$ to represent the voxel grid with future timestamps. The neural network produces *logits*, which are then converted to probabilities of voxels belonging to freespace through sigmoid.

Here, we formulate freespace forecasting as a binary classification and do not represent a *unknown* state as a separate state. This is because, as we have mentioned, when the ground-truth freespace state of a voxel future is *unknown*, its *true* state is either *occupied* or *free*. When computing the binary cross entropy loss, we ignore such voxels with an ambiguous freespace state.

Residual forecasting: In most scenarios, future freespace does not look much different from historical freespace. This means we may be able to predict a majority of future freespace through interpolating historical freespace. Therefore, we decompose our freespace forecasting model into two parts, i.e., linear extrapolation and non-linear residual.

$$f_\theta(\mathbf{u};\mathbf{x_1},\mathbf{y_1}) = \overbrace{f_\alpha(\text{ray}(\mathbf{u_1};\mathbf{x_1},\mathbf{y_1}))}^{\text{linear extrapolation}} + \overbrace{f_{\tilde{\theta}}(\mathbf{u};\mathbf{x_1},\mathbf{y_1})}^{\text{non-linear residual}} \tag{7.3}$$

where f_α represents a linear extrapolation over spatially-aligned historical freespace and $f_{\tilde{\theta}}$ represents a non-linear predictor that forecasts *residual logits*. As we will show in Tab. 7.1, residual forecasting (7.3) is crucial to good accuracy.

7.3.3 Planning with forecasted freespace

Now we have introduced a self-supervised approach to freespace forecasting, how can an off-the-shelf planner work with forecasted freespace? We answer this question in the context of planners learned via both behavior cloning (BC) and inverse optimal control (IOC).

Behavior cloning (BC): A behavior cloning planner takes sensor data and ego-trajectories $\mathbf{x_1},\mathbf{y_1}$ as input and predicts an expert-like future trajectory $\hat{\mathbf{y}}_\mathbf{2}$. The planner needs to know if the ego-vehicle can safely traverse each space-time voxel along the future trajectory. Our freespace forecasting model is designed to answer such queries, with one *caveat*: the model is trained to output a *soft* probability. We have to introduce a threshold that turns *soft* probabilities into *hard* decisions, similar to the fact that we have to pick a confidence score threshold for object detectors in standard autonomy stacks. Let τ be the threshold, we can test if a candidate future trajectory $\mathbf{y} = \{\mathbf{u}\}$ is safe by

$$q = \wedge_{\mathbf{u}\in\hat{\mathbf{y}}_\mathbf{2}} \left[f_\theta(\mathbf{u};\mathbf{x_1},\mathbf{y_1}) \leq \tau\right] \tag{7.4}$$

The planner passes the test of predicted future freespace when q is true. When it fails, we override the plan with a fall-back option, such as emergency braking maneuvers.

Inverse optimal control (IOC): An inverse optimal control approach to planning learns a cost map that scores potential trajectories, where the best one is found through optimization. We define the cost of a candidate trajectory $\hat{\mathbf{y}}_{\mathbf{2}}$ to be the sum of costs along its spacetime points:

$$C_\psi(\hat{\mathbf{y}}_{\mathbf{2}}; \mathbf{x_1}, \mathbf{y_1}) = \sum_{\mathbf{u} \in \hat{\mathbf{y}}_{\mathbf{2}}} \text{cost}_\psi(\mathbf{u}; \mathbf{x_1}, \mathbf{y_1}) \tag{7.5}$$

where cost_ψ is a spacetime *costmap* generated by a neural net, structurally similar to the freespace forecaster from Sec. 7.3.2. It is important that any candidate trajectory $\hat{\mathbf{y}}_{\mathbf{2}}$ maintains consistent and smooth dynamics with its immediate past $\mathbf{y_1}$ hence the conditioning. When integrating forecasted freespace into IOC planners, we directly modify the costmap to ensure that voxels predicted to be likely occupied incur very large costs,

$$C_{\psi,\theta}(\hat{\mathbf{y}}_{\mathbf{2}}; \mathbf{x_1}, \mathbf{y_1}) = \sum_{\mathbf{u} \in \hat{\mathbf{y}}_{\mathbf{2}}} [(\text{cost}_\psi + \gamma f_\theta)(\mathbf{u}; \mathbf{x_1}, \mathbf{y_1})] \tag{7.6}$$

where γ is a predefined cost w.r.t. future freespace violation.

Implementation: An IOC planner may search over an exponentially large number of potential trajectories with dynamic programming [142], explore a local set of trajectories through gradient based optimization [141], or evaluate a set of sampled trajectories through exhaustive search [203]. We make use of the latter. Following [203], instead of modeling smoothness as part of the costmap, we enforce them as a constraint by restricting the space of viable trajectories $\mathbf{Y}(\mathbf{y_1})$ that is searched. This is formally equivalent to assigning trajectories not in $\mathbf{Y}(\mathbf{y_1})$ to be infinite cost.

$$\min_{\mathbf{y} \in \mathbf{Y}} C_\psi(\mathbf{y}; \mathbf{x_1}, \mathbf{y_1}) = \min_{\mathbf{y} \in \mathbf{Y}(\mathbf{y_1})} \sum_{\mathbf{u} \in \mathbf{y}} \text{cost}_\psi(\mathbf{u}; \mathbf{x_1}) \tag{7.7}$$

When the space of viable trajectories $\mathbf{Y}(\mathbf{y_1})$ is available, we found it useful to restrict the freespace forecasting loss to the set of spacetime voxels reachable by the ego-vehicle. This results in a *sparse* loss in contrast to the original *dense* one. We will refer to freespace forecasting learned with the *sparse* loss as *planning-aware* freespace forecasting.

$$\min_\theta \text{BCE}\Big(\sigma\big(f_\theta(\mathbf{u}; \mathbf{x_1}, \mathbf{y_1})\big), \text{ray}(\mathbf{u}; \mathbf{x_2}, \mathbf{y_2})\Big), \forall \mathbf{u} \in \mathbf{Y}(\mathbf{y_1}) \tag{7.8}$$

7.3.4 Learning to plan with future freespace

We have discussed how an off-the-shelf planner can work with forecasted freespace. In particular, we show one can modify an IOC planner's costmap based on predicted future freespace. A follow-up question is: can we use ground-truth future freespace to learn a costmap that *naturally* reflects future freespace?

An IOC planner learns a neural net to predict a space-time costmap. The network will be trained to ensure that ground truth future trajectory $\mathbf{y_2}$ has lower cost than others:

$$C_\psi(\mathbf{y_2};\mathbf{x_1},\mathbf{y_1}) \leq \min_{\mathbf{y}\in\mathbf{Y}} C_\psi(\mathbf{y};\mathbf{x_1},\mathbf{y_1}) \tag{7.9}$$

Because not all alternative trajectories are equally bad, one often introduces a penalty that ensures the ground-truth *dominates* over those trajectories that lie far away by a margin $l(\mathbf{y},\mathbf{y_2})$ [142, 203]:

$$C_\psi(\mathbf{y_2};\mathbf{x_1},\mathbf{y_1}) \leq \min_{\mathbf{y}\in\mathbf{Y}} \left(C_\psi(\mathbf{y};\mathbf{x_1},\mathbf{y_1}) - l(\mathbf{y},\mathbf{y_2})\right) \tag{7.10}$$

The margin term $l(\mathbf{y},\mathbf{y_2})$ is often chosen to be a measure of dissimilarity between $\mathbf{y}$ and $\mathbf{y_2}$, for example, Euclidean distance:

$$l(\mathbf{y},\mathbf{y_2}) = \mathrm{Dist}(\mathbf{y},\mathbf{y_2}) \tag{7.11}$$

One can rewrite the constraint from (7.10) as a loss that penalizes the cost of the ground-truth while maximizing the cost of the worst-offender:

$$\mathrm{loss}(\psi) = \left[C_\psi(\mathbf{y_2};\mathbf{x_1},\mathbf{y_1}) - \left(\min_{\mathbf{y}\in\mathbf{Y}} C_\psi(\mathbf{y};\mathbf{x_1},\mathbf{y_1}) - l(\mathbf{y},\mathbf{y_2})\right)\right]^+ \tag{7.12}$$

where $[\cdot]^+ = \max(\cdot, 0)$. The minimization reaches the minimum at 0 when (7.10) is satisfied.

Importantly, [203] query additional supervision in the form of object bounding boxes. These bounding boxes in future frames are converted to a binary object occupancy grid, denoted by O, as visualized by red in Fig. 7.2. If $O[\mathbf{u}] = 1$, there is an object occupying spacetime voxel $\mathbf{u}$. Any trajectory $\mathbf{y}$ that appears at such spacetime voxels should bear an additional *margin* cost for collisions:

$$l(\mathbf{y},\mathbf{y_2}) = \mathrm{Dist}(\mathbf{y},\mathbf{y_2}) + \gamma_o \sum_{\mathbf{u}\in\mathbf{y}} O[\mathbf{u}] \tag{7.13}$$

where γ_o is a predefined cost of object collision. Instead of relying on human annotations, our approach extracts supervision from raycasted future-freespace, as visualized by green in Fig. 7.2.

$$l(\mathbf{y},\mathbf{y_2}) = \mathrm{Dist}(\mathbf{y},\mathbf{y_2}) + \gamma \sum_{\mathbf{u}\in\mathbf{y}} [\mathrm{ray}(\mathbf{u};\mathbf{x_2},\mathbf{y_2})]^+ \tag{7.14}$$

where γ is a predefined cost of future freespace violation.

7.4 Experiments

We use CARLA to evaluate freespace forecasting as a task itself. We use both NoCrash and nuScenes to evaluate planning performance. On one hand, NoCrash offers an interactive environment where an agent's action has lasting consequences, allowing for on-policy evaluation; on the other hand, nuScenes offers real-world sensor data and driving scenarios, allowing for realistic off-policy evaluation.

CARLA and NoCrash: CARLA is an open-source urban driving simulator [36] and NoCrash is the latest planning benchmark on CARLA [25]. On NoCrash, an agent succeeds if it completes a predefined route in time without collisions. NoCrash features various towns, weather conditions, and traffic densities. Town 1 is for training and Town 2 is for testing. A subset of weather conditions is also held out for testing. An agent has access to a sensor suite to perceive the environment and needs to produce control signals to apply to motors.

nuScenes: nuScenes is one of the latest real-world driving datasets collected by autonomous fleets. We choose nuScenes because its unique release of CAN bus data makes it possible to implement our baseline motion planner [203]. Since the official server does not evaluate planning, we create a protocol for nuScenes to evaluate planning. We randomly split the 850 annotated scenes into training (550), validation (150), and test sets (150), which amounts to about 17K, 5K, 4K frames respectively.

We refer readers to the supplementary materials for implementation details such as neural net architectures. We plan to make our implementation publicly available.

7.4.1 Freespace Forecasting

Setup: We let a driving agent roam around in Town 1 under an autopilot policy [25] to collect 400 trajectories for training and and 100 trajectories for validation. We follow the same practice to collect 100 trajectories in Town 2 for testing. In total, we have about 164K frames for training, 39K for validation, and 31K for testing.

Evaluation: Under the binary classification formulation, two classes in freespace forecasting turns out to be highly imbalanced. Compared to occupied space, freespace constitutes a vast majority of freespace states in the future at a rate of 35 to 1. Therefore, we plot a precision-recall curve w.r.t. occupied space and compute average precision. We also evaluate the maximum F1 score on the PR curve.

Residual forecasting: We test the idea of residual forecasting, including four variants as shown in Tab. 7.1. Our results suggest residual forecasting is highly effective. Please refer to the caption for details.

Scalability: Learning to forecast freespace requires no human annotation. We evaluate

α, θ	Town 1(val)			Town 2(test)		
	BCE	F1	AP	BCE	F1	AP
$(\alpha_0, \mathbf{0})$	0.594	0.659	0.453	0.377	0.740	0.561
$(\alpha, \mathbf{0})$	0.101	0.688	0.610	0.089	**0.786**	0.730
$(\mathbf{0}, \theta)$	0.043	0.687	0.752	0.099	0.406	0.364
(α, θ)	**0.034**	**0.772**	**0.830**	**0.047**	0.755	**0.773**

Table 7.1: Ablation studies on residual forecasting. $(\alpha_0, \mathbf{0})$ replicates the latest frame as the future prediction, without any learning. $(\alpha, \mathbf{0})$ learns to linearly predict from historical freespace observations, dramatically improving accuracy on towns used for training, but reducing performance on new towns. $(\mathbf{0}, \theta)$ forecasts directly learns a nonlinear predictor without a residual, which also appears to heavily overfit to the training town. (α, θ) learns a nonlinear residual that is added to the linear prediction, which outperforms all variants.

#Logs	Town 1(val)			Town 2(test)		
	BCE	F1	AP	BCE	F1	AP
25	0.051	0.692	0.707	0.051	0.759	0.752
50	0.046	0.710	0.740	0.049	**0.760**	0.760
100	0.042	0.731	0.773	0.048	**0.760**	0.768
200	0.037	0.754	0.805	0.048	0.757	0.772
400	**0.034**	**0.772**	**0.830**	**0.047**	0.755	**0.773**

Table 7.2: Ablation studies on increasing the amount of training data. We see a dramatic improvement on towns used for training and a slower improvement on new towns in AP.

the performance of freespace forecasting models trained with an increasing amount of data. Our results in Tab. 7.2 suggest performance in training towns improves dramatically as we increase the amount of training data. The slower improvement on new towns suggests we should collect data on new towns as well. This would be particularly viable for freespace forecasting as it does not need additional human annotations.

7.4.2 Planning on NoCrash

Baseline: LBC [21] is the state-of-the-art planner on NoCrash. At test time, a LBC agent receives sensor data and a high-level instruction (*turn-left, turn-right, go-straight, follow-lane*) as input every 0.1s and outputs a trajectory in the form of a series of bird's-eye-view waypoints. In particular, there are 5 waypoints from 0.5s to 2.5s at every 0.5s. The planner then uses heuristics to select one waypoint and translates it via pure-pursuit controllers to control signals that can be applied to motors for a duration of 0.1s before re-planning when

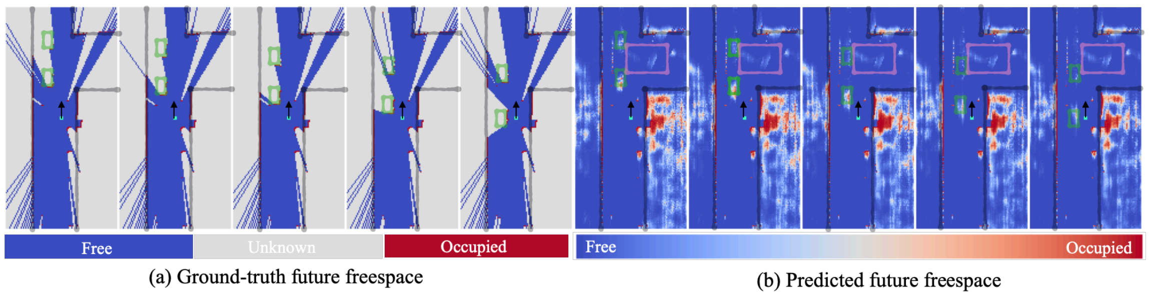

Free
Unknown
Occupied
Free
Occupied
(a) Ground-truth future freespace
(b) Predicted future freespace

Figure 7.3: Freespace forecasting qualitative result (bird's-eye-view). On the left, we visualize ground-truth future freespace. occupied is red, unknown is gray, freespace is blue. On the right, we visualize predicted future freespace. The model does not treat unknown as a separate class, instead it predicts a probability for every voxel. We highlight two vehicles in the opposing lane with green boxes. Notice the predicted freespace tracks the first (bottom) vehicle. Also, it predicts that the second vehicle could have turned left, shown by the predicted freespace in the pink box. We urge readers to view video version of this figure (and other results) in our supplement.

new sensor data and instructions arrive.

Results: We learn a residual forecasting model that takes historical freespace from the past 2s and predicts future freespace up to 2.5s. We incorporate future freespace predicted by this forecasting model into an off-the-shelf LBC planner in a *post-hoc* fashion. Based on the predicted bird's-eye-view future freespace (Fig. 7.3), we check if LBC's waypoint passes the test according to Eq. (7.4). We identify relevant voxels by drawing an oriented box for the ego-vehicle centered at the selected waypoint. When the test fails, we override LBC's plan with a trajectory that represents the action of staying still, which would then translate to the control signals that correspond to an emergency brake through the controllers. As results in Table 7.3 show, such *post-hoc* integration significantly improve the overall success rate on most testing suites compared to LBC's off-the-shelf performance. We further break down remaining failures by their causes in Fig. 7.4. When incorporating forecasted freespace *post-hoc*, we dramatically reduce the number of collisions, converting some to timeouts. This suggests avoiding imminent collisions is not enough to guarantee successful long-term planning.

7.4.3 Planning on nuScenes

Baselines: Neural motion planner (NMP) [203] is a state-of-the-art planner on real-world driving data. Since there is no official implementation for NMP, we reimplement it based on details from the paper and with help from the authors. The planner first samples a list of

Task	Weather	PV [21]	LBC [21]	LBC+FF
Empty		100 ± 0	**100** ± 1	99 ± 1
Regular	Train	95 ± 1	94 ± 3	**96** ± 2
Dense		46 ± 8	51 ± 3	**57** ± 4
Empty		100 ± 0	**70** ± 4	66 ± 3
Regular	Test	93 ± 1	62 ± 2	**73** ± 1
Dense		45 ± 6	39 ± 6	**44** ± 5

Table 7.3: Planning results on CARLA NoCrash (test town). PV: privileged agent (see [21]). LBC: Learning By Cheating (our baseline planner on CARLA). LBC+FF: we combine a learned freespace forecaster (FF) with LBC and override any plan of LBC that "collides" with the predicted future freespace. By using forecasted freespace, we significantly improve the state-of-the-art planner LBC's overall success rates on most test suites, in particular, on those test suites that have moving objects. On empty towns, using forecasted freespace leads to slightly worse performance, likely due to false positives in freespace forecasting. Fig. 7.4 further breaks down failures into collisions versus timeouts, demonstrating that freespace is even more beneficial for avoiding safety-critical collisions.

Gray shade : new approaches from this work.

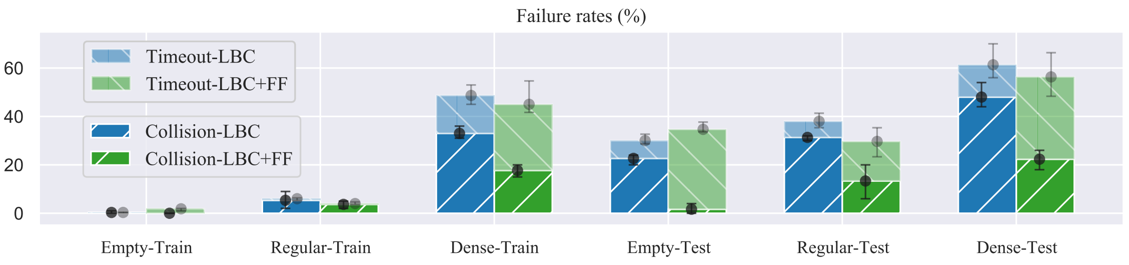


Figure 7.4: Breaking down NoCrash failures. Our approach of incorporating freespace forecasting into off-the-shelf black-box planners reduces overall failure rates in most scenarios. Most importantly, it dramatically reduces the collision rates by converting many collisions to timeouts. This suggests that avoiding imminent collisions is not enough to assure the success of long-term goal-reaching planning. For example, an autonomous vehicle would not want to drive on the opposing lane even if it does not lead to any collision in the near term.

plausible trajectories based on the ego-vehicle's kinematic state. Then it takes sensor data from the last 2 seconds as input and predicts a space-time bird's-eye-view cost map for the next 3 seconds, including 6 time-slices from 0.5s to 3.0s at every 0.5s. Finally, it score every sampled trajectory according to the predicted cost maps and pick the best-scoring one.

We implement three NMP baselines. First, we implement a *vanilla* NMP, where it penalizes trajectories based on how much they deviate from expert trajectories, as described in (7.11). Second, we implement a *object-supervised* NMP, where it applies additional penalties to trajectories that collide with object box occupancy in space-time, as described in (7.13). This baseline serves as a faithful reimplementation of [203] given what is available on nuScenes ([203] also applies penalties based on the real-time traffic light status which is not available on nuScenes). Third, we implement a NMP with improved object supervision. Specifically, we modify binary object occupancy grid O as in (7.13) by performing raycasting over O. This gives us a bird's-eye-view occlusion patterns imposed by object occupancy, based on a heuristics that the ego-vehicle should learn to stay away from not only the spacetime voxels that are *occupied* by objects but also those that are *occluded* by objects.

Evaluation: We evaluate planned trajectories within a 3s horizon at every 0.5s. We focus on two evaluation metrics: L2 error and collision rate. First, we compute the difference between the planned and the expert trajectory by the average L2 error between corresponding waypoints. Second, we evaluate how often the ego vehicle would collide with other objects. We place oriented bird's-eye-view boxes that represent the ego-vehicle at every waypoint along the planned trajectory and detects if there is any collision with other annotated boxes in the scene. One caveat for this evaluation is that we assume the scene plays out as recorded.

Results (L2P): We compare different learning-to-plan approaches. As Tab. 7.4 shows, the *vanilla* baseline achieves the lowest L2 errors but larger collision rates compared to *object-supervised* baselines. Learning with *improved* object-supervision leads to the lowest collision rates, suggesting staying away from object occlusion is a good heuristics when learning costmaps for planning. Finally, learning with future freespace reduces collision rates compared to the *vanilla* baseline without requiring human annotations.

Results (Plan w/ FF): We evaluate approaches that post-process the *vanilla* baseline with forecasted freespace. We explore both *dense* and *sparse* loss for learning the freespace forecasting model. As Tab. 7.5 shows, post-processing with planning-aware freespace forecasting (FF) greatly reduces the *vanilla* baseline's collision rates, and largely bridge the gap toward the best *object-supervised* baseline (Tab. 7.4). Interestingly, post-processing with planning-aware freespace forecasting (7.8) turns out more effective than learning to plan with future freespace. We posit that max-margin learning does not take full advantage

Learn to plan	L2 (m)			Collision (%)		
	$1s$	$2s$	$3s$	$1s$	$2s$	$3s$
vanilla	**0.50**	**1.25**	**2.80**	0.68	0.98	2.76
+ object	0.61	1.44	3.18	**0.66**	0.90	2.34
+ object*	0.61	1.40	3.16	0.71	**0.81**	**1.45**
+ freespace	0.57	1.28	2.94	**0.66**	0.87	2.17

Table 7.4: We compare learning-to-plan approaches with different additional supervision (+) on the held-out test set of nuScenes. The *vanilla* approach learns with ego-vehicle trajectories as the only source of supervision, achieving the lowest L2 errors but the largest collision rates. Learning with additional *object* (especially *object** – improved) supervision, significantly reduces the collision rates. Learning with additional *future freespace* supervision reduces collision rates without requiring human annotations. Note that vanilla+object represents a faithful reimplementation of neural motion planning (NMP) [203].

Red: approaches that need human annotations

Plan w/ FF	L2 (m)			Collision (%)		
	$1s$	$2s$	$3s$	$1s$	$2s$	$3s$
vanilla	**0.50**	**1.25**	**2.80**	0.68	0.98	2.76
→ FF (dense)	0.57	1.34	3.18	0.66	0.98	2.43
→ FF (sparse)	0.56	1.27	3.08	**0.65**	**0.86**	**1.64**

Table 7.5: We evaluate planning-with-forecasted-freespace approaches on the test set of nuScenes. We compare two loss functions for freespace forecasting: *dense* (7.2) and *sparse* (7.8). The *sparse* loss is strictly better than the *dense* loss, suggesting freespace forecasting is more effective at helping planning when it is aware of what is reachable space for the planner. In addition, post-processing a *vanilla* baseline with *planning-aware* freespace forecasting can largely bridge the gap toward learning with *object*-supervision.*

of future freespace by penalizing only the worst offender.

Conclusion: Standard approaches to planning in a dynamic environment usually require an object-centric perception system that is trained to forecast the future evolution of the scene. Providing object annotations is an expensive venture that cannot scale to the magnitude of data generated by autonomous fleets. We introduce self-supervised *future freespace forecasting* as an *annotation-free*, scalable representation for safe, expert-like motion planning and show that it serves as an effective augmentation to standard methods. In practical settings, future freespace forecasting is versatile because it can (1) be directly incorporated into existing learning-based approaches for motion planning or (2) be used as an additional post-hoc predictive collision-checking step on top of an existing motion planner.

Chapter 8

Self-Supervised Occupancy Forecasting

In the last chapter, we explore future freespace as an alternative representation as opposed to the predominant object-centric representation. We show that the forecasting of future freespace can be naturally self-supervised, where ground-truth future freespace can be directly computed via raycasting "for free". We demonstrate that such freespace supervision can potentially improve the safety aspect of local motion planning.

However, one distinctive criticism over future freespace as a representation is that it is view-specific (Fig. 8.1-b). In contrast, object-centric representation (Fig. 8.1-a) is view-independent. In this chapter, we address this criticism of future freespace. We develop a novel approach that allows us to forecast view-independent representations (Fig. 8.1-c) given view-specific freespace supervision. The key insight of this approach is the differentiable rendering of freespace.

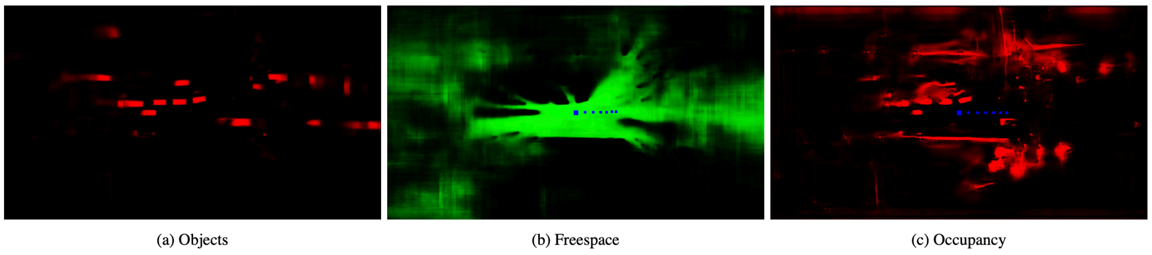


Figure 8.1: (a) View-independent object occupancy; (b) **Chap. 7:** view-specific freespace; (c) **Proposed:** view-independent latent occupancy.

In the sections to follow, we will first review future freespace and how one can integrate freespace supervision into max-margin neural motion planner [142, 203]. Then we will introduce how one could utilize differentiable freespace rendering to produce occupancy-like view-independent representations.

8.1 Background

Our goal is learning to map sensor data to "planning-ready" cost maps, as illustrated in Fig. 8.2 [142]. A "planning-ready" cost map is structured such that the optimal trajectory achieves the lowest cost.

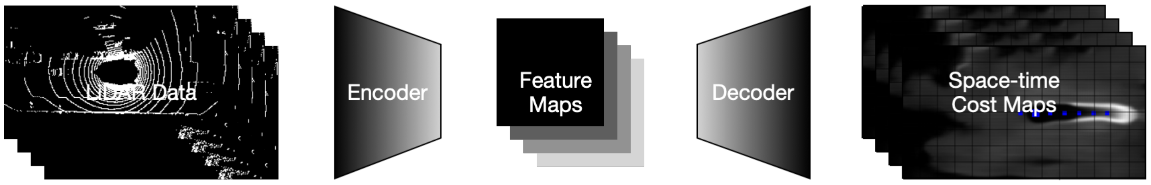


Figure 8.2: Learning to map sensor data to cost maps.

We adopt a sampling-based approach to trajectory planning (Fig. 8.3). We discretize each candidate trajectory into a series of waypoints at discrete timestamps. We assign a cost to each waypoint by using it as an index on the cost map. Finally, we select the trajectory with the lowest cost as the final plan.

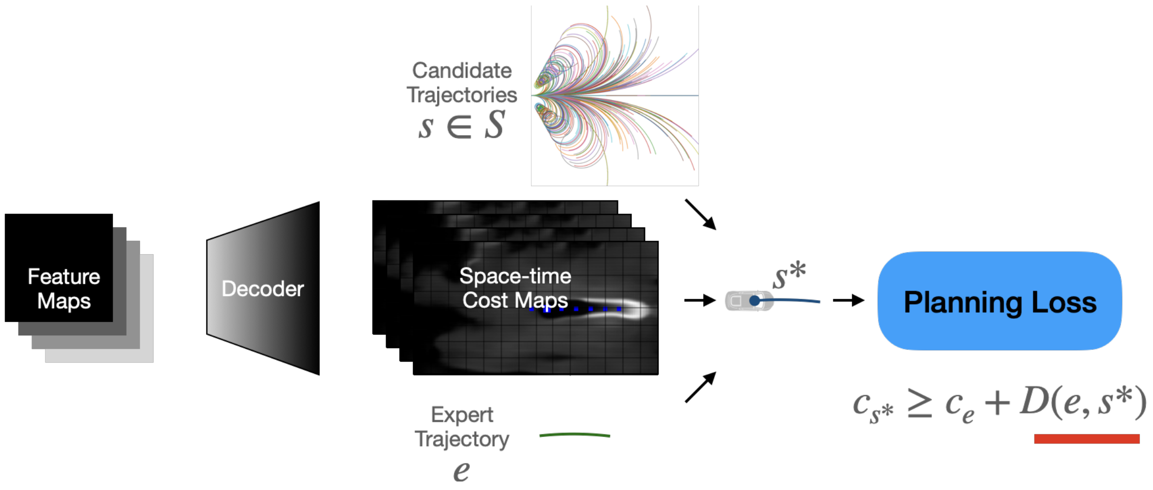


Figure 8.3: Max-margin neural motion planner.

To learn a neural network that produces such "planning-ready" cost maps, we adopt a max-margin learning framework. This requires expert demonstrations, which are readily available in driving logs. The learning objective is to score a candidate trajectory depending on how different it is from the expert trajectory. The more it looks like the expert trajectory, the lower the cost should be. We often use L2 distance between waypoints to quantify such difference. We can write the objective with the following max-margin loss function.

$$L = \max\Big(0,\ c_e - \min_s\left(c_s - D(e,s)\right)\Big) \tag{8.1}$$

Such learning setup is scalable because it does not require laborious and expensive human annotations. However, there is an important criticism about learning from demonstration alone: the loss function provides no other incentive to distinguish candidate trajectories, regardless of the context. We provide an example (Fig. 8.4) where one trajectory should cost more than the other despite the fact they are equally distant from the expert demonstration.

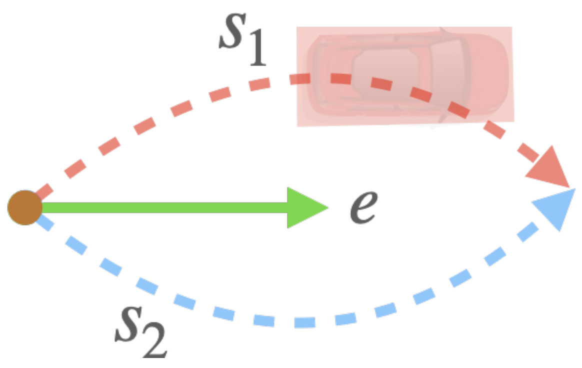


Figure 8.4: When scored, the red trajectory (s_1) should cost more than the blue trajectory (s_2) given the presence of the car.

A costly but effective remedy, as suggested by Zeng et al. [203], is to use object labels to guide cost learning. With object labels, we could modify the learning objective such that a candidate trajectory that collides with objects costs even more in addition to the deviation from expert demonstration. We refer to this approach as guided cost learning (Fig. 8.5). We can modify the loss function accordingly.

$$L_g = \max\Big(0,\ c_e - \min_s\left(c_s - D(e,s) - O(s)\right)\Big) \tag{8.2}$$

Another approach, as we have introduced in the last chapter, is to use freespace supervision instead of object labels (Fig. 8.6) to guide cost learning. We only have to substitute one term in the loss function.

$$L_g = \max\Big(0,\ c_e - \min_s\left(c_s - D(e,s) - \bar{F}(s)\right)\Big) \tag{8.3}$$

where $\bar{F}(s)$ represents the cost of a candidate trajectory "collides" with non-freespace.

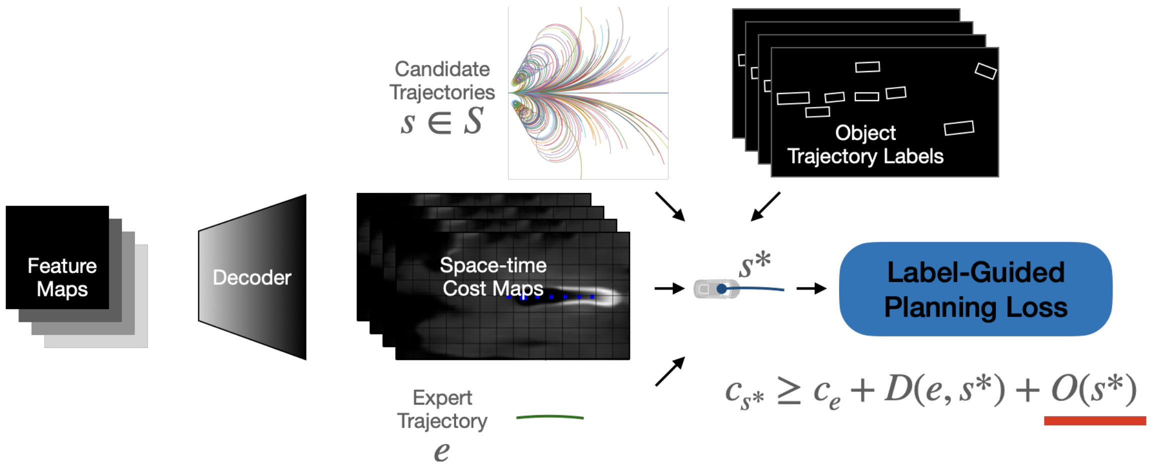


Figure 8.5: Object labels guided cost learning.

As we have shown in the last chapter, guiding max-margin cost learning with supervision in addition to expert demonstrations can significantly reduce the collision rates during motion planning. We show that object labels are the most effective but also are the most expensive. In the meanwhile, we demonstrate freespace supervision is also effective in lowering collision rates albeit requires no human supervision.

One important criticism over guided cost learning is that it does not produce an interpretable intermediate representation that describes the dynamic environment. Recent works have explored multi-task learning in order to produce such interpretable intermediate representations. Zeng et al. [203] designs a perception loss that focuses on learning to detect objects in addition to the max-margin loss that focuses on planning. However, the object detection does not directly contribute to the cost maps for planning. Because perception and planning are decoupled, a mistake in planning cannot be pinpointed to a mistake in object detection. Later works [19, 150, 204] improve the multi-task learning design and allow the result of perception and prediction to directly contribute to the cost maps.

In particular, Sadat et al. [150] proposes future semantic occupancy as an interpretable intermediate representation and propose a multi-task learning setup that consists of label-guided cost learning and semantic occupancy forecasting, as illustrated in Fig. 8.7.

In a similar fashion, we can build a multi-task learning setup that consists of freespace-guided cost learning and freespace forecasting, as illustrated in 8.8.

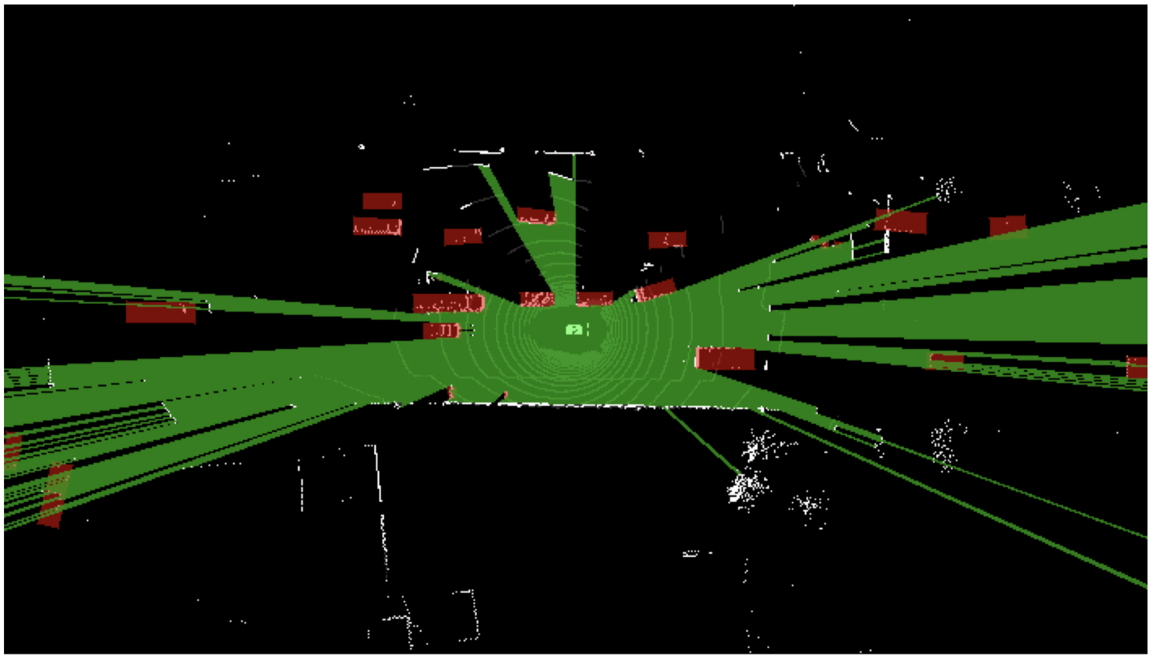


Figure 8.6: *Label-intensive* object supervision vs. *label-free* freespace supervision.

8.2 Approach

We propose an end-to-end learnable network that performs (latent) occupancy forecasting and motion planning. We plot the training-time architecture of the proposed approach in Fig. 8.9. In contrast to Fig. 8.8, note that we add an additional step named differentiable rendering. During training, the model predicts view-independent latent occupancy, which is rendered into view-specific freespace with the knowledge of LiDAR pose. Finally, the rendered freespace is compared to the observed freespace and produce gradients for learning. Notice we are learning with the exact same freespace forecasting loss, meaning we are using the same freespace supervision. But the introduction of rendering is what enables the model to produce view-independent intermediate representations (Fig. 8.1-c) with view-specific supervision.

We propose a differentiable freespace rendering algorithm, which enables the end-to-end learning of the model described in Fig. 8.9. We illustrate why such a rendering process is differentiable in Fig. 8.10. Please refer to the caption for more details.

The proposed approach has a slightly different architecture at test time, because we no longer have access to LiDAR pose at future timestamps. We modify the architecture such that the predicted latent occupancy directly contributes to the cost map. We illustrate the test-time architecture in Fig. 8.11.

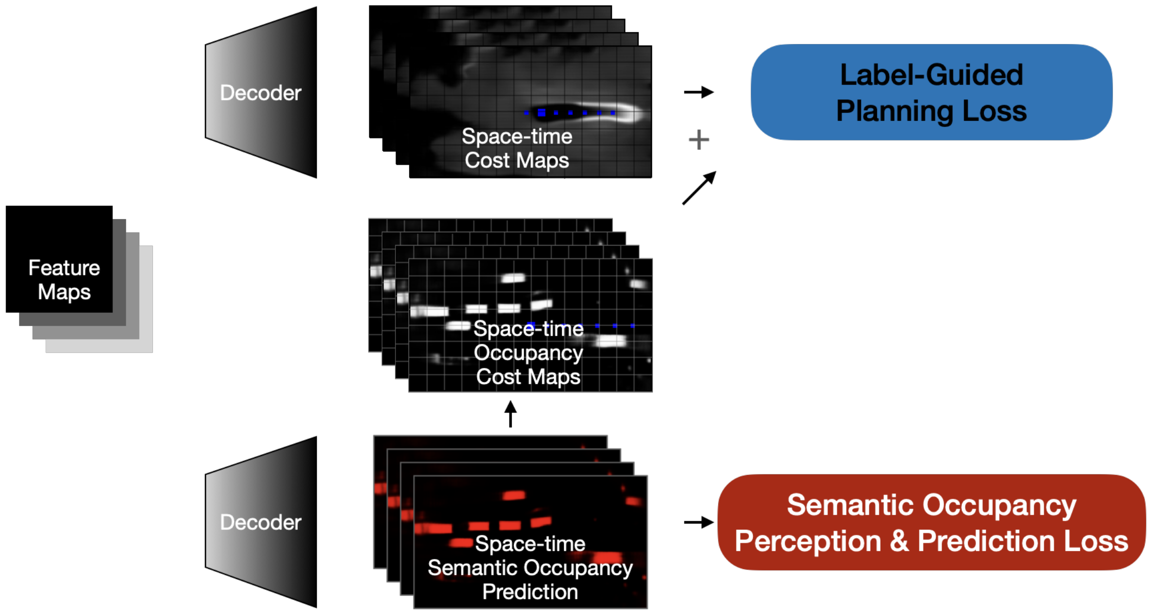

Decoder
Space-time Cost Maps
Label-Guided Planning Loss
+
Feature Maps
Space-time Occupancy Cost Maps
Decoder
Space-time Semantic Occupancy Prediction
Semantic Occupancy Perception & Prediction Loss

Figure 8.7: Multi-task: label-guided cost learning with semantic occupancy forecasting.

8.3 Experiments

We adopt the same experimental setup as in Sec. 7.4.3 of Chap. 7. We include main results in Tab. 8.1. Here we evaluate three metrics: (1) L2 distance between the output trajectory and the expert trajectory; (2) Point collision: collision rate with the ego vehicle as a point; (3) Box collision: collision rate with the ego vehicle as an oriented box.

Object label's efficacy comes at a price. Freespace supervision (Tab. 8.1-i), which requires no human supervision, only slightly underperforms object supervision (Tab. 8.1-d) at 1s and 2s horizon and outperforms it at 3s horizon. Compared to the approach from Chap. 7 (Tab. 8.1-e), our new approach (Tab. 8.1-i) achieves either same or lower collision rates across the board while providing interpretable intermediate representations.

Ablation studies: cost margin Using supervision in addition to expert demonstration to guide cost learning is crucial for achieving lower collision rates. This is true with both object supervision and freespace supervision. However, there is a trade-off. When we introduce other penalties into the cost margin, the L2 errors tend to increase as being expert-like (at all costs) is no longer the only objective. This can be seen in Tab. 8.1 by comparing (d) to (c), (g) to (f), or (i) to (h).

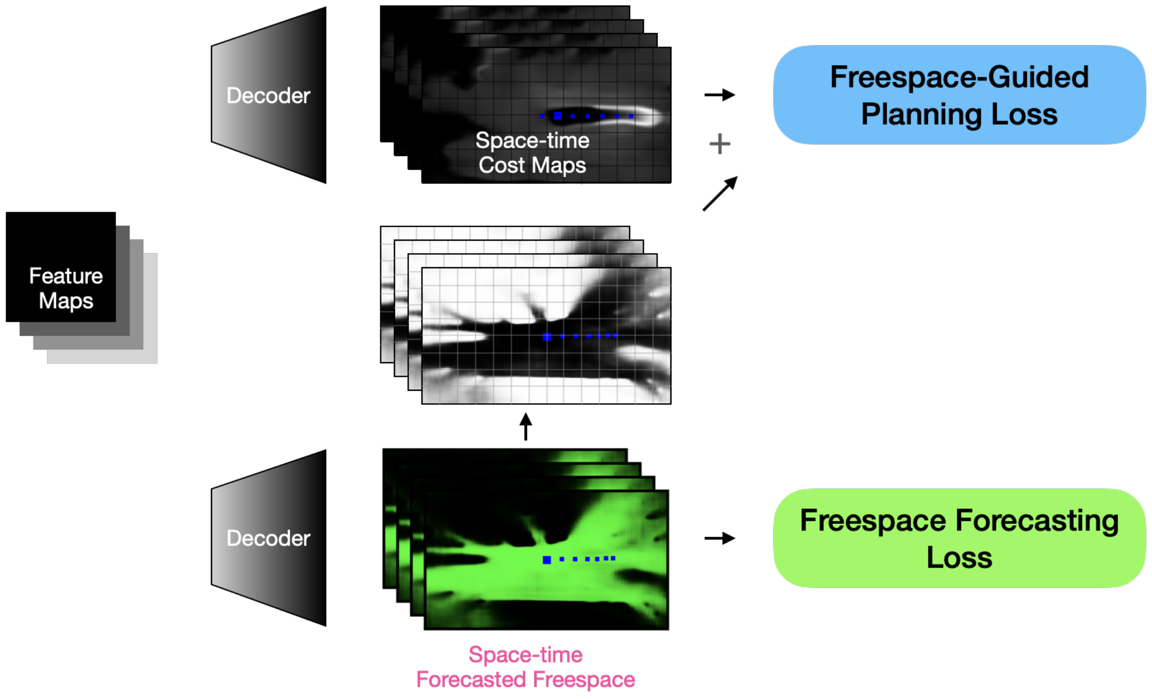


Figure 8.8: Multi-task: freespace-guided cost learning with freespace forecasting.

Ablation studies: intermediate task Learning with an intermediate task, let it be supervised object occupancy forecasting or self-supervised freespace forecasting, not only provides interpretable intermediate representations, but also reduces collision rates especially at 1s and 2s horizon. This can be seen in Tab. 8.1 by comparing (d) to (b), (g) to (e) or (i) to (e).

Ablation studies: occupancy vs. freespace Our approach produces view-independent latent occupancy with the introduction of differentiable freespace rendering. If we compare (i) to (h), we find that (i) outperforms (h) significantly especially at 3s. This suggests that view-independent occupancy as an intermediate representation is more suitable for motion planning compared to view-specific future freespace.

Problem: predicted occupancy diffuses over time We observe that occupancy forecasting suffers from a problem where predicted occupancy tends to diffuses over time. First off, this is inevitable due to the uncertain nature of future events. Second, Casas et al. [19] proposed occupancy flow to encourage more temporal consistency in order to mitigate this problem.

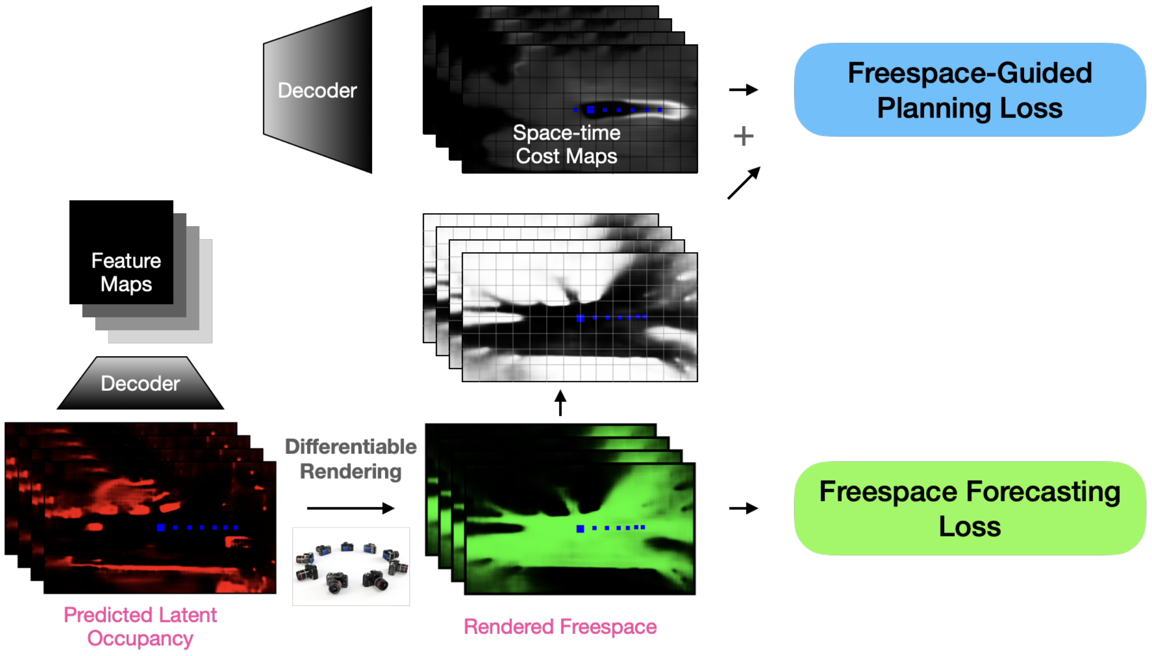


Figure 8.9: Training-time architecture of the proposed approach.

Conclusion We address an important drawback of future freespace as an intermediate representation that it is view-specific. We propose latent future occupancy that is view-independent. We introduce a novel approach that learns to forecast view-independent latent occupancy with view-specific freespace supervision. We demonstrate that latent future occupancy is more effective than future freespace in both interpretability and safety.

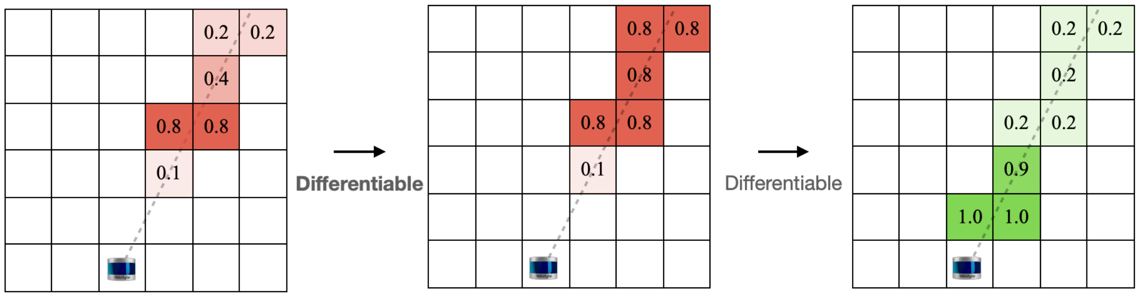


Figure 8.10: Differentiable Freespace Rendering. We start with predicted latent occupancy on the left. We first perform cumulative max along each LiDAR ray. Note that this process is differentiable as it is essentially re-indexing. This produces non-freespace, which is finally converted to freespace. Note the second step is also differentiable.

	Cost	Mid	Diff.	L2 Distance (m)			Point Collision (%)			Box Collision (%)		
	Margin	Task	Render	1s	2s	3s	1s	2s	3s	1s	2s	3s
(a)	-	-	-	0.44	1.15	2.47	**0.00**	**0.00**	0.35	0.08	0.27	1.95
(b)	O	-	-	0.53	1.25	2.67	**0.00**	**0.00**	0.08	0.04	0.12	**0.87**
(c)	-	O	-	0.43	1.08	2.33	**0.00**	0.02	0.10	0.10	0.17	1.60
(d)	O	O	-	0.59	1.34	2.82	**0.00**	**0.00**	0.07	**0.00**	**0.05**	1.03
(e)	F	-	-	0.55	1.20	2.54	**0.00**	0.01	0.04	0.06	0.17	1.07
(f)	-	F	-	**0.42**	**1.06**	**2.30**	**0.00**	0.02	0.08	0.08	0.17	1.29
(g)	F	F	-	0.52	1.22	2.64	**0.00**	**0.00**	0.08	0.02	0.10	1.10
(h)	-	F	✓	0.45	1.09	**2.30**	**0.00**	**0.00**	0.08	0.08	0.15	1.49
(i)	F	F	✓	0.67	1.36	2.78	**0.00**	0.01	**0.03**	0.04	0.09	0.88

Table 8.1: Planning results on nuScenes-val. F: Freespace. O: Objects.

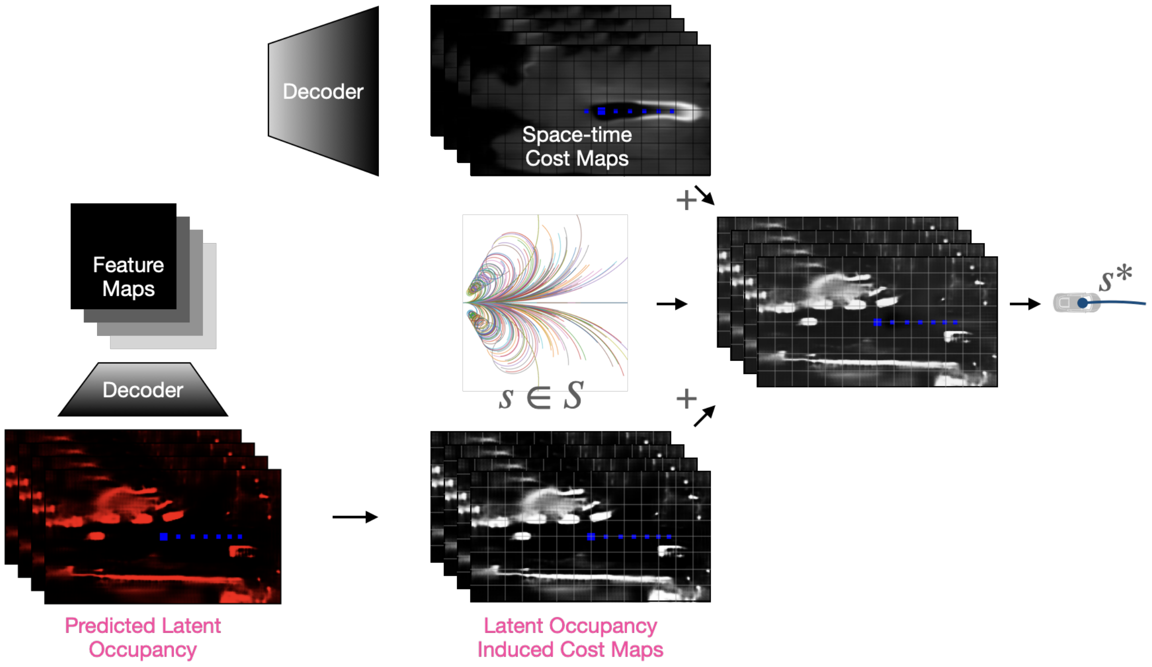


Figure 8.11: Test-time architecture of the proposed approach.

Chapter 9

Discussion

In the first part of the thesis, we explored how to build stand-alone vision models that are robust to different types of variance in sensor data. We adopt standard learning protocols and interfaces. For example, we assume that the dataset, the labels, and the input-output interfaces are fixed. In the second part of the thesis, we zoom out one level and re-examine the learning protocols and input-output interfaces through the lens of scalability.

To build safe autonomous robots, we will continue to face the challenge of robustness and scalability. I believe the most important question we should keep asking is “what is a good representation?” Here are a few properties a good representation should possess.

- **Actionable:** Planners should be able to work directly with a good representation to figure out what to do next.
- **Interpretable:** A good representation should be interpretable at least to a degree where the representation can be benchmarked.
- **Robust:** The mapping function should be invariant to different types of variance in sensor data.
- **Scalable:** Since data collection easily outpaces annotation, the learning of the mapping function should be scalable.
- **Efficient:** A good representation can be computed within reasonable computational budget (c.f. bounded rationality).

Standard autonomy software stacks adopt intuitive object-centric representations, including object bounding-box trajectories. Object detectors, object trackers, and trajectory forecasters are developed. Such object-centric representations are actionable and interpretable. However, the vision models responsible for producing such representations are not as robust in rare scenarios. Moreover, learning these vision models often require manual annotation, making them less scalable. Finally, the efficiency deteriorates when reasoning about hundreds of object instances (e.g., crowds) and possible interactions among them.

In the last two chapters, we have seen promising results that suggest vision models may learn to forecast object motion without defining objects. In other words, the notion of objects emerge from learning to forecast future sensor data. This observation has significant implications. It may suggest that we do not need to solve detection, tracking, or trajectory forecasting.

How much does scalability offer? Currently, there is no publicly available dataset that offers magnitudes more unlabeled driving logs. As a result, we cannot fully test this idea, but once such datasets become available, it would be interesting to see what the scalability can offer. In addition, we compare self-supervised representations to supervised object representations in the downstream. Is there a mid level apples-to-apples comparison?

Do objects still have a role? Practically, forecasted freespace tends to diffuse over time, making it harder to work with for planning over a longer time horizon. The traditional approach seems to shine because we can build physical models that naturally encode common sense such as object permanence.

How do we reason about interactions without objects? We did not explore the possible interaction between the ego agent's action and how the future unfolds. It is easier to reason such interactions with a notion of objects. Without such, how can we make forecasts conditioned on the ego agent's action such that one can start plan game-theoretic maneuvers?

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004. 7.2

[2] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. What is an object? In *CVPR*, pages 73–80. IEEE, 2010. 5.1

[3] John Amanatides and Andrew Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. In *EG 1987-Technical Papers*. Eurographics Association, 1987. doi: 10.2312/egtp.19871000. 4.3.2

[4] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *CVPR*, 2014. 3.4, 3.3

[5] Asma Azim and Olivier Aycard. Detection, classification and tracking of moving objects in a 3d environment. In *IV*, pages 802–807. IEEE, 2012. 5.2

[6] Aayush Bansal, Xinlei Chen, Bryan Russell, Abhinav Gupta, and Deva Ramanan. Pixelnet: Towards a general pixel-level architecture. *arXiv preprint arXiv:1609.06694*, 2016. 2.2

[7] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018. 7.2

[8] Fethi Belkhouche. Reactive path planning in a dynamic environment. *IEEE Transactions on Robotics*, 25(4):902–911, 2009. 7.2

[9] Sean Bell, C Lawrence Zitnick, Kavita Bala, and Ross Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. *arXiv preprint arXiv:1512.04143*, 2015. 2.2

[10] Rodrigo Benenson, Markus Mathias, Radu Timofte, and Luc Van Gool. Pedestrian detection at 100 frames per second. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2903–2910. IEEE, 2012. 2.2

[11] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*,

2016. 7.2

[12] George EP Box and Norman R Draper. *Empirical model-building and response surfaces.* John Wiley & Sons, 1987. 6.4

[13] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *ICCV*, volume 1, pages 377–384. IEEE, 1999. 5.2

[14] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 4.1

[15] Joachim Buhmann, Wolfram Burgard, Armin B Cremers, Dieter Fox, Thomas Hofmann, Frank E Schneider, Jiannis Strikos, and Sebastian Thrun. The mobile robot rhino. *AI Magazine*, 16(2):31–31, 1995. 4.2.1

[16] Xavier P Burgos-Artizzu, Pietro Perona, and Piotr Dollár. Robust face landmark estimation under occlusion. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1513–1520. IEEE, 2013. 3.4, **??**

[17] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019. 4.3.1, 4.1, 4.4, 4.2

[18] Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. Human pose estimation with iterative error feedback. *arXiv preprint arXiv:1507.06550*, 2015. 3.3

[19] Sergio Casas, Abbas Sadat, and Raquel Urtasun. Mp3: A unified model to map, perceive, predict and plan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14403–14412, 2021. 8.1, 8.3

[20] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015. 7.2

[21] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020. 7.2, 7.4.2, **??**, **??**, 7.3

[22] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, pages 1907–1915, 2017. 5.4

[23] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. *arXiv preprint arXiv:1904.08755*, 2019. 4.2.2

[24] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018. 7.2

[25] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9329–9338, 2019. 7.2, 7.4, 7.4.1

[26] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research (JAIR)*, 1996. 6.4

[27] Timothee Cour, Ben Sapp, and Ben Taskar. Learning from partial labels. *Journal of Machine Learning Research*, 12(May):1501–1536, 2011. 6.2.2, 6.4

[28] Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In *AAAI*, 2005. 6.4

[29] Ido Dagan and Sean P Engelson. Committee-based sampling for training probabilistic classifiers. In *ICML*, 1995. 6.4

[30] Hendrik Dahlkamp, Adrian Kaehler, David Stavens, Sebastian Thrun, and Gary R Bradski. Self-supervised monocular road detection in desert terrain. In *Robotics: science and systems*, volume 38. Philadelphia, 2006. 7.2

[31] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, pages 5828–5839, 2017. 5.4.3

[32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. IEEE, 2009. 6.1

[33] Julie Dequaire, Peter Ondrúška, Dushyant Rao, Dominic Wang, and Ingmar Posner. Deep tracking in the wild: End-to-end tracking using recurrent neural networks. *The International Journal of Robotics Research*, 37(4-5):492–512, 2018. 7.2

[34] Santosh K Divvala, Derek Hoiem, James H Hays, Alexei A Efros, and Martial Hebert. An empirical study of context in object detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1271–1278. IEEE, 2009. 2.1

[35] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016. 7.2

[36] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017. 7.4

[37] Rodney J Douglas, Christof Koch, Misha Mahowald, KA Martin, and Humbert H Suarez. Recurrent excitation in neocortical circuits. *Science*, 269(5226):981–985, 1995. 3.1

[38] Bertrand Douillard, James Underwood, Noah Kuntz, Vsevolod Vlaskine, Alastair Quadros, Peter Morton, and Alon Frenkel. On the segmentation of 3d lidar point clouds. In *ICRA*, pages 2798–2805. IEEE, 2011. 5.2

[39] Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *arXiv preprint arXiv:1710.05268*, 2017. 7.2

[40] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust rgb-d object recognition. In *IROS*, pages 681–687. IEEE, 2015. 4.1

[41] Nico Engel, Stefan Hoermann, Philipp Henzler, and Klaus Dietmayer. Deep object tracking on dynamic occupancy grid maps using rnns. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3852–3858. IEEE, 2018. 7.2

[42] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59(2):167–181, 2004. 5.2

[43] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 32(9):1627–1645, 2010. 2.1, 3.1

[44] Francois Fleuret, Jerome Berclaz, Richard Lengagne, and Pascal Fua. Multicamera people tracking with a probabilistic occupancy map. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):267–282, 2007. 7.2

[45] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. *arXiv:1703.02910*, 2017. 6.4

[46] Carolina Galleguillos and Serge Belongie. Context based object categorization: A critical survey. *Computer Vision and Image Understanding*, 114(6):712–722, 2010. 2.1

[47] Michael R Garey. Optimal binary identification procedures. *SIAM Journal on Applied Mathematics*, 1972. 6.4

[48] Michael R Garey and Ronald L. Graham. Performance bounds on the splitting algorithm for binary testing. *Acta Informatica*, 1974. 6.4

[49] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *IJRR*, 32(11):1231–1237, 2013. 4.4

[50] Golnaz Ghiasi and Charless C Fowlkes. Occlusion coherence: Localizing occluded faces with a hierarchical deformable part model. In *CVPR*, pages 1899–1906, 2014. **??**

[51] Golnaz Ghiasi and Charless C Fowlkes. Using segmentation to predict the absense of occluded parts. In *BMVC*, 2015. 3.4, **??**

[52] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587. IEEE, 2014. 2.1, 2.4.1, 3.1

[53] Ross Girshick, Forrest Iandola, Trevor Darrell, and Jitendra Malik. Deformable part models are convolutional neural networks. In *CVPR*. IEEE, 2015. 3.1

[54] Georgia Gkioxari, Pablo Arbeláez, Lubomir Bourdev, and Jagannath Malik. Articulated pose estimation using discriminative armlet classifiers. In *CVPR*, 2013. **??**

[55] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep

feedforward neural networks. In *Artificial Intelligence and Statistics (AISTATS)*, 2010. 6.3

[56] Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *ICNN*, volume 1, pages 347–352. IEEE, 1996. 3.2

[57] Ian Goodfellow, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Multi-prediction deep boltzmann machines. In *NIPS*, 2013. 3.2

[58] Yves Grandvalet and Yoshua Bengio. Learning from partial labels with minimum entropy. *Centre interuniversitaire de recherche en analyse des organisations (CIRANO)*, 2004. 6.4

[59] Timo Hackel, Jan D Wegner, and Konrad Schindler. Fast semantic segmentation of 3d point clouds with strongly varying density. *ISPRS annals of the photogrammetry, remote sensing and spatial information sciences*, 3(3):177–184, 2016. 5.2

[60] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *ICCV*, 2011. 3.4

[61] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 447–456, 2015. 2.1, 2.2

[62] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. 7.2

[63] Simon Haykin. *Neural networks and learning machines*, volume 3. Pearson Education Upper Saddle River, 2009. 3.2

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 2.3

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2016. 6.3

[66] Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering free space of indoor scenes from a single image. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2807–2814. IEEE, 2012. 7.2

[67] David Held, Jesse Levinson, Sebastian Thrun, and Silvio Savarese. Combining 3d shape, color, and motion for robust anytime tracking. In *RSS*, 2014. 5.2, 5.4.3

[68] David Held, Devin Guillory, Brice Rebsamen, Sebastian Thrun, and Silvio Savarese. A probabilistic framework for real-time 3d segmentation using spatial, temporal, and semantic cues. In *RSS*, 2016. 5.1, 5.4, 5.4

[69] Michael Himmelsbach, Felix V Hundelshausen, and H-J Wuensche. Fast segmentation of 3d point clouds for ground vehicles. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 560–565. IEEE, 2010. 7.3.1

[70] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. 3.2

[71] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *ICANN*, pages 44–51. Springer, 2011. 3.2

[72] Shaul Hochstein and Merav Ahissar. View from the top: Hierarchies and reverse hierarchies in the visual system. *Neuron*, 36(5):791–804, 2002. 3.1

[73] Stefan Hoermann, Martin Bach, and Klaus Dietmayer. Dynamic occupancy grid prediction for urban autonomous driving: A deep learning approach with fully automatic labeling. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2056–2063. IEEE, 2018. 7.2

[74] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. Diagnosing error in object detectors. In *European conference on computer vision*, pages 340–353. Springer, 2012. 2.5

[75] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013. 4.1, 4.2.1, 4.3.2, 4.4, 4.3.2

[76] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 1976. 6.4

[77] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 3.3

[78] Minami Ito and Charles D Gilbert. Attention modulates contextual influences in the primary visual cortex of alert monkeys. *Neuron*, 22(3):593–604, 1999. 3.1

[79] Ya Jin and Stuart Geman. Context and hierarchy in a probabilistic image model. In *CVPR*, 2006. 3.1

[80] Andrew E. Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *TPAMI*, 21(5):433–449, 1999. 4.2.1

[81] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018. 7.2

[82] Michael Kampffmeyer, Arnt-Børre Salberg, and Robert Jenssen. Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks. In *CVPR*, 2016. 6.4

[83] Eric R Kandel, James H Schwartz, Thomas M Jessell, et al. *Principles of neural science*, volume 4. McGraw-Hill New York, 2000. 3.2

[84] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010. doi: 10.15607/RSS.2010.VI.034. 7.2

[85] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating

image descriptions. In *CVPR*, 2015. 3.1

[86] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996. 7.2

[87] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *NIPS*, 2017. 6.4

[88] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv:1511.02680*, 2015. 6.4

[89] Eunyoung Kim and Gerard Medioni. 3d object recognition in range images using visibility context. In *IROS*, pages 3800–3807. IEEE, 2011. 4.1

[90] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2014. 6.3

[91] Klaas Klasing, Dirk Wollherr, and Martin Buss. A clustering method for efficient segmentation of 3d laser data. In *ICRA*, pages 4043–4048. IEEE, 2008. 5.2

[92] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *ICCV*, pages 863–872, 2017. 4.2.1, 5.3.3

[93] Stephen M Kosslyn, William L Thompson, Irene J Kim, and Nathaniel M Alpert. Topographical representations of mental images in primary visual cortex. *Nature*, 378 (6556):496–498, 1995. 3.1

[94] Martin Köstinger, Paul Wohlhart, Peter M Roth, and Horst Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *ICCV Workshops*, pages 2144–2151. IEEE, 2011. 3.4

[95] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. 3.1

[96] Norbert Kruger, Peter Janssen, Sinan Kalkan, Markus Lappe, Ale Leonardis, Justus Piater, Antonio Jose Rodriguez-Sanchez, and Laurenz Wiskott. Deep hierarchies in the primate visual cortex: What can we learn for computer vision? *PAMI*, 35(8): 1847–1871, 2013. 3.1

[97] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L Waslander. Joint 3d proposal generation and object detection from view aggregation. In *IROS*, pages 1–8. IEEE, 2018. 4.2.2, 4.3, 4.3.1, 5.2, 5.4.1

[98] Jason Ku, Alex D Pon, and Steven L Waslander. Monocular 3d object detection leveraging accurate proposals and shape reconstruction. In *CVPR*, pages 11867–11876, 2019. 5.2

[99] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *arXiv:1812.05784*, 2018. 5.2, 5.4.1

[100] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar

Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019. 4.1, 4.2.2, 4.3, 4.3.2, 4.4

[101] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015. 2.2

[102] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3.1, 3.2

[103] Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE, 2019. 7.2

[104] Tai Sing Lee and David Mumford. Hierarchical bayesian inference in the visual cortex. *JOSA A*, 20(7):1434–1448, 2003. 3.1

[105] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *NeurIPS*, pages 820–830, 2018. 4.2.1, 5.3.3

[106] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014. 2.3.2

[107] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, and Scott Reed. Ssd: Single shot multibox detector. *arXiv preprint arXiv:1512.02325*, 2015. 2.2

[108] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*. IEEE, 2015. 3.4, 3.2, 3.3, 3.4, 3.7

[109] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 2.1, 2.2, 2.3

[110] Jonathan L Long, Ning Zhang, and Trevor Darrell. Do convnets learn correspondence? In *NIPS*, pages 1601–1609, 2014. 3.4, **??**, 3.2

[111] Donald W Loveland. Performance bounds for binary testing with arbitrary weights. *Acta Informatica*, 1985. 6.4

[112] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 2.2

[113] Wenjie Luo, Alex Schwing, and Raquel Urtasun. Latent structured active learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2013. 6.4

[114] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *CVPR*, pages 3569–3577, 2018. 4.2.2, 5.2

[115] Fangchang Ma, Guilherme Venturelli Cavalheiro, and Sertac Karaman. Self-supervised

sparse-to-dense: Self-supervised depth completion from lidar and monocular camera. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3288–3295. IEEE, 2019. 7.2

[116] David Marr. Vision: A computational approach, 1982. 3.1

[117] David Marr and Herbert Keith Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 200(1140):269–294, 1978. 4.1

[118] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *ICANN*, pages 52–59. Springer, 2011. 3.2

[119] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11):205, 2017. 5.1, 5.2

[120] Gregory P Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K Wellington. Lasernet: An efficient probabilistic 3d object detector for autonomous driving. In *CVPR*, pages 12677–12686, 2019. 4.1, 4.2.2

[121] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 1995. 6.2

[122] Yuji Mo, Stephen D Scott, and Doug Downey. Learning hierarchically decomposable concepts with active over-labeling. In *International Conference on Data Mining (ICDM)*, 2016. 6.4

[123] Nima Mohajerin and Mohsen Rohani. Multi-step prediction of occupancy grid maps with recurrent neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 7.2

[124] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, et al. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597, 2008. 5.1

[125] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2146–2153. IEEE, 2017. 7.2

[126] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 3.2

[127] Nam Nguyen and Rich Caruana. Classification with partial labels. In *Knowledge discovery and data mining (KDD)*. ACM, 2008. 6.4

[128] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *arXiv preprint arXiv:1505.04366*, 2015. 3.2

[129] Aude Oliva and Antonio Torralba. The role of context in object recognition. *Trends in cognitive sciences*, 11(12):520–527, 2007. 2.1

[130] Peter Ondruska and Ingmar Posner. Deep tracking: Seeing beyond seeing using

recurrent neural networks. *arXiv preprint arXiv:1602.00991*, 2016. 7.2

[131] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*, 2018. 7.2

[132] Aljoša Ošep, Wolfgang Mehner, Paul Voigtlaender, and Bastian Leibe. Track, then decide: Category-agnostic vision-based multi-object tracking. In *ICRA*, pages 1–8. IEEE, 2018. 5.2

[133] Dennis Park, Deva Ramanan, and Charless Fowlkes. Multiresolution models for object detection. In *European conference on computer vision*, pages 241–254. Springer, 2010. 2.2

[134] AJ Patel. 2d visibility: wall tracking, 2012. 7.3.1

[135] Stéphane Petti and Thierry Fraichard. Safe motion planning in dynamic environments. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2210–2215. IEEE, 2005. 7.2

[136] Leonid Pishchulin, Mykhaylo Andriluka, Peter Gehler, and Bernt Schiele. Poselet conditioned pictorial structures. In *CVPR*, pages 588–595, 2013. **??**

[137] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989. 7.2

[138] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017. 1.1, 4.1, 4.2.1

[139] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *CVPR*, pages 918–927, 2018. 4.3.1, 5.2, 5.3.3

[140] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, pages 5099–5108, 2017. 4.2.1, 5.1, 5.2, 5.3.3

[141] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494. IEEE, 2009. 7.3.3

[142] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736, 2006. 7.2, 7.3.3, 7.3.4, 8, 8.1

[143] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 2.1, 2.2

[144] Shaoqing Ren, Kaiming He, Ross Girshick, Xiangyu Zhang, and Jian Sun. Object detection networks on convolutional feature maps. *arXiv preprint arXiv:1504.06066*,

2015. 2.2

[145] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. Deep imitative models for flexible inference, planning, and control. *arXiv preprint arXiv:1810.06544*, 2018. 7.2

[146] Thomas Roddick and Roberto Cipolla. Predicting semantic map representations from images using pyramid occupancy networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11138–11147, 2020. 7.2

[147] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011. 7.2

[148] Radu Bogdan Rusu. Semantic 3d object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, 24(4):345–348, 2010. 5.1, 5.3.4

[149] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *IROS*, pages 2155–2162. IEEE, 2010. 4.2.1

[150] Abbas Sadat, Sergio Casas, Mengye Ren, Xinyu Wu, Pranaab Dhawan, and Raquel Urtasun. Perceive, predict, and plan: Safe motion planning through interpretable semantic representations. *ECCV*, 2020. 7.2, 8.1

[151] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *AISTATS*, pages 448–455, 2009. 3.2

[152] Ben Sapp and Ben Taskar. Modec: Multimodal decomposable models for human pose estimation. In *CVPR*, 2013. **??**

[153] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: a core-set approach. In *ICLR*, 2017. 6.3.2

[154] Young-Woo Seo, Nathan Ratliff, and Chris Urmson. Self-supervised aerial images analysis for extracting parking lot structure. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009. 7.2

[155] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52 (55-66):11, 2010. 6.4

[156] Burr Settles. From theories to queries: Active learning in practice. In *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, pages 1–18, 2011. 6.1

[157] Burr Settles, Mark Craven, and Lewis Friedland. Active learning with real annotation costs. In *NeurIPS*, 2008. 6.4

[158] Yanyao Shen, Hyokun Yun, Zachary C. Lipton, Yakov Kronrod, and Animashree Anandkumar. Deep active learning for named entity recognition. In *ICLR*, 2018. 6.3, 6.4

[159] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Depart-*

mental Papers (CIS), page 107, 2000. 5.2

[160] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *CVPR*, pages 770–779, 2019. 4.2.2, 5.2, 5.4.1

[161] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. *arXiv preprint arXiv:1604.03540*, 2016. 2.4.1

[162] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds. In *ECCV*, pages 197–209. Springer, 2018. 4.2.2, 4.3

[163] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 3.1, 3.4, 3.3, 3.3, 3.4

[164] Nicholas D Socci, Daniel D Lee, and H Sebastian Seung. The rectified gaussian distribution. *NIPS*, pages 350–356, 1998. 3.1, 3.2, 3.2, 3.2

[165] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, pages 129–136, 2011. 3.2

[166] Boris Sofman, Ellie Lin, J Andrew Bagnell, John Cole, Nicolas Vandapel, and Anthony Stentz. Improving robot navigation through self-supervised online learning. *Journal of Field Robotics*, 23(11-12):1059–1075, 2006. 7.2

[167] Anthony Stentz. Optimal and efficient path planning for partially known environments. In *Intelligent Unmanned Ground Vehicles*, pages 203–220. Springer, 1997. 7.2

[168] Anthony Stentz et al. The focussed d$\hat{}$* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995. 7.2

[169] Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *AISTATS*, 2011. 3.2

[170] Johannes Strom, Andrew Richardson, and Edwin Olson. Graph-based segmentation for colored 3d laser point clouds. In *IROS*, pages 2131–2136. IEEE, 2010. 5.1

[171] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3476–3483. IEEE, 2013. 3.3

[172] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*. IEEE, 2015. 3.1

[173] Alex Teichman, Jesse Levinson, and Sebastian Thrun. Towards 3d object recognition via classification of arbitrary object tracks. In *ICRA*, pages 4034–4041. IEEE, 2011. 5.2

[174] Sebastian Thrun and Arno Bücken. Integrating grid-based and topological maps

for mobile robot navigation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 944–951, 1996. 4.1

[175] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005. 4.3.2

[176] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler. Efficient object localization using convolutional networks. *arXiv preprint arXiv:1411.4280*, 2014. 3.3

[177] Jonathan Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *Advances in Neural Information Processing Systems*, pages 1799–1807, 2014. 3.3

[178] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler. Efficient object localization using convolutional networks. In *CVPR*, 2015. 3.4, ??

[179] Antonio Torralba, Kevin P Murphy, William T Freeman, and Mark A Rubin. Context-based vision system for place and object recognition. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 273–280. IEEE, 2003. 2.1

[180] Shubham Tulsiani and Jitendra Malik. Viewpoints and keypoints. *arXiv preprint arXiv:1411.6067*, 2014. 3.3

[181] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008. 5.1, 7.1

[182] Mustafa Gokhan Uzunbas, Chao Chen, and Dimitris Metaxas. An efficient conditional random field approach for automatic and interactive neuron segmentation. *Medical image analysis*, 2016. 5.2, 5.3

[183] Jur P Van Den Berg and Mark H Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897, 2005. 7.2

[184] Rufin VanRullen and Simon J Thorpe. Is it a bird? is it a plane? ultra-rapid visual categorisation of natural and artifactual objects. *Perception-London*, 30(6):655–668, 2001. 3.1

[185] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR*, 11:3371–3408, 2010. 3.2

[186] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang, and Liang Lin. Cost-effective active learning for deep image classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 2016. 6.4

[187] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM TOG*, 38 (5):146, 2019. 4.2.1, 5.3.3

[188] Xinshuo Weng and Kris Kitani. A baseline for 3d multi-object tracking. *arXiv*,

abs/1907.03961, 2019. 5.2

[189] Lior Wolf and Stanley Bileschi. A critical view of context. *International Journal of Computer Vision*, 69(2):251–261, 2006. 2.1

[190] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, pages 9621–9630, 2019. 4.2.1, 5.3.3

[191] Yue Wu and Qiang Ji. Robust facial landmark detection under significant head poses and occlusion. In *ICCV*, 2015. ??, 3.10

[192] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015. 7.2

[193] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 4.2.2, 4.3, 4.3.1, 4.3.2, 5.2, 5.4.1

[194] Bin Yang, Junjie Yan, Zhen Lei, and Stan Z Li. Aggregate channel features for multi-view face detection. In *Biometrics (IJCB), 2014 IEEE International Joint Conference on*, pages 1–8. IEEE, 2014. ??

[195] Bin Yang, Ming Liang, and Raquel Urtasun. Hdnet: Exploiting hd maps for 3d object detection. In *CoRL*, pages 146–155, 2018. 4.2.2, 4.3

[196] Heng Yang, Xuming He, Xuhui Jia, and Ioannis Patras. Robust face alignment under occlusion via regional predictive power estimation. *IEEE Trans on Image Processing*, 2015. ??

[197] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. From facial parts responses to face detection: A deep learning approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3676–3684, 2015. ??, ??

[198] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *Proceedings of the IEEE International Conference on Computer Vision*, 2016. ??

[199] Yi Yang and Deva Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *CVPR*, pages 1385–1392. IEEE, 2011. ??

[200] Theodore C Yapo, Charles V Stewart, and Richard J Radke. A probabilistic representation of lidar range data for efficient 3d object detection. In *CVPR Workshops*, pages 1–8. IEEE, 2008. 4.1, 4.2.1, 4.2.2

[201] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Robert Fergus. Deconvolutional networks. In *CVPR*, 2010. 3.2, 3.2

[202] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018. 7.2

[203] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8660–8669,

2019. 7.2, 7.3.3, 7.3.4, 7.3.4, 7.4, 7.4.3, 7.4, 8, 8.1, 8.1

[204] Wenyuan Zeng, Shenlong Wang, Renjie Liao, Yun Chen, Bin Yang, and Raquel Urtasun. Dsdnet: Deep structured self-driving network. *arXiv preprint arXiv:2008.06041*, 2020. 8.1

[205] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multi-task cascaded convolutional networks. *arXiv preprint arXiv:1604.02878*, 2016. **??**

[206] Ye Zhang, Matthew Lease, and Byron C Wallace. Active discriminative text representation learning. In *AAAI*, 2017. 6.4

[207] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *Computer Vision–ECCV 2014*, pages 94–108. Springer, 2014. 3.3

[208] Qian-Yi Zhou and Ulrich Neumann. A streaming framework for seamless building reconstruction from large-scale aerial lidar data. In *CVPR*, pages 2759–2766. IEEE, 2009. 5.1

[209] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, pages 4490–4499, 2018. 4.1, 4.2.2, 4.3, 5.2

[210] Chenchen Zhu, Yutong Zheng, Khoa Luu, and Marios Savvides. Cms-rcnn: Contextual multi-scale region-based cnn for unconstrained face detection. *arXiv preprint arXiv:1606.05413*, 2016. 2.2, **??**

[211] Long Leo Zhu, Yuanhao Chen, and Alan Yuille. Recursive compositional models for vision: Description and review of recent work. *Journal of Mathematical Imaging and Vision*, 41(1-2):122–146, 2011. 3.1

[212] Xiangxin Zhu and Deva Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *CVPR*, 2012. 3.7

[213] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008. 7.2