Amodal Visual Scene Representations With and Without Geometry

Adam W. Harley

CMU-RI-TR-22-13

May 2022

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

Most computer vision models in deployment today describe the pixels of images. This does not suffice, because images are only projections of the scene in front of the camera. In this thesis we build representations that attempt to describe the scene itself. We call these representations "amodal" (i.e., without modality), emphasizing the fact that they describe elements of the scene for which we have no sensory input.

We present two methods for amodal visual scene representation. The first focuses on modelling space, and proposes geometry-based methods for lifting images into 3D maps, where the objects are complete, despite partial occlusions in the imagery. We show that this representation allows for self-supervised learning from multi-view data, and yields state-of-the-art results as a perception system for autonomous vehicles, where the goal is to estimate a "bird's eye view" semantic map from multiple sensors. The second method focuses on modelling time, and proposes geometry-free methods for tracking image elements through partial and full occlusions across a video. Using learned temporal priors and within-inference optimization, we show that our model can track points across occlusions, and outperform flow-based and feature-matching methods on fine-grained multi-frame correspondence tasks.

Acknowledgments

Thank you to my advisor, Katerina Fragkiadaki. At the beginning of my PhD, I asked you to push me to be better, and here at the end, I can easily say that you did. You welcomed me into your fast-paced world. It felt like everything we want is just around the corner, and we need to get there fast, or else we won't be the first. You have such a unique energy for research, I doubt I will find it again elsewhere. Thank you for sharing your time and energy with me.

Thank you to Chris Atkeson. You showed me how to question everything, and convinced me that it is often a good option to toss it all out, and restart from scratch. In personal meetings and also when speaking publicly, you behaved the way I thought only fictional characters do: you always had a joke, you spoke in perfect paragraphs, you won any argument you wanted to, and even when pointing out critical flaws in someone's proudest work, you made everyone comfortable enough to laugh.

Thank you to Deva Ramanan, for setting a high standard of work, with crystal-clear motivations and sensible methods, and for relaxed discussions about in-progress efforts. Thank you to Andrew Zisserman, for giving me a clear goal of making things work in real data, and for recognizing when I made progress toward that goal.

Thank you to Leo Keselman, for our frequent visits to each other's whiteboard, discussing partial results, and commiserating over related and unrelated work. Thank you to Ankit Bhatia, Brian Okorn, and Ben Newman, for your care, and for pulling me closer to a reasonable center, too many times to count.

I am grateful to so many remarkable people in my PhD cohort, especially Peiyun Hu, Rogerio Bonatti, Anahita Mohseni Kabir, Rohit Girdhar, Adithya Murali, Lerrel Pinto, and Alex Spitzer. I also want to acknowledge Xiaolong Wang, Ishan Misra, Achal Dave, Kenny Marino, Jacob Walker, Aayush Bansal, and Nick Rhinehart, who provided sage advice despite being multiple years separated in our RI PhDs.

Thank you to Aljosa Osep and Jonathon Luiten, for our Friday coffees and rambling discussions on the research we like and dislike. Thank you to Shubham Tulsiani, Kris Kitani, and Dave Held, for being so accessible, and for providing thoughtful answers to my scattered questions about life and career.

Thank you also to Shih-En Wei, Jason Saragih, Chaitanya Ahuja, Shugao Ma, Tomas Simon, Stephen Lombardi, Iain Matthews, Yaser Sheikh, and everyone else I had the pleasure of meeting through my summer at the Meta Reality Labs in Pittsburgh. That experience was a memorable peek at “the good life”.

Thank you to Kosta Derpanis. Without your early guidance, I would likely have landed in a different field. Without your unwavering support, my PhD roller-coaster would have been a lot less fun.

Through Katerina’s lab, I collaborated with so many wonderful people, and I am sure we will continue. Thank you to Arpit Agarwal, Ayush Jain, Ayush Mangal, Darshan Patil, N. Dinesh Reddy, Fangyu Li, Fish Tung, Gabe Sarch, Gaurav Pathak, Jing Wen, Jingyun Yang, Mihir Prabhudesai, Paul Schydlo, Ricson Cheng, Ritwick Chaudhry, Rakesh R. Menon, Shamit Lal, Shrinidhi Lakshmikanth, Shubhankar Potdar, Wen-Hsuan Chu, Xian Zhou, Yiming Zuo, Yunchu Zhang, and Zhaoyuan Fang. We didn’t turn every idea into a paper, but we certainly tried.

Finally, thank you to my parents, and my three big brothers. The time away from you is sometimes difficult to bear. Thank you for your support, your love, and your patience.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Computer vision has made tremendous progress in describing the pixels of images. Every image, however, is only a projection of the scene in front of the camera, and these projections bring nuisance artifacts, such as occlusions and perspective distortions. By featurizing pixels, we are allowing our visual representations to reflect all of those same artifacts. In this thesis, we argue that we should featurize the scene itself, not just the pixels it projects to. Scene representations are unconstrained by what the pixels happen to show frame-by-frame, and can leverage spatial and temporal priors that are true in "scene space" but not true in "pixel space", such as the fact that objects have complete 3D shapes, and the fact that things persist over time. We call our scene representations "amodal" (i.e., without modality), since they describe elements of the scene for which we have no sensory input. We are also motivated here by the human ability of "amodal completion" [107], which is the perception of whole objects when our sensory input only shows parts. For example, when we see a cat behind a picket fence, we perceive a whole cat, rather than just the visible parts. Computer vision models with this ability can be expected to detect and track objects more effectively, and be less sensitive to occlusions.

In this thesis, we present two methods for amodal visual scene representation. The first focuses on modelling space, and proposes geometry-based methods for lifting images into 3D maps, where the objects are complete, despite partial occlusions in the imagery. We show that this representation allows for self-supervised learning from multi-view data, and yields state-of-the-art results as a perception system for autonomous vehicles, where the goal is to estimate a "bird's eye view" semantic map from multiple sensors. The second method focuses on modelling time, and proposes geometry-free methods for tracking image elements through partial and full occlusions across a video. Using learned temporal priors and within-inference optimization, we show that our model can track points across occlusions, and outperform flow-based and feature-matching methods on fine-grained multi-frame correspondence tasks.

Besides trading focus on space vs. time, and geometry-based vs. geometry-free models, our two broad approaches can be distinguished at a fundamental level using terminology from fluid mechanics, as Eulerian vs. Lagrangian [184]. In the Eulerian (geometry-based) approach, we estimate features at fixed stations in a map. In the Lagrangian (geometry-free) approach, we follow particles, and each particle has its own travelling coordinate system. As might be expected, the two approaches have different strengths, and we expect future work will combine or at least alternate between them, depending on the requirements of the task at hand.

1.2 Geometry-Based Amodal Scene Representations

The majority of the thesis uses geometry-based models to build amodal representations of the scene. By this, we mean: models that take input from one or more sensors, and leverage the geometry of the sensor setup to produce a representation of the metric 3D space depicted in the observations.

In Chapter 2, we develop a method for learning amodal 3D scene representations, using multi-view geometry to drive a self-supervised loss. We motivate our approach with predictive coding theories [41, 129], which suggest that the brain learns by predicting observations at various levels of abstraction. Given images and pointclouds obtained from a static scene, we ask the model to predict how the scene would look from a held-out viewpoint. We then reveal the new viewpoint, and compare the revealed features to the predicted features. If the model does not correctly predict the features of the held-out view, it receives a signal to update its parameters. Note that this task does not require any human-provided annotations; the training signal is freely available to a mobile agent in a 3D world who can estimate its egomotion [117]. In our experiments, we explore the link between view-predictive learning and the emergence of amodal 3D perception. To the best of our knowledge, this is the first work that empirically shows view prediction to be a scalable self-supervised task beneficial to 3D object detection.

In Chapter 3, we build on amodal prediction across views, to obtain correspondences across time. We hypothesize that appearance-based correspondences learned in static scenes will also work well in dynamic scenes. This is motivated by the fact that the content of dynamic scenes is the same as the content of static scenes—cars are cars, whether they are speeding down the highway or sitting in a parking lot. We couple our geometry-based feature learning approach with a simple tracking pipeline, and show that the learned visual feature representations can accurately track moving objects in 3D, in challenging environments with a moving camera.

A computational challenge in these 3D feature learning efforts is *resolution*. Most of the works in this thesis use a voxel-based representation of 3D space, whose computational expense grows cubically with resolution. We explore this issue in Chapter 4, by switching to a high-fidelity continuous 3D representation, allowing the model to produce sub-voxel features at arbitrary resolutions. This change improves both detection and tracking accuracy. In this chapter we also investigate our method's ability to correspond parts across objects, and observe that our accuracy here is still low. These results reveal new challenges for self-supervised correspondence learning.

A critical assumption in the work thus far is that we can actually *obtain* multi-view correspondence in scenes, using knowledge of egomotion, which then allows us to learn amodal completion and feature correspondence. In Chapter 5 we give up this assumption, and learn from single-view observations. We present a model that takes a real pointcloud as input and produces a densified pointcloud as output, and we show that this leads to improved detection performance. This fits with our intuition: models that understand *complete* 3D scenes from their input—which is a pre-requisite for view prediction—do better than models which only perceive the scene surface.

Finally, Chapter 6 deploys our 3D scene representation on a competitive task: "bird's eye view" semantic mapping for autonomous vehicles. With only minor modifications to the model from Chapter 2, we obtain a model which matches the accuracy of current published systems, which are specialized for the task. We further show that if we incorporate the available data from radar, which is typically ignored, we exceed the state-of-the-art.

1.3 Geometry-Free Amodal Scene Representations

The second part of the thesis uses geometry-free models to build amodal representations of the scene. While the first part focused on amodal completion of 3D shapes and objects (i.e., space), this part focuses on amodal completion of trajectories (i.e., time). Here we also shift from an Eulerian approach to a Lagrangian approach: Instead of representing a video by updating the per-timestep features at fixed 3D locations, we focus on individual particles, and update their positions as we follow them across time.

To build our amodal temporal representation of scenes, in Chapter 7 we revisit Sand and Teller's "particle video" approach [141]. This approach assumes a permanence about scene particles, which is typically reserved for objects. We argue that particles (i.e., parts of objects, and sub-part points), should be tracked across occlusions and appearance changes, using temporal priors. Typically, occlusion-resistant object tracking relies on category-specific detectors (e.g., tracking people relies on per-frame person detection [23]), but detectors are unavailable for arbitrary particles. We therefore implement our amodal "particle" tracker using components that drive the current state-of-the-art in optical flow, such as dense cost maps and iterative optimization. We train our models using long-range amodal point trajectories mined from existing optical flow datasets that we synthetically augment with occlusions. We test our approach in trajectory estimation benchmarks and in keypoint label propagation tasks, and it compares favorably against state-of-the-art optical flow and feature tracking methods.

1.4 Overview

The thesis is split into two parts, corresponding to the two approaches just described. Most of the chapters within the parts are standalone papers. We provide the publication references here, partly as outline, but more importantly to give credit to all authors:

1. **Geometry-Based Amodal Scene Representations**
    - Chapter 2: Learning from Unlabelled Videos with Contrastive Predictive Neural 3D Mapping [54] (ICLR 2020)
    - Chapter 3: Tracking Emerges by Looking Around Static Scenes with Neural 3D Mapping [55] (ECCV 2020)
    - Chapter 4: CoCoNets: Continuous Contrastive 3D Scene Representations [84] (CVPR 2021)
    - Chapter 5: Self-Supervised Pointcloud Completion Improves Object Detection (unpublished)
    - Chapter 6: A Baseline for Neural 3D Mapping, Without Depth or Egomotion (unpublished)
2. **Geometry-Free Amodal Scene Representations**
    - Chapter 7: Particle Videos Revisited: Tracking Through Occlusions Using Point Trajectories [56] (arXiv)

Finally, Chapter 8 provides a summary, and suggests future work.

Part I

Geometry-based amodal scene representations

Chapter 2

Learning from Unlabelled Videos with Contrastive Predictive Neural 3D Mapping

2.1 Introduction

Predictive coding theories [41, 129] suggest that the brain learns by predicting observations at various levels of abstraction. These theories currently have extensive empirical support: stimuli are processed more quickly if they are predictable [101, 121], prediction error is reflected in increased neural activity [7, 129], and disproven expectations lead to learning [145]. A basic prediction task is view prediction: from one viewpoint, predict what the scene would look like from another viewpoint. Learning this task does not require supervision from any annotations; supervision is freely available to a mobile agent in a 3D world who can estimate its egomotion [117]. Humans excel at this task: we can effortlessly imagine plausible hypotheses for the occluded side of objects in a photograph, or guess what we would see if we walked around our office desks. Our ability to *imagine* information missing from the current image view—and necessary for predicting alternative views—is tightly coupled with visual perception. We infer a mental representation of the world that is 3-dimensional, in which the objects are distinct, have 3D extent, occlude one another, and so on. Despite our 2-dimensional visual input, and despite never having been supplied a bounding box or segmentation mask as supervision, our ability for 3D perception emerges early in infancy [153, 154].

In this chapter, we explore the link between view-predictive learning and the emergence of 3D perception in computational models of perception, on mobile agents in static and dynamic scenes. Our models are trained to predict views of static scenes given 2.5D (color and depth; RGB-D) video streams as input, and are evaluated on

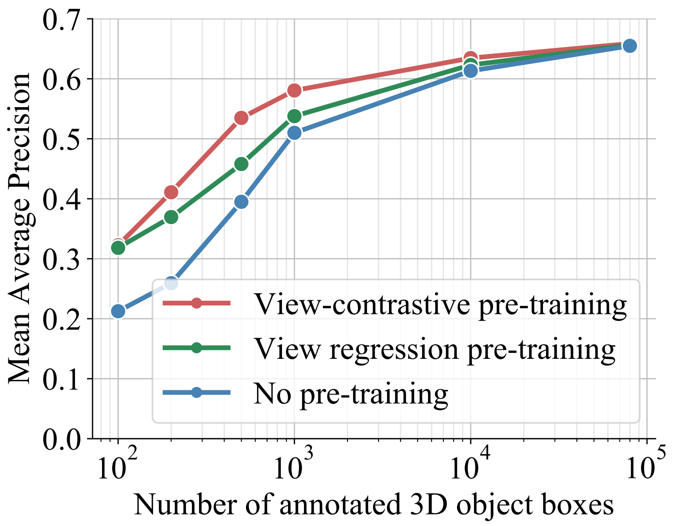

Figure 2.1: **Semi-supervised 3D object detection.** Pre-training with view-contrastive prediction improves results, especially when there are few 3D bounding box annotations.

their ability to detect objects in 3D. Our models map the 2.5D input streams into 3D feature volumes of the depicted scene. At every frame, the architecture estimates and accounts for the motion of the camera, so that the internal 3D representation remains stable under egomotion. The model projects its inferred 3D feature maps to novel viewpoints, and matches them against visual representations extracted from the target view. We propose contrastive losses to measure the match error, and backpropagate gradients end-to-end in our differentiable modular architecture. At test time, our model forms plausible 3D completions of the scene given RGB-D video streams or even a *single RGB-D image* as input: it learns to fill in information behind occlusions, and infer the 3D extents of objects.

We evaluate the trained 3D representations in two tasks. **(1)** Semi-supervised learning of 3D object detectors (Figure 2.1): We show that view-contrastive pre-training helps detect objects in 3D, especially in the low-annotations regime. **(2)** Unsupervised 3D moving object detection (Figure 2.3-right): Our model can detect moving objects in 3D without any human annotations, by forming a 3D feature volume for each timestep, then estimating the motion field between volumes, and clustering the motion into objects.

We have the following contributions over prior works:

1. **Novel view-contrastive prediction objectives**: We show that our contrastive losses outperform RGB regression [34, 171] and variational auto-encoder (VAE) alternatives [36] in semi-supervised 3D object detection.

2. **A novel unsupervised 3D moving object detection method**: By estimating 3D motion inside egomotion-stabilized 3D feature maps, we can detect moving objects unsupervised, outperforming 2.5D baselines and iterative generative what-where VAEs of previous works [63, 81].
3. **Simulation-to-real transfer of the acquired 3D feature representations**: We show that view-contrastive pre-training in simulation boosts the performance of 3D object detection in real data.

2.2 Related work

Predictive coding Predictive coding theories suggest that much of the learning in the brain is of a predictive nature [41, 129]. Recent work in unsupervised learning of word representations has successfully used ideas of predictive coding to learn word representations by predicting neighboring words [104]. Many challenges emerge in going from a finite-word vocabulary to the continuous high-dimensional image data manifold. Unimodal losses such as mean squared error are not very useful when predicting high dimensional data, due to the stochasticity of the output space. Researchers have tried to handle such stochasticity using latent variable models [94] or autoregressive prediction of the output pixel space, which involves sampling each pixel value from a categorical distribution conditioned on the output thus far [173]. Another option is to make predictions in a feature space which is less stochastic than the input. Recently, [113] followed this direction and used an objective that preserves the mutual information between "top-down" contextual features predicted from input observations, and "bottom-up" features produced from future observations; it applied this objective in speech, text, and image crops. The view-contrastive loss proposed in this work is a non-probabilistic version of their contrastive objective. However, our work focuses on the video domain as opposed to image patches, and uses drastically different architectures for both the top-down and bottom-up representations, involving a 3D egomotion-stabilized bottleneck.

View prediction View prediction has been the center of much recent research effort. Most methods test their models in single-object scenes, and aim to generate beautiful images for graphics applications [74, 139, 147, 161], as opposed to learning general-purpose visual representations. In this work, we use view prediction to help object detection, not the inverse. Eslami et al. [36] attempted view prediction in full scenes, yet only experimented with toy data containing a few colored 3D blocks. Their model cannot effectively generalize beyond the training distribution, e.g., cannot generalize across scenes of variable numbers of objects. The work of Tung et al. [171] is the closest to our work. Their model is also an inverse graphics network equipped with a 3-dimensional feature bottleneck, trained for view prediction; it showed strong generalization across scenes, number of objects, and arrangements. However, the authors demonstrated its abilities only in toy simulated scenes, similar to those used by Eslami et al. [36]. Furthermore, they did not evaluate the usefulness of the learned features for a downstream semantic task (beyond view prediction). This raises questions on the scalability and usefulness of view prediction as an objective for self-supervised visual representation learning, which our work aims to address. We compare against the state-of-the-art model of Tung et al. [171] and show that the features learned under our proposed view-contrastive losses are more semantically meaningful (Figure 2.1). To the best of our knowledge, this is the first work that can discover objects in 3D from a single camera viewpoint, without any human annotations of object boxes or masks.

3D feature representations A long-standing debate in Computer Vision is whether it is worth pursuing 3D models in the form of binary voxel grids, meshes, or 3D pointclouds as the output of visual recognition. The "blocks world" of [134] set as its goal to reconstruct the 3D scene depicted in the image in terms of 3D solids found in a database. Pointing out that replicating the 3D world in one's head is not enough to actually make decisions, [8] argued for feature-based representations, as done by recent work in end-to-end deep learning [87]. Our work proposes *learning-based 3D feature representations* in place of previous human-engineered ones, attempting to reconcile the two sides of the debate.

Some recent works have attempted various forms of map-building [52, 60] as a form of geometrically-consistent temporal integration of visual information, in place of geometry-unaware long short-term memory (LSTM; 61) or gated recurrent unit (GRU; 22) models. The closest to ours are Learned Stereo Machines (LSMs; 74), DeepVoxels [147], and Geometry-aware Recurrent Neural Nets (GRNNs; 171), which integrate images sampled from a viewing sphere into a latent 3D feature memory tensor, in an egomotion-stabilized manner, and predict views. All of these works consider very simplistic non-photorealistic environments and 2-degree-of-freedom cameras (i.e., with the camera locked to a pre-defined viewing sphere). None of these works evaluate the suitability of their learned features for a downstream task. Rather, their main objective is to accurately predict views.

Motion estimation and object segmentation Most motion segmentation methods operate in 2D image space, and use 2D optical flow to segment moving objects. While earlier approaches attempted motion segmentation completely unsupervised, by exploiting motion trajectories and integrating motion information over time [9, 111], recent works focus on learning to segment objects in videos, supervised by annotated video benchmarks [39, 78]. Our work differs in the fact that we address object detection and segmentation in 3D as opposed to 2D, by estimating 3D motion of the "imagined" (complete) scene, as opposed to 2D motion of the observed scene.

2.3 Contrastive Predictive Neural 3D Mapping

We consider a mobile agent that can move about the scene at will. The agent has a color camera with known intrinsics, and a depth sensor registered to the camera's coordinate frame. At training time, the agent has access to its camera pose, and it learns in this stage to imagine full 3D scenes (via view prediction), and to estimate egomotion (from ground-truth poses). It is reasonable to assume that a mobile agent who moves at will has access to its approximate egomotion, since it chooses where to move and what to look at [117]. Active vision is outside the scope of this work, so our agent simply chooses viewpoints randomly. At test time, the model estimates egomotion on-the-fly from its RGB-D inputs. We use groundtruth depth provided by the simulation environment, and we will show in Sec. 2.4 that the learned models generalize to the real world, where (sparser) depth is provided by a LiDAR unit. We describe our model architecture in Sec. 2.3.1, our view-contrastive prediction objectives in Sec. 2.3.2, and our unsupervised 3D object segmentation method in Sec. 2.3.3.

2.3.1 Neural 3D Mapping

Our model's architecture is illustrated in Figure 2.2-left. It is a recurrent neural network (RNN) with a memory state tensor $\mathcal{M}^{(t)} \in \mathbb{R}^{w \times h \times d \times c}$, which has three spatial dimensions (width w, height h, and depth d) and a feature dimension (c channels per grid location). The latent state aims to capture an informative and geometrically-consistent 3D deep feature map of the world space. Therefore, the spatial extents correspond to a large cuboid of world space, defined with respect to the camera's position at the first timestep. We refer to the latent state as the model's *imagination* to emphasize that most of the grid locations in $\mathcal{M}^{(t)}$ will not be observed by any sensor, and so the feature content must be "imagined" by the model.

Our model is made up of differentiable modules that go back and forth between 3D feature imagination space and 2D image space. It builds on the recently proposed geometry-aware recurrent neural networks (GRNNs) of [171], which also have a 3D egomotion-stabilized latent space, and are trained for RGB prediction. Our model can be considered a type of GRNN. In comparison to [171]:

- Our egomotion module can handle general camera motion, as opposed to a 2-degree-of-freedom sphere-locked camera. This is a critical requirement for handling data that comes from freely-moving cameras, such as those mounted on mobile vehicles, as opposed to only orbiting cameras.
- Our 3D-to-2D projection module decodes the 3D map into 2D feature maps, as opposed to RGB images, and uses view-contrastive prediction as the objective, as opposed to regression.

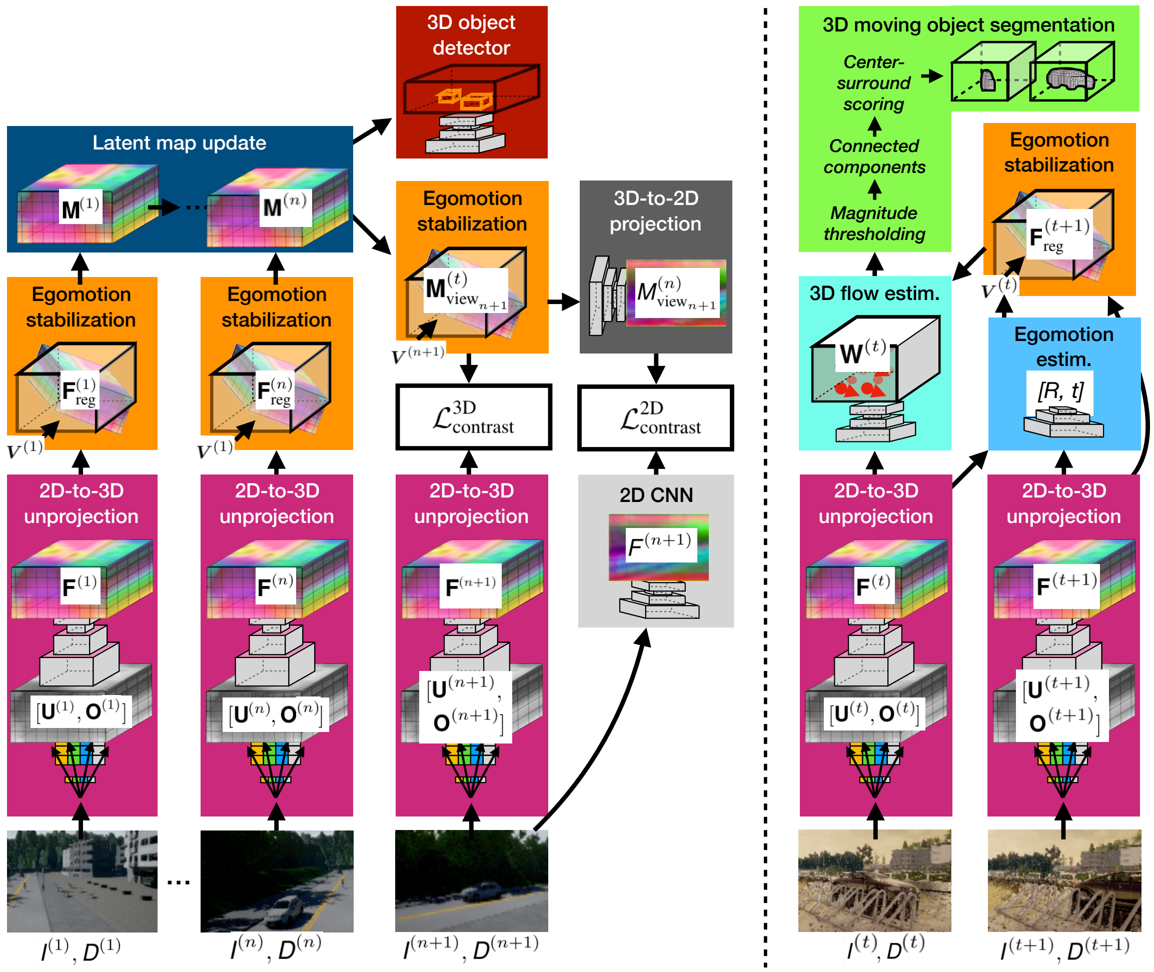

Figure 2.2: **Contrastive predictive neural 3D mapping.** *Left:* Learning visual feature representations by moving in static scenes. The neural 3D mapper learns to lift 2.5D video streams to egomotion-stabilized 3D feature maps of the scene by optimizing for view-contrastive prediction. *Right:* Learning to segment 3D moving objects by watching them move. Non-zero 3D motion in the egomotion-stabilized 3D feature space reveals independently moving objects and their 3D extent, without any human annotations.

We briefly describe each neural module of our architecture next. Implementation details are in the appendix.

2D-to-3D unprojection This module converts the input RGB image $I^{(t)} \in \mathbb{R}^{w\times h\times 3}$ and pointcloud $D^{(t)} \in \mathbb{R}^{n\times 3}$ into 3D tensors. The RGB is "unprojected" into a 3D tensor $\mathcal{U}^{(t)} \in \mathbb{R}^{w\times h\times d\times 3}$ by filling each 3D grid location with the RGB value of its corresponding subpixel. The pointcloud is converted to a 3D occupancy grid

$\mathcal{O}^{(t)} \in \mathbb{R}^{w \times h \times d \times 1}$, by assigning each voxel a value of 1 or 0, depending on whether or not a point lands in the voxel. We then convert the concatenation of these tensors into a 3D feature tensor $\mathcal{F}^{(t)} \in \mathbb{R}^{w \times h \times d \times c}$, via a 3D convolutional encoder-decoder network with skip connections. We L_2-normalize the feature in each grid cell.

Egomotion estimation This module computes the relative 3D rotation and translation between the current camera pose (at time t) and the reference pose (from time 1), allowing us to warp the feature tensor $\mathcal{F}^{(t)}$ into a registered version $\mathcal{F}^{(t)}_{\text{reg}}$. In principle any egomotion estimator could be used here, but we find that our 3D feature tensors are well-suited to a 3D coarse-to-fine alignment search, similar to the 2D process in the state-of-the-art optical flow model PWC-Net [156]. Given the 3D tensors of the two timesteps, $\mathcal{F}^{(1)}$ and $\mathcal{F}^{(t)}$, we incrementally warp $\mathcal{F}^{(t)}$ into alignment with $\mathcal{F}^{(1)}$, by estimating the approximate transformation at a coarse scale, then estimating residual transformations at finer scales. This is done efficiently with 6D cross correlations and cost volumes. Following PWC-Net, we use fully-connected layers to convert the cost volumes into motion estimates. While our neural architecture is trained end-to-end to optimize a view prediction objective, our egomotion module by exception is trained supervised using pairs of frames with annotated egomotion. In this way, it learns to be invariant to moving objects in the scene.

Latent map update This module aggregates egomotion-stabilized (registered) feature tensors into the memory tensor $\mathcal{M}^{(t)}$. On the first timestep, we set $\mathcal{M}^{(1)} = \mathcal{F}^{(1)}$. On later timesteps, we update the memory with a simple running average.

3D-to-2D projection This module "renders" the 3D feature state $\mathcal{M}^{(t)}$ into a 2D feature map of a desired viewpoint $\mathcal{V}^{(k)}$. We first warp the 3D feature map $\mathcal{M}^{(t)}$ into a view-aligned version $\mathcal{M}^{(t)}_{\text{view}_k}$, then map it to a 2D feature map $M^{(t)}_{\text{view}_k}$ with a 2-block 2D ResNet [57].

3D object detection Given images with annotated 3D object boxes, we train a 3D object detector that takes as input the 3D feature map $\mathcal{M}^{(t)}$, and outputs 3D bounding boxes for the objects present. Our object detector is a 3D adaptation of the state-of-the-art 2D object detector, Faster-RCNN [131]. The model outputs 3D axis-aligned boxes with objectness confidences.

2.3.2 Contrastive Predictive Training

Given a set of input RGB images $(I^{(0)}, \ldots, I^{(n)})$, pointclouds $(D^{(0)}, \ldots, D^{(n)})$, and camera poses $(V^{(0)}, \ldots, V^{(n)})$, we train our model to predict feature abstractions of an unseen input $(I^{(n+1)}, D^{(n+1)}, V^{(n+1)})$, as shown in Figure 2.2-left. We consider two types of representations for the target view: a top-down one, $\mathcal{T} = f[(I^{(0)}, D^{(0)}, V^{(0)}), \ldots, (I^{(n)}, D^{(n)}, V^{(n)}), V^{(n+1)}]$, where f is the top-down feature prediction process, and a bottom-up one, $\mathcal{B} = g[I^{(n+1)}, D^{(n+1)}]$, where g is the bottom-up feature extraction process. Note that the top-down representation has access to the viewpoint $V^{(n+1)}$ but not to observations from that viewpoint $(I^{(n+1)}, D^{(n+1)})$, while the bottom-up representation is only a function of those observations.

We construct 2D and 3D versions of these representation types, using our architecture modules:

- We obtain $\mathcal{T}^{3D} = \mathcal{M}^{(n)}$ by encoding the set of inputs from indices $1, \ldots, n$.
- We obtain $\mathcal{B}^{3D} = \mathcal{F}^{(n+1)}$ by encoding the single input from index $n+1$.
- We obtain $\mathcal{T}^{2D} = M^{(n)}_{\text{view}_{n+1}}$ by rendering $\mathcal{M}^{(n)}$ from viewpoint $V^{(n+1)}$.
- We obtain $\mathcal{B}^{2D} = F^{(n+1)}$ by convolving $I^{(n+1)}$ with a 3-block 2D ResNet [57].

Finally, the contrastive losses pull corresponding (top-down and bottom-up) features close together in embedding space, and push non-corresponding ones beyond a margin of distance:

$$\mathcal{L}^{2D}_{\text{contrast}} = \sum_{i,j,m,n} \max\left(\mathcal{Y}^{2D}_{ij,mn}(\|\mathcal{T}^{2D}_{ij} - \mathcal{B}^{2D}_{mn}\|_2^2 - \alpha), 0\right), \tag{2.1}$$

$$\mathcal{L}^{3D}_{\text{contrast}} = \sum_{i,j,k,m,n,o} \max\left(\mathcal{Y}^{3D}_{ijk,mno}(\|\mathcal{T}^{3D}_{ijk} - \mathcal{B}^{3D}_{mno}\|_2^2 - \alpha), 0\right), \tag{2.2}$$

where α is the margin size, and $\mathcal{Y}$ is 1 at indices where $\mathcal{T}$ corresponds to $\mathcal{B}$, and -1 everywhere else. The losses ask tensors depicting the same scene, but acquired from different viewpoints, to contain the same features. The performance of a metric learning loss depends heavily on the sampling strategy used [143, 151, 152]. We use the distance-weighted sampling strategy proposed by [189] which uniformly samples "easy" and "hard" negatives; we find this outperforms both random sampling and semi-hard sampling [143].

2.3.3 Unsupervised 3D Moving Object Detection

Upon training, our model learns to map even a *single* RGB-D input to a complete 3D imagination, as we show in Figure 2.2-right. Given two temporally consecutive and egomotion-stabilized 3D maps $\mathcal{F}^{(t)}, \mathcal{F}^{(t+1)}_{\text{reg}}$, predicted independently using inputs $(I^{(t)}, D^{(t)})$ and $(I^{(t+1)}, D^{(t+1)})$, we train a motion estimation module to predict the 3D motion field $\mathcal{W}^{(t)}$ between them, which we call 3D imagination flow. Since we have accounted for camera motion, this 3D motion field should only be non-zero for independently moving objects. We obtain 3D object proposals by clustering the 3D flow vectors, extending classic motion clustering methods [9, 111] to an egomotion-stabilized 3D feature space, as opposed to 2D pixel space.

Estimating 3D Imagination Flow

Our 3D "imagination flow" module is a 3D adaptation of the PWC-Net 2D optical flow model [156]. Note that our model only needs to estimate motion of the independently-moving part of the scene, since egomotion has been accounted for. It works by iterating across scales in a coarse-to-fine manner; at each scale, we compute a 3D cost volume, convert these costs to 3D displacement vectors, and incrementally warp the two tensors to align them. We use two self-supervised tasks:

1. Synthetic rigid transformations: We apply random rotations and translations to $\mathcal{F}^{(t)}$ and ask the model to recover the dense 3D flow field that corresponds to the transformation.
2. Unsupervised 3D temporal feature matching:

$$\mathcal{L}_{\text{warp}} = \sum_{i,j,k} ||\mathcal{F}^{(t)}_{i,j,k} - \text{warp}(\mathcal{F}^{(t+1)}_{\text{reg}}, \mathcal{W}^{(t)})_{i,j,k}||^2_2, \tag{2.3}$$

   where "warp" is an operation that back-warps $\mathcal{F}^{(t+1)}_{\text{reg}}$ to align it with $\mathcal{F}^{(t)}$, using the estimated flow $\mathcal{W}^{(t)}$. We apply the warp with a differentiable 3D spatial transformer layer, which does trilinear interpolation to resample each voxel. This extends self-supervised 2D optical flow [203] to 3D feature constancy (instead of 2D brightness constancy).

We found that both types of supervision are essential for obtaining accurate 3D flow field estimates. Since we are not interested in the 3D motion of empty air voxels, we additionally estimate 3D voxel occupancy, and supervise this using the input pointclouds. We describe our 3D occupancy estimation in more detail in the appendix. At test time, we set the 3D motion of all unoccupied voxels to zero.

The proposed 3D imagination flow enjoys significant benefits over 2D optical flow or 3D scene flow. It does not suffer from occlusions and dis-occlusions of image content or projection artifacts [155], which typically transform rigid 3D transformations into non-rigid 2D flow fields. In comparison to 3D scene flow [62], which concerns visible 3D points, 3D imagination flow is computed between visual features that may never have appeared in the field of view, but are rather filled in by the model (i.e., "imagined").

3D Moving Object Segmentation

We obtain 3D object segmentation proposals by thresholding the 3D imagination flow magnitude, and clustering voxels using connected components. We score each component using a 3D version of a center-surround motion saliency score employed by numerous works for 2D motion saliency detection [42, 98]. This score is high when the 3D box interior has lots of motion but the surrounding shell does not. This results in a set of scored 3D segmentation proposals for each video scene.

2.4 Experiments

We train our models in CARLA [34], an open-source photorealistic simulator of urban driving scenes, which permits moving the camera to any desired viewpoint in the scene. We obtain data from the simulator as follows. We generate 1170 autopilot episodes of 50 frames each (at 30 FPS), spanning all weather conditions and all locations in both "towns" in the simulator. We define 36 viewpoints placed regularly along a 20m-radius hemisphere in front of the ego-car. This hemisphere is anchored to the ego-car (i.e., it moves with the car). In each episode, we sample 6 random viewpoints from the 36 and randomly perturb their pose, and then capture each timestep of the episode from these 6 viewpoints. We generate train/test examples from this, by assembling all combinations of viewpoints (e.g., $N \leq 5$ viewpoints as input, and 1 unseen viewpoint as the target). We filter out frames that have zero objects within the metric "in bounds" region of the GRNN ($32m \times 32m \times 4m$). This yields 172524 frames (each with multiple views): 124256 in Town1, and 48268 in Town2. We treat the Town1 data as the "training" set, and the Town2 data as the "test" set, so there is no overlap between the train and test images.

For additional testing with real-world data, we use the (single-view) object detection benchmark from the KITTI dataset [43], with the official train/val split: 3712 training frames, and 3769 validation frames.

We evaluate our view-contrastive 3D feature representations in three tasks: **(1)** semi-supervised 3D object detection, **(2)** unsupervised 3D moving object detection, and **(3)** 3D motion estimation. We use [171] as a baseline representing the state-of-the-art, but evaluate additional related works in the appendix.

2.4.1 Semi-Supervised Learning of 3D Object Detection

We use the proposed view-contrastive prediction as pre-training for 3D object detection. Prior work has evaluated self-supervised 2D deep features in this way, by re-purposing them for a 2D detection task Doersch et al. [32]. We pretrain the inverse graphics network weights, and then train a 3D object detector module supervised to map a 3D feature volume $\mathcal{M}$ to 3D object boxes, as described in Section 2.3.1. We are interested in seeing the benefit of this pre-training across different amounts of label supervision, so we first use the full CARLA train set for view prediction training (without using box labels), and then use a randomly-sampled subset of the CARLA train set for box supervision; we evaluate on the CARLA validation set. We varied the size of the box supervision subset across the following range: 100, 200, 500, 1000, 10000, 80000. We show mean average precision (at an IoU of 0.75) for car detection as a function of the number of annotated 3D bounding box examples in Figure 2.1. We compare our model against a version of our model that optimizes

Method	**mAP@IOU**		
	0.33	0.50	0.75
No pre-training (random init.)	.59	.52	.17
View regression pret., frozen	.64	.54	.15
View regression pret., finetuned	.65	.55	.18
View-contrastive pret., frozen	.67	.58	.15
View-contrastive pret., finetuned	**.70**	**.60**	**.19**

Table 2.1: **CARLA-to-KITTI feature transfer**. We train a 3D detector module on top of the inferred 3D feature maps $\mathcal{M}$ using KITTI 3D object box annotations

for RGB regression, similar to [171] but with a 6 DoF camera motion as opposed to 2 DoF, as well as a model trained from random weight initialization (i.e., without pre-training). After pre-training, we *freeze* the feature layers after view predictive learning, and only supervise the detector module; for the fully supervised baseline (from random initialization), we train end-to-end.

As expected, the supervised model performs better with more labelled data. In the low-data regime, pre-training greatly improves results, and more so for view-contrastive learning than RGB learning. We could not compare against alternative view prediction models as the overwhelming majority of them consider pre-segmented scenes (single object setups; e.g., 74) and cannot generalize beyond those settings. The same is the case for the model of [36], which was greatly outperformed by GRNNs in the work of [171].

Sim-to-Real (CARLA-to-KITTI) Transfer

We evaluate whether the 3D predictive feature representations learned in the CARLA simulator are useful for learning 3D object detectors in the real world by testing on the real KITTI dataset [43]. Specifically, we use view prediction pre-training in the CARLA train set, and box supervision from the KITTI train set, and evaluate 3D object detection in the KITTI validation set.

Existing real-world datasets do not provide enough camera viewpoints to support view-predictive learning. Specifically, in KITTI, all the image sequences come from a moving car and thus all viewpoints lie on a near-straight trajectory. Thus, simulation-to-real transferability of features is especially important for view predictive learning.

We show sim-to-real transfer results in Table 2.1. We compare the proposed view-contrastive prediction pre-training with view regression pre-training and random weight initialization (no pre-training). In all cases, we train a 3D object detector on the features, supervised using KITTI 3D box annotations. We also compare

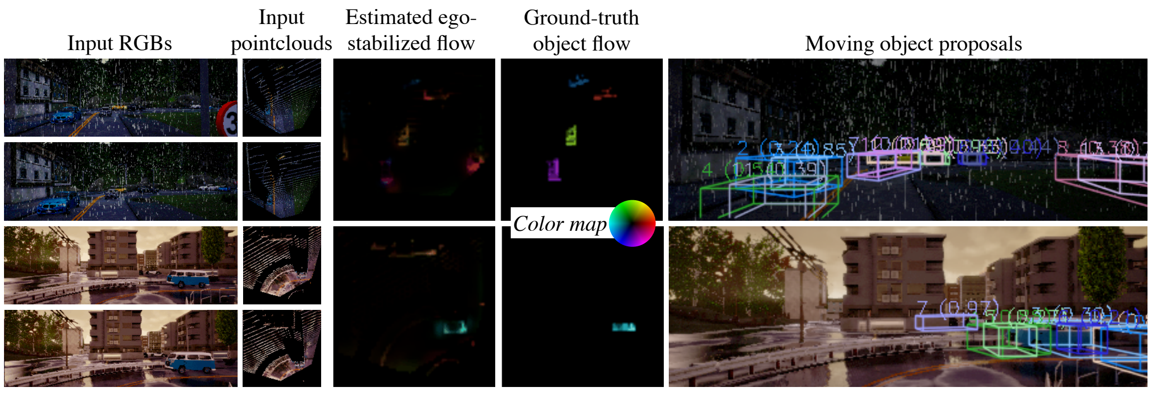

Figure 2.3: **3D feature flow and object proposals, in dynamic scenes.** Given the input frames on the left, our model estimates dense egomotion-stabilized 3D flow fields, and converts these into object proposals. We visualize colorized pointclouds and flow fields in a top-down (bird's eye) view.

freezing versus *finetuning* the weights of the pretrained inverse graphics network. The results are consistent with the CARLA tests: view-contrastive pre-training is best, view regression pre-training is second, and learning from annotations alone is worst. Note that depth in KITTI is acquired by a real LiDAR sensor, and therefore has lower density and more artifacts than CARLA, yet our model generalizes across this distribution shift.

2.4.2 Unsupervised 3D Moving Object Detection

In this section, we test our model's ability to detect moving objects in 3D with no 3D object annotations, simply by clustering 3D motion vectors. We use two-frame sequences of dynamic scenes from our CARLA validation set, and split it into two parts for evaluation: scenes where the camera is stationary, and scenes where the camera is moving. This splitting is based on the observation that moving object detection is made substantially more challenging under a moving camera.

We show precision-recall curves for 3D moving object detection under a *stationary camera* in Figure 2.4. We compare our model against a similar one trained with RGB view regression [171] and a 2.5D baseline. The 2.5D baseline computes 2D optical flow using PWC-Net [156], then proposes object masks by thresholding and clustering 2D flow magnitudes; these 2D masks are mapped to 3D boxes by segmenting the input pointcloud and fitting boxes to the points. Our model outperforms the baselines. Note that even with ground-truth 2D flow, ground-truth depth, and an oracle threshold, a 2.5D baseline can at best only capture the currently-visible fragment of each object. As a result, a 3D proposal computed from 2.5D often underestimates the extent of the object. Our model does not have this issue, since it imagines the full 3D scene at

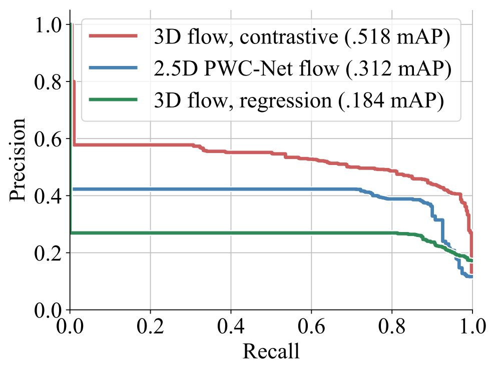

Figure 2.4: **Unsupervised 3D moving object detection with a *stationary* camera.**

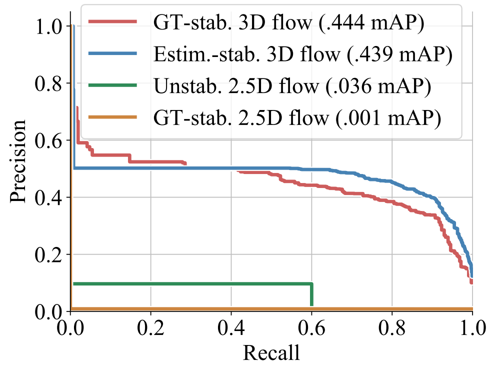

Figure 2.5: **Unsupervised 3D moving object detection with a *moving* camera**

each timestep.

We show precision-recall curves for 3D moving object detection under a *moving camera* in Figure 2.5. We compare our model where egomotion is predicted by our neural egomotion module, against our model with ground-truth egomotion, as well as a 2.5D baseline, and a stabilized 2.5D baseline. The 2.5D baseline uses optical flow estimated from PWC-Net as before. To stabilize the 2.5D flow, we subtract the ground-truth scene flow from the optical flow estimate before generating proposals. Our model's performance is similar to its level in static scenes, suggesting that the egomotion module's stabilization mechanism effectively disentangles camera motion from the 3D feature maps. The 2.5D baseline performs poorly in this setting, as expected. Surprisingly, performance drops further after stabilizing the 2D flows for egomotion. We confirmed this is due to the estimated scene flow being imperfect: subtracting ground-truth scene flow leaves many motion fragments in the background. With ground-truth 2D flow, the baseline performs similar to its static-scene level.

We have attempted to compare against the unsupervised object segmentation methods proposed in Kosiorek et al. [81] and Hsieh et al. [63] by adapting the publicly available code to the task. These models consume a video as input, and predict the locations of 2D object bounding boxes, as well as frame-to-frame displacements, in order to minimize a view regression error. We were not able to produce meaningful results from these models. The success of Hsieh et al. [63] may partially depend on carefully selected priors for the 2D bounding box locations and sizes, to match the statistics of the "Moving MNIST" dataset used in that work (as suggested in the official code). For our CARLA experiments, we do not assume knowledge of priors for box locations or sizes.

Method	Full	Static	Moving
Zero-motion	**0.19**	**0.0**	6.97
View regression pret.	0.77	0.63	5.55
View-contrastive pret.	0.26	0.17	**3.33**

Table 2.2: **Mean endpoint error of the 3D flow in egomotion-stabilized scenes.** View-contrastive features lead to more accurate motion estimation. Error is L_1 in voxel units.

2.4.3 3D Motion Estimation

In this section, we evaluate accuracy of our 3D imagination flow module. The previous section evaluated this module indirectly since it plays a large part in unsupervised 3D moving object detection; here we evaluate its accuracy directly. We use two-frame video sequences of dynamic scenes from our CARLA validation set.

We compare 3D flow trained over frozen 3D feature representations obtained from the proposed view-contrastive prediction, against flow trained over frozen features from the baseline RGB regression model [171]. We also compare against a zero-motion baseline that predicts zero motion everywhere. Since approximately 97% of the voxels belong to the static scene, a zero-motion baseline is very competitive in an overall average. We therefore evaluate error separately in static and moving parts of the scene. We show the motion estimation errors in Table 2.2. Our method achieves dramatically lower error than the RGB regression baseline, which suggests that the proposed view-contrastive objectives in 3D and 2D result in learning features that are correspondable across time, even for moving objects, despite the fact that the features were learned using only multi-view data of static scenes.

2.4.4 Limitations

The proposed model has two important limitations. First, our work assumes an embodied agent that can move around at will. This is hard to realize in the real world, and indeed there are almost no existing datasets with enough camera views to approximate this. Second, our model architecture consumes a lot of GPU memory, due to its extra spatial dimension . This severely limits either the resolution or the metric span of the latent map $\mathcal{M}$. On 12G Titan X GPUs we encode a space sized $32m \times 32m \times 8m$ at a resolution of $128 \times 128 \times 32$; with a batch size of 4, iteration time is 0.2s/iter. Supervised 3D object detectors typically cover twice this metric range. Sparsifying our feature grid, or using points instead of voxels, are clear areas for future work.

2.5 Conclusion

We propose models that learn space-aware 3D feature abstractions of the world given 2.5D input, by minimizing 3D and 2D view-contrastive prediction objectives. We show that view-contrastive prediction leads to features useful for 3D object detection, both in simulation and in the real world. We further show that the ability to visually imagine full 3D egomotion-stable scenes allows us to estimate dense 3D motion fields, where clustering non-zero motion allows objects to emerge without any human supervision. Our experiments suggest that the ability to imagine 3D visual information can drive 3D object detection. Instead of learning from annotations, the model learns by moving and watching objects move [46].

Chapter 3

Tracking Emerges by Looking Around Static Scenes, with Neural 3D Mapping

3.1 Introduction

A large part of the real world almost never moves. This may be surprising, since moving entities easily attract our attention [40], and because we ourselves spend most of our waking hours continuously in motion. Objects like roads and buildings, however, stay put. Can we leverage this property of the world to learn visual features suitable for interpreting complex scenes with many moving objects?

In this paper, we hypothesize that a correspondence module learned in static scenes will also work well in dynamic scenes. This is motivated by the fact that the content of dynamic scenes is the same as the content of static scenes. We would like our hypothesis to be true, because correspondences are far cheaper to obtain in static scenes than in dynamic ones. In static scenes, one can simply deploy a Simultaneous Localization and Mapping (SLAM) module to obtain a 3D reconstruction of the scene, and then project reconstructed points back into the input imagery to obtain multiview correspondence labels. Obtaining correspondences in dynamic scenes would require taking into account the motion of the objects (i.e., tracking).

We propose to leverage multiview data of static points in arbitrary scenes (static or dynamic), to learn a neural 3D mapping module which produces features that are correspondable across viewpoints *and* timesteps. The neural 3D mapper consumes RGB-D (color and depth) data as input, and produces a 3D voxel grid of deep features as output. We train the voxel features to be correspondable across viewpoints, using a contrastive loss. At test time, given an RGB-D video with approximate camera poses, and given the 3D box of an object to track, we track the target object by generating a map of each timestep and locating the object's features within each map.

In contrast to models that represent video streams in 2D [176, 180, 182], our model's 3D scene representation is disentangled from camera motion and projection artifacts. This provides an inductive bias that scene elements maintain their size and shape across changes in camera viewpoint, and reduces the need for scale invariance in the model's parameters. Additionally, the stability of the 3D map under camera motion allows the model to constrain correspondence searches to the 3D area where the target was last seen, which is a far more reliable cue than 2D pixel coordinates. In contrast to models that use 2.5D representations (e.g., scene flow [102]), our neural 3D maps additionally provide features for partially occluded areas of the scene, since the model can infer their features from context. This provides an abundance of additional 3D correspondence candidates at test time, as opposed to being limited to the points observed by a depth sensor.

Our work builds on geometry-aware recurrent neural networks (GRNNs) [171]. GRNNs are modular differentiable neural architectures that take as input RGB-D streams of a static scene under a moving camera and infer 3D feature maps of the scene, estimating and stabilizing against camera motion. The work of Harley et al. [54] showed that training GRNNs for contrastive view prediction in static scenes helps semi-supervised 3D object detection, as well as moving object discovery. In this paper, we extend those works to learn from dynamic scenes with independently moving objects, and simplify the GRNN model by reducing the number of losses and modules. Our work also builds on the work of Vondrick et al. [176], which showed that 2D pixel trackers can emerge without any tracking labels, through self-supervision on a colorization task. In this work, we show that 3D voxel trackers can emerge without tracking labels, through contrastive self-supervision on static data. In the fact that we learn features from correspondences established through triangulation, our work is similar to Dense Object Nets [37], though that work used object-centric data, and used background subtraction to apply loss on static objects, whereas in our work we do moving-object subtraction to apply loss on *anything* static. We do not assume a priori that we know which objects to focus on.

We test the proposed architectures in simulated and real datasets of urban driving scenes (CARLA [34] and KITTI [43]). We evaluate the learned 3D visual feature representations on their ability to track objects over time in 3D. We show that the learned visual feature representations can accurately track objects in 3D, simply supervised by observing static data. Furthermore, our method outperforms 2D and 2.5D baselines, demonstrating the utility of learning a 3D representation for this task instead of a 2D one.

The main contribution of this paper is to show that learning feature correspondences from static 3D points causes 3D object tracking to emerge. We additionally introduce a neural 3D mapping module which simplifies prior works on 3D inverse graphics, and learns from a simpler objective than considered in prior works.

3.2 Related Work

Learning to see by moving Both cognitive psychology and computational vision have realised the importance of motion for the development of visual perception [46, 187]. Predictive coding theories [41, 129] suggest that the brain predicts observations at various levels of abstraction; temporal prediction is thus of central interest. These theories currently have extensive empirical support: stimuli are processed more quickly if they are predictable [101, 121], prediction error is reflected in increased neural activity [7, 129], and disproven expectations lead to learning [145]. Several computational models of frame predictions have been proposed [36, 113, 129, 161, 171]. Alongside future frame prediction, predicting some form of contextual or missing information has also been explored, such as predicting frame ordering [86], temporal instance-level associations [180] color from grayscale [176], egomotion [1, 70] and motion trajectory forecasting [177]. Most of these unsupervised methods are evaluated as pre-training mechanisms for object detection or classification [54, 106, 177, 180].

Video motion segmentation literature explores the use of videos in unsupervised moving object discovery [122]. Most motion segmentation methods operate in 2D image space, and cluster 2D optical flow vectors or 2D flow trajectories to segment moving objects [9, 38, 111], or use low-rank trajectory constraints [19, 25, 167]. Our work differs in that we address object detection and segmentation in 3D as opposed to 2D, by estimating 3D motion of the "imagined" (complete) scene, as opposed to 2D motion of the pixel observation stream.

Vision as inverse graphics Earlier works in Computer Vision proposed casting visual recognition as inverse rendering [112, 204], as opposed to feedforward labelling. The "blocks world" of Roberts [134] had the goal of reconstructing the 3D scene depicted in the image in terms of 3D solids found in a database. A key question to be addressed is: what representations should we use for the intermediate latent 3D structures? Most works seek to map images to explicit 3D representations, such as 3D pointclouds [170, 175, 190, 206], 3D meshes [75, 95], or binary 3D voxel occupancies [74, 168, 191]. The aforementioned manually designed 3D representations, e.g., 3D meshes, 3D keypoints, 3D pointclouds, 3D voxel grids, may not be general enough to express the rich 3D world, which contains liquids, deformable objects, clutter, dirt, wind, etc., and at the same time may be over descriptive when detail is unnecessary. In this work, we opt for *learning-based 3D feature representations* extracted end-to-end from the RGB-D input stream as proposed by Tung et al. [171] and Harley et al. [54]. We extend the architectures of those works to handle and learn from videos of dynamic scenes, as opposed to only videos of static scenes.

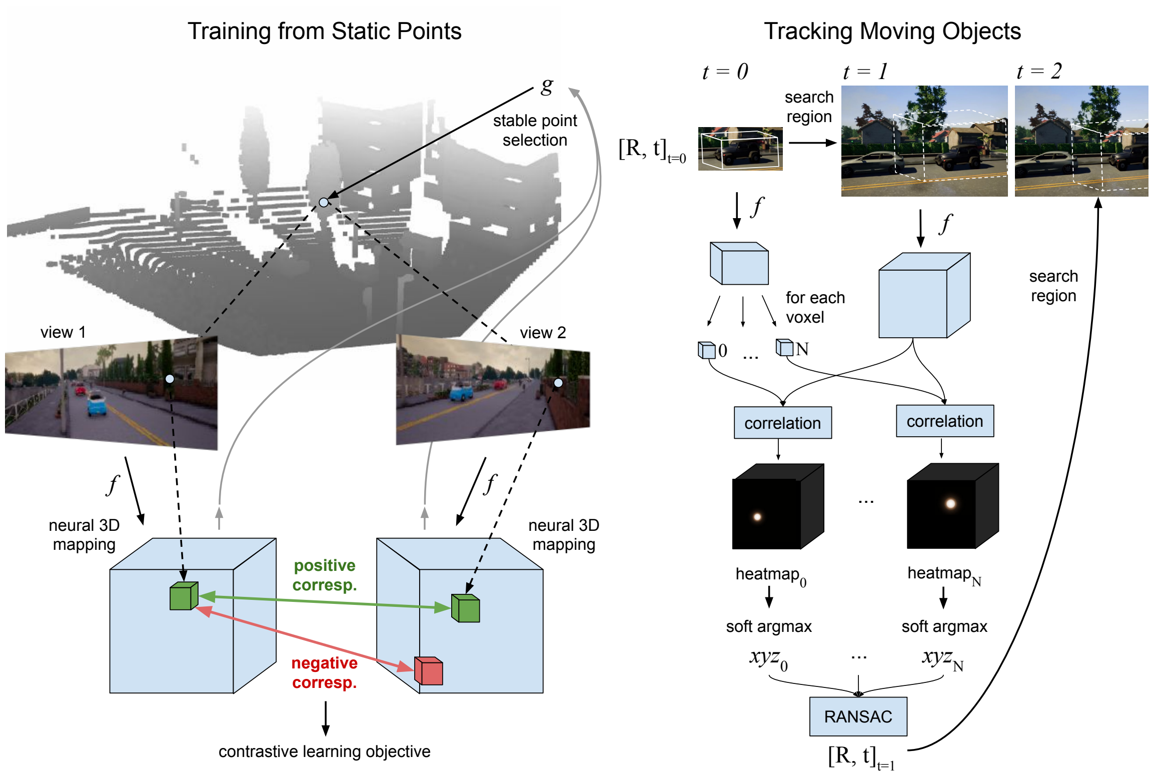

Figure 3.1: **Tracking emerges by looking around in static scenes, with neural 3D mapping.** *Left: Training regime.* Our learned 3D neural mapper f maps RGB-D inputs into featurized 3D scene maps. Points that correspond across multiple views, provided by static data or selected by a reliability network g, provide labels for a contrastive learning objective. *Right: Testing regime.* Given an object box in the zeroth frame, our encoder creates features for the object, and searches for re-occurences of those features in the 3D scene maps of future time steps. The model then explains the full-object motion via a robust estimate of the rigid transformation across frames. Each estimated transformation is used to initialize the search region in the next timestep. Our model does not use any human supervision, and never trains explicitly for tracking; instead, it is supervised only by observing static points in 3D scenes.

3.3 Learning to Track with Neural 3D Mapping

We consider a mobile agent that can move around in a 3D scene, and observe it from multiple viewpoints. The scene can optionally contain dynamic (moving and/or deforming) objects. The agent has an RGB camera with known intrinsics, and a depth sensor registered to the camera's coordinate frame. It is reasonable to assume that a mobile agent who moves at will has access to its approximate egomotion, since it chooses where to move and what to look at [117]. In simulation, we use ground truth camera poses; in real data, we use approximate camera poses provided by an inertial navigation system (GPS and IMU). In simulation, we use random viewpoints; in real data, we use just one forward-facing camera (which is all that is available). Note that a more sophisticated mobile agent might attempt to select viewpoints intelligently at training time.

Given the RGB-D and egomotion data, our goal is to learn 3D feature representations that can correspond entities across time, despite variations in pose and appearance. We achieve this by training inverse graphics neural architectures that consume RGB-D videos and infer 3D feature maps of the full scenes, as we describe in Section 3.3.1. To make use of data where some parts are static and other parts are moving, we learn to identify static 3D points by estimating a reliability mask over the 3D scene, as we describe in Section 3.3.2. Finally, we track in 3D, by re-locating the object within each timestep's 3D map, as described in Section 3.3.3. Figure 3.1 shows an overview of the training and testing setups.

3.3.1 Neural 3D Mapping

Our model learns to map an RGB-D (RGB and depth) image to a 3D feature map of the scene in an end-to-end differentiable manner. The basic architecture is based on prior works [54, 171], which proposed view prediction architectures with a 3D bottleneck. In our case, the 3D feature map is the *output* of the model, rather than an intermediate representation.

Let $\mathcal{M} \in \mathbb{R}^{W \times H \times D \times C}$ denote the 3D feature map representation, where W, H, D, C denote the width, height, depth and number of feature channels, respectively. The map corresponds to a large cuboid of world space, placed at some pose of interest (e.g., surrounding a target object). Every (x, y, z) location in the 3D feature map $\mathcal{M}$ holds a C-length feature vector that describes the semantic and geometric content of the corresponding location of the world scene. To denote the feature map of timestep i, we write $\mathcal{M}^{(i)}$. We denote the function that maps RGB-D inputs to 3D feature maps as $f : (I, D) \mapsto \mathcal{M}$. To implement this function, we voxelize the inputs into a 3D grid, then pass this grid through a 3D convolutional network, and L_2-normalize the outputs.

Tung et al. [171] learned the parameters of f by predicting RGB images of unseen viewpoints, and applying a regression loss; Harley et al. [54] demonstrated that this can be outperformed by contrastive prediction objectives, in 2D and 3D. Here, we drop the view prediction task altogether, and focus entirely on a 3D correspondence objective: if a static point (x, y, z) is observed in two views i and j, the corresponding features $m_i = \mathcal{M}^{(i)}_{x,y,z}, m_j = \mathcal{M}^{(j)}_{x,y,z}$ should be similar to each other, and distinct from other features. We achieve this with a cross entropy loss [16, 58, 113, 151]:

$$\mathcal{L}_{i,j} = -\log \frac{\exp(m_i^\top m_j/\tau)}{\sum_{k\neq i} \exp(m_i^\top m_k/\tau)}, \tag{3.1}$$

where τ is a temperature parameter, which we set to 0.07, and the sum over $k \neq i$ iterates over non-corresponding features. Note that indexing correctly into the 3D scene maps to obtain the correspondence pair m_i, m_j requires knowledge of the relative camera transformation across the input viewpoints; we encapsulate this registration and indexing in the notation $\mathcal{M}_{x,y,z}$. Following He et al. [58], we obtain a large pool of negative correspondences through the use of an offline dictionary, and stabilize training with a "slow" copy of the encoder, f_{slow}, which is learned via high-momentum updates from the main encoder parameters.

Since the neural mapper (a) does not know a priori which voxels will be indexed for a loss, and (b) is fully convolutional, it learns to generate view-invariant features densely in its output space, even though the supervision is sparse. Furthermore, since (a) the model is encouraged to generate correspondable features invariant to viewpoint, and (b) varying viewpoints provide varying contextual support for 3D locations, the model learns to *infer* corrrespondable features from limited context, which gives it robustness to partial occlusions.

3.3.2 Inferring static points for self-supervision in dynamic scenes

The training objective in the previous subsection requires the location of a static point observed in two or more views. In a scenario where the data is made up entirely of static scenes, as can be achieved in simulation or in controlled environments, obtaining these static points is straightforward: any point on the surface of the scene will suffice, provided that it projects into at least two camera views.

To make use of data where some parts are static and other parts are moving, we propose to simply discard data that appears to be moving. We achieve this by training a neural module to take the difference of two scene features $\mathcal{M}^{(i)}, \mathcal{M}^{(j)}$ as input, and output a "reliability" mask indicating a per-voxel confidence that the scene cube within the voxel is static: $g : (\mathbf{sg}(\mathcal{M}^{(i)} - \mathcal{M}^{(j)})) \mapsto \mathcal{R}^{W,H,D}$, where $\mathbf{sg}$ stops gradients from flowing from g into the function f which produces $\mathcal{M}^{(i)}, \mathcal{M}^{(j)}$. We implement g as a per-voxel classifier, with 2-layer fully-connected network applied fully convolutionally. We do not assume to have true labels of moving/static voxels, so we generate synthetic labels using static data: given two maps of the *same* scene $\mathcal{M}^{(i)}, \mathcal{M}^{(j)}$, we generate positive-label inputs with $\mathcal{M}^{(i)} - \mathcal{M}^{(j)}$ (as normal), and generate negative-label inputs with $\mathcal{M}^{(i)} - \text{shuffle}(\mathcal{M}^{(j)})$, where the shuffle operation ruins the correspondence between the two tensors. After this training, we deploy this network on pairs of frames from dynamic scenes, and use it to select high-confidence static data to further train the encoder f.

Our training procedure is then: (1) learn the encoder f on static data; (2) learn the reliability function g; (3) in dynamic data, finetune f on data selected by g. Steps 2-3 can be repeated a number of times. In practice we find that results do not change substantially after the first pass.

3.3.3 Tracking via point correspondences

Using the learned neural 3D mapper, we track an object of interest over long temporal horizons by re-locating it in a map produced at each time step. Specifically, we re-locate each voxel of the object, and use these new locations to form a motion field. We convert these voxel motions into an estimate for the entire object by fitting a rigid transformation to the correspondences, via RANSAC.

We assume we are given the target object's 3D box on the zeroth frame of a test video. Using the zeroth RGB-D input, we generate a 3D scene map centered on the object. We then convert the 3D box into a set of coordinates X_0, Y_0, Z_0 which index into the map. Let $m_i = \mathcal{M}^{(0)}_{x_i \in X_0, y_i \in Y_0, z_i \in Z_0}$ denote a voxel feature that belongs to the object. On any subsequent frame t, our goal is to locate the new coordinate of this feature, denoted x_i^t, y_i^t, z_i^t. We do so via a soft spatial argmax, using the learned feature space to provide correspondence confidences:

$$(x_i^t, y_i^t, z_i^t) = \sum_{x \in X, y \in Y, z \in Z} \left(\frac{\exp(m_i^\top \mathcal{M}^{(t)}_{x,y,z})}{\sum_{x_2 \in X, y_2 \in Y, z_2 \in Z} \exp(m_i^\top \mathcal{M}^{(t)}_{x_2,y_2,z_2})} (x, y, z) \right), \qquad (3.2)$$

where X, Y, Z denote the set of coordinates in the search region. We then compute the motion of the voxel as $(\delta^t_{x_i}, \delta^t_{y_i}, \delta^t_{z_i}) = (x_i^t, y_i^t, z_i^t) - (x_i, y_i, z_i)$. After computing the motion of every object voxel in this way, we use RANSAC to find a rigid transformation that explains the majority of the correspondences. We apply this rigid transform to the input box, yielding the object's location in the next frame.

Vondrick et al. [176] computed a similar attention map during tracking (in 2D), but did not compute its soft argmax, nor explain the object's motion with a single transformation, but rather propagated a "soft mask" to the target frame, which is liable to grow or shrink. Our method takes advantage of the fact that all coordinates are 3D, and makes the assumption that the objects are rigid, and propagates a fixed-size box from frame to frame.

We empirically found that it is critical to constrain the search region of the tracker in 3D. In particular, on each time step we create a search region centered on the object's last known position. The search region is $16m \times 2m \times 16m$ large, which is half of the typical full-scene resolution. This serves three purposes. The first is: it limits the number of spurious correspondences that the model can make, since it puts a cap on the metric range of the correspondence field, and thereby reduces errors. Second: it "re-centers" the model's field of view onto the target, which allows the model to incorporate maximal contextual information surrounding the target. Even if the bounds were sufficiently narrow to reduce spurious correspondences, an object at the *edge* of the field of view will have less-informed features than an object at the middle, due to the model's convolutional architecture. The third reason is computational: even 2D works [176] struggle with the computational expense of the large matrix multiplications involved in this type of soft attention, and in 3D the expense is higher. Searching locally instead of globally makes Eq. 3.2 tractable.

3.4 Experiments

We test our model in the following two datasets:

1. **Synthetic RGB-D videos of urban scenes rendered in the CARLA simulator** [34]. CARLA is an open-source photorealistic simulator of urban driving scenes. It permits moving the camera to any desired viewpoint in the scene. We obtain data from the simulator as follows. We begin by generating 10000 autopilot episodes of 16 frames each, at 10 FPS. We define 18 viewpoints along a 40m-radius hemisphere anchored to the ego-car (i.e., it moves with the car). In each episode, we sample 6 random viewpoints from the 18 and randomly perturb their pose, and then capture each timestep of the episode from these 6 viewpoints. We discard episodes that do not have an object in-bounds for the duration of the episode.

   We treat the Town1 data as the "training" set, and the Town2 data as the "test" set, so there is no overlap between the train and test sets. This yields 4313 training videos, and 2124 test videos.

2. **Real RGB-D videos of urban scenes, from the KITTI dataset** [43]**.** This data was collected with a sensor platform mounted on a moving vehicle, with a human driver navigating through a variety of areas in Germany. We use the "left" color camera, and LiDAR sweeps synced to the RGB images.

   For training, we use the "odometry" subset of KITTI; it includes egomotion information accurate to within 10cm. The odometry data includes ten sequences, totalling 23201 frames.

   We test our model in the validation set of the "tracking" subset of KITTI, which has twenty labelled sequences, totalling 8008 frames. For supervised baselines, we split this data into 12 training sequences and 8 test sequences. For evaluation, we create 8-frame subsequences of this data, in which a target object has a valid label for all eight frames. This subsequencing is necessary since objects are only labelled when they are within image bounds. The egomotion

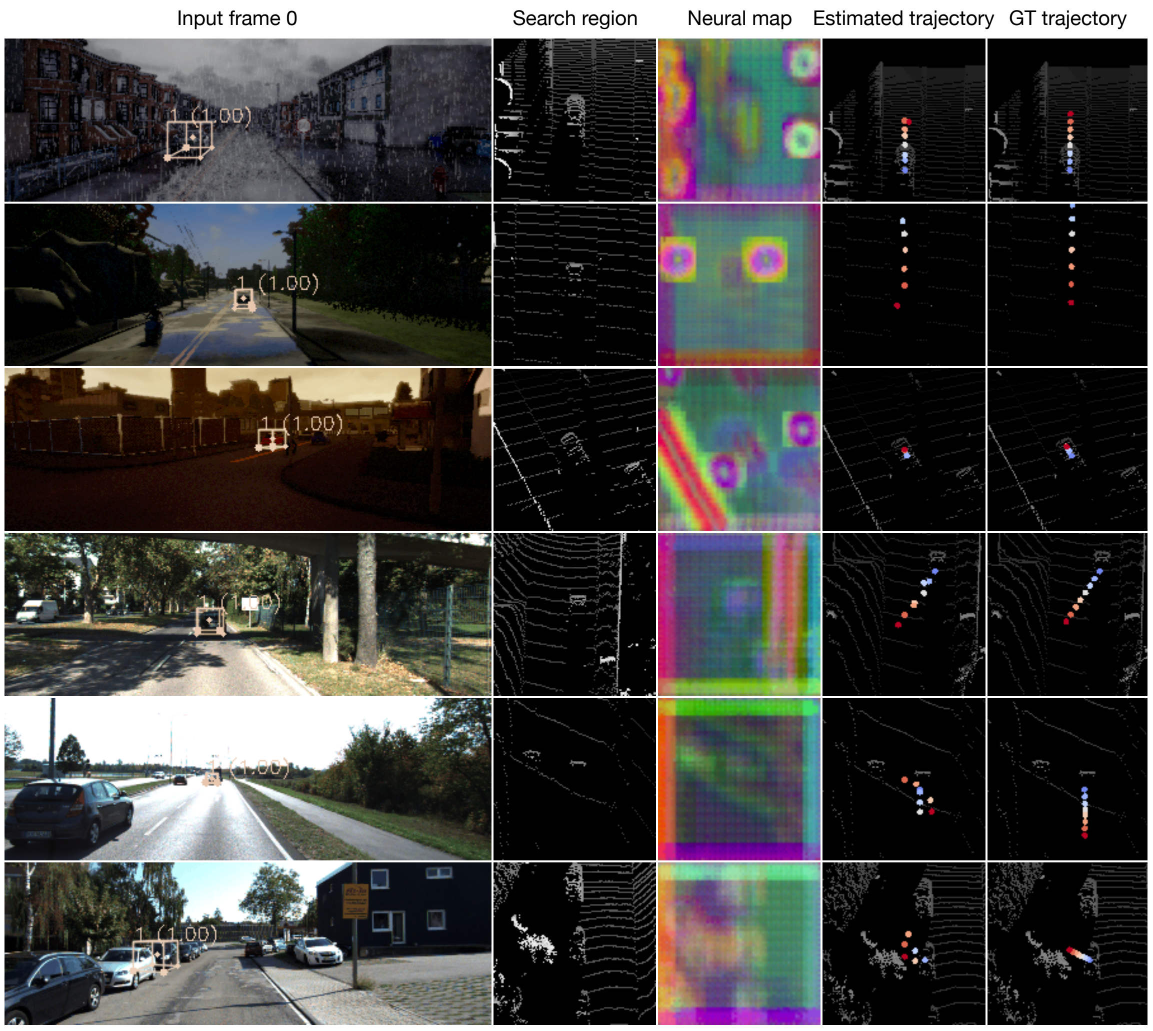

Figure 3.2: **Visualization of tracking inputs, inferred 3D scene features, and tracking outputs.** Given the input frame on the left, along with a 3D box specifying the object to track, the model (1) creates a search region for the object centered on the object's position (bird's eye view of occupancy shown), (2) encodes this region into a 3D neural map (bird's eye view of PCA compression shown), and (3) finds correspondences for the object's features. The top three rows show CARLA results; the bottom three rows show KITTI results.

information in the "tracking" data is only approximate.

We evaluate our model on its ability to track objects in 3D. On the zeroth frame, we receive the 3D box of an object to track. On each subsequent frame, we estimate the object's new 3D box, and measure the intersection over union (IOU) of the estimated box with the ground truth box. We report IOUs as a function of timesteps.

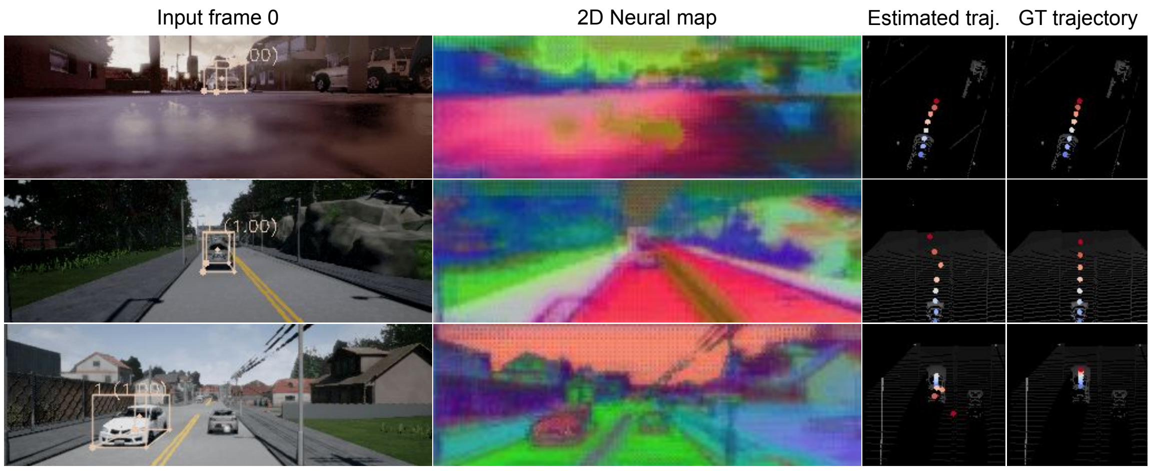

Figure 3.3: **Visualization of tracking inputs, inferred 2D scene features, and tracking outputs, for our adaptation of Dense Object Nets [37].** This model operates in the same way as our full 3D one, but creates features in 2D, and unprojects them to a sparse 3D grid for tracking.

3.4.1 Baselines

We evaluate the following baselines.

- **Unsupervised 3D flow [54].** This model uses an unsupervised architecture similar to ours, but with a 2-stage training procedure, in which features are learned first from static scenes and frozen, then a 3D flow module is learned over these features in dynamic scenes. We extend this into a 3D tracker by "chaining" the flows across time, and by converting the trajectories into rigid motions via RANSAC.
- **2.5D dense object nets [37].** This model learns to map input images into dense 2D feature maps, and uses a contrastive objective at known correspondences across views. We train this model using static points for correspondence labels (like our own model). We extend this model into a 3D tracker by "unprojecting" the learned embeddings into sparse 3D scene maps, then applying the *same tracking pipeline* as our own model.
- **2.5D tracking by colorization [83, 176].** This model learns to map input images into dense 2D feature maps, using an an RGB reconstruction objective. The model trains as follows: given two RGB frames, the model computes a feature map for each frame; for each pixel of the first frame's feature map, we compute that feature's similarity with all features of the second frame, and then use that similarity matrix to take a weighted combination of the second frame's colors; this color combination at every pixel is used as the reconstruction of

the first frame, which yields an error signal for learning. We extend this model into a 3D tracker in the same way that we extended the "dense object nets" baseline.

- **3D neural mapping with random features.** This model is equivalent to our proposed model but with randomly-initialized network parameters. This model may be expected to perform at better-than-chance levels due to the power of random features [128] and due to the domain knowledge encoded in the architecture design.
- **3D fully convolutional siamese tracker (supervised)** [4]**.** This is a straightforward 3D upgrade of a fully convolutional 2D siamese tracker, which uses the object's feature map as a cross correlation template, and tracks the object by taking an argmax of the correlation heatmap at each step. It is necessary to supervise this model with ground-truth box trajectories. We also evaluate a "cosine windowing" variant of this model, which suppresses correlations far from the search region's centroid [4].
- **3D siamese tracker with random features.** This model is equivalent to the 3D siamese supervised tracker, but with randomly-initialized network parameters. Similar to the random version of 3D neural mapping, this model measures whether random features and the implicit biases are sufficient to track in this domain.
- **Zero motion.** This baseline simply uses the input box as its final estimate for every time step. This baseline provides a measure for how quickly objects tend to move in the data, and serves as a lower bound for performance.

All of these are fairly simple trackers. A more sophisticated approach might incrementally update the object template [99], but we leave that for future work.

Ablations. We compare our own model against the following ablated versions. First, we consider a model without search regions, which attempts to find correspondences for each object point in the *entire* 3D map at test time. This model is far more computationally expensive, since it requires taking the dot product of each object feature with the entire scene. Second, we consider a similar "no search region" model but at half resolution, which brings the computation into the range of the proposed model. This ablation is intended to reveal the effect of resolution on accuracy. Third, we omit the "static point selection" (via the function g). This is intended to evaluate how correspondence errors caused by moving objects (violating the static scene assumption) can weaken the model.

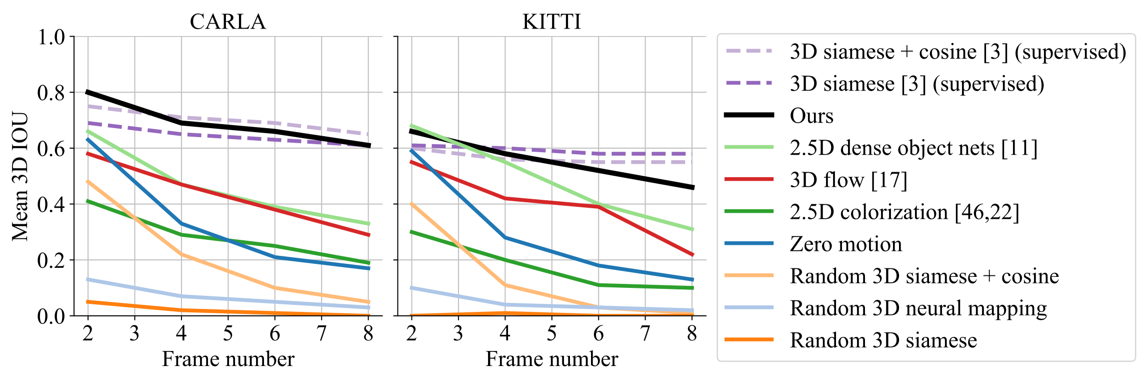

Figure 3.4: **Single-object tracking accuracy, in mean IOU across time, in CARLA (left) and KITTI (right).** Methods are sorted in the legend in decreasing order of mean IOU.

3.4.2 Quantitative results

We evaluate the mean 3D IOU of our trackers over time. Figure 3.4-left shows the results of this evaluation in CARLA. As might be expected, the supervised 3D trackers perform best, and cosine windowing improves results.

Our model outperforms all other unsupervised models, and nearly matches the supervised performance. The 2.5D dense object net performs well also, but its accuracy is likely hurt by the fact that it is limited exclusively to points observed in the depth map. Our own model, in contrast, can match against both observed and unobserved (i.e., hallucinated or inpainted) 3D scene features. The colorization model performs under the 2.5D dense object net approach, likely because this model only indirectly encourages correspondence via the colorization task, and therefore is a weaker supervision than the multi-view correspondence objectives used in the other methods.

Random features perform worse than the zero-motion baseline, both with a neural mapping architecture and a siamese tracking architecture. Inspecting the results qualitatively, it appears that these models quickly propagate the 3D box off of the object and onto other scene elements. This suggests that random features and the domain knowledge encoded in these architectures are not enough to yield 3D trackers in this data.

We perform the same evaluation in KITTI, and show the results in Figure 3.4-right. On this benchmark, accuracies are lower for all models, indicating that the task here is more challenging. This is likely related to the fact that (1) the egomotion is imperfect in this data, and (2) the tracking targets are frequently farther away than they are in CARLA. Nonetheless, the ranking of methods is the same, with our 3D neural mapping model performing best. One difference is that cosine windowing actually worsens siamese tracking results in KITTI. This is likely related to the fact that egomotion stabilization is imperfect in KITTI: a zero-motion prior is only helpful in frames where the target is stationary *and* camera motion is perfectly accounted for; otherwise it is detrimental.

We additionally split the evaluation on stationary vs. moving objects in CARLA. We use a threshold of $1m$ total distance (in world coordinates) across the 8-frame trajectories to split these categories. At the last timestep, the mean 3D IOU for all objects together it is 0.61 (as shown in Figure 3.4-left); for static objects only, the value is 0.64; for moving objects only, it is 0.56. This suggests that the model tracks stationary objects more accurately than moving objects, likely because their appearance changes less with respect to the camera and the background.

Finally, we evaluate the top two models in CARLA using the standard 2D tracking metrics, multi-object tracking accuracy (MOTA) and multi-object tracking precision (MOTP) [3], though we note that our task only has one target object per video. We find that the 3D siamese + cosine model (supervised) achieves a MOTA of 0.9578 and MOTP of 0.7407, while our model achieves MOTA 0.9125 and MOTP 0.8384. This suggests that the supervised tracker makes fewer misses, but our method delivers slightly better precision.

3.4.3 Qualitative results

We visualize our tracking results, along with inputs and colorized visualizations of our neural 3D scene maps, in Figure 3.2. To visualize the neural 3D maps, we take a mean along the vertical axis of the grid (yielding a "bird's eye view" 2D grid), compress the deep features in each grid cell to 3 channels via principal component analysis, normalize, and treat these values as RGB intensities. Comparing the 3D features learned in CARLA vs those learned in KITTI reveals a very obvious difference: the KITTI features appear blurred and imprecise in comparison with the CARLA features. This is likely due to the imperfect egomotion information, which leads to slightly inaccurate correspondence data at training time (i.e., occasional failures by the static point selector g).

In Figure 3.3, we visualize the features learned by Dense Object Nets in this data. From the PCA colorization it appears that objects are typically colored differently from their surroundings, which is encouraging, but the embeddings are not as clear as those in the original work [37], likely because this domain does not have the benefit of background image subtraction and object-centric losses.

3.4.4 Ablations

We evaluate ablated versions of our model, to reveal the effect of (1) search regions, (2) resolution, and (3) static point selection in dynamic scenes. Results are summarized

Method	Mean IOU
Ours in CARLA	0.61
. . . without search regions	0.40
. . . without search regions, at half resolution	0.25
Ours in KITTI	0.46
. . . without static point selection	0.39

Table 3.1: Ablations of our model.

in Table 3.1.

Without search regions, the accuracy of our model drops by 20 IOU points, which is a strong impact. We believe this drop in performance comes from the fact that search regions take advantage of 3D scene constancy, by reducing spurious correspondences in far-away regions of the scene.

Resolution seems to have a strong effect as well: halving the resolution of the wide-search model reduces its performance by 15 IOU points. This may be related to the fact that fewer points are then available for RANSAC to find a robust estimate of the object's rigid motion.

Since static point selection is only relevant in data with moving objects, we perform this experiment in KITTI (as opposed to CARLA, where the training domain is all static by design). The results show that performance degrades substantially without the static point selection. This result is to be expected, since this ablation causes erroneous correspondences to enter the training objective, and thereby weakens the utility of the self-supervision.

3.4.5 Limitations

The proposed model has three important limitations. First, our work assumes access to RGB-D data with accurate egomotion data at training time, with a wide variety of viewpoints. This is easy to obtain in simulators, but real-world data of this sort typically lies along straight trajectories (as it does in KITTI), which limits the richness of the data. Second, our model architecture requires a lot of GPU memory, due to its third spatial dimension. This severely limits either the resolution or the metric span of the latent map $\mathcal{M}$. On 12G Titan X GPUs we encode a space sized $32m \times 4m \times 32m$ at a resolution of $128 \times 32 \times 128$, with a batch size of 4; iteration time is 0.2s/iter. Sparsifying our feature grid, or using points instead of voxels, are clear areas for future work. Third, our test-time tracking algorithm makes two strong assumptions: (1) a tight box is provided in the zeroth frame, and (2) the object is rigid. For non-rigid objects, merely propagating the box with the RANSAC solution would be insufficient, but the voxel-based correspondences might still be helpful.

3.5 Conclusion

We propose a model which learns to track objects in dynamic scenes just from observing static scenes. We show that a multi-view contrastive loss allows us to learn rich visual representations that are correspondable not only across views, but across time. We demonstrate the robustness of the learned representation by benchmarking the learned features on a tracking task in real and simulated data. Our approach outperforms prior unsupervised 2D and 2.5D trackers, and approaches the accuracy of supervised trackers. Our 3D representation benefits from denser correspondence fields than 2.5D methods, and is invariant to the artifacts of camera projection, such as apparent scale changes of objects. Our approach opens new avenues for learning trackers in arbitrary environments, without requiring explicit tracking supervision: if we can obtain an accurate pointcloud reconstruction of an environment, then we can learn a tracker for that environment too.

Chapter 4

CoCoNets: Continuous Contrastive 3D Scene Representations

4.1 Introduction

Understanding the three-dimensional structure of objects and scenes may be a key for success of machine perception and control in object detection, tracking, manipulation and navigation. Exciting recent works have explored learning representations of objects and scenes from multiview imagery and capture the three-dimensional scene structure implicitly or explicitly with 3D binary or feature grids [147, 168, 169], 3D point feature clouds [185], implicit functions that map continuous world coordinates to 3D point occupancy [17, 20, 44, 103, 115, 149], as well as 1D or 2D feature maps [36]. These methods typically evaluate the accuracy of the inferred 3D scene occupancy [17, 20, 103, 110, 115, 168] and the fidelity of image views rendered from the 3D representation [36, 85, 105, 149, 185], as opposed to the suitability of representations for downstream semantic tasks. Methods that indeed focus on rendering photo-realistic images often give up on cross-scene generalization [105, 133], or focus on single-object scenes [149]. Methods that instead focus on learning semantically relevant scene representations are expected to generalize across scenes, and handle multi-object scenes. In the 2D image space, contrastive predictive coding has shown to generate state-of-the-art visual features for correspondence and recognition [53, 113], but does not encode 3D scene structure. In 3D voxel feature learning methods [54, 55], convolutional latent 3D feature grids encode the 3D structure and a view contrastive objective learns semantically useful 3D representations, but the grid resolution limits the discriminability of the features learnt. Recent exciting works combine 3D voxel grids and implicit functions and learn to predict 3D scene and object 3D occupancy from a single view with unlimited spatial resolution [118, 124]. The model proposed in this work brings together these two powerful ideas: 3D feature grids as a 3D-informed neural bottleneck for contrastive view prediction [54], and implicit functions for handling the resolution limitations of 3D grids [118].

We propose Continuous Contrastive 3D Networks (CoCoNets), a model that learns to map RGB-D images to infinite-resolution 3D scene feature representations by contrastively predicting views, in an object and scene agnostic way. Our model is trained to predict views of static scenes given 2.5D (color and depth; RGB-D) video streams as input, and is evaluated on its ability to detect and recognize objects in 3D. CoCoNets map the 2.5D input streams into 3D feature grids of the depicted scene. Given a target view and its viewpoint, the model first warps its inferred 3D feature map from the input view to a target view, then queries point features using their continuous coordinates, and pulls these features closer to the point features extracted from the target view at the same 3D locations. We use a contrastive loss to measure the matching error, and backpropagate gradients end-to-end to our differentiable modular architecture. At test time, our model forms plausible 3D completions of the scene given a *single RGB-D image* as input: it learns to fill in information behind occlusions, and infer the 3D extents of objects.

We demonstrate the advantages of combining 3D neural bottleneck, implicit functions and contrastive learning for 3D representation learning by comparing our model against state-of-the-art self-supervised models, such as i) contrastive learning for pointclouds [192], which shares a similar loss but not the amodal predictive ability of our model, ii) contrastive neural mapping [55], which can amodally inpaint a 3D discrete feature grid but suffers from limited spatial grid resolution, and iii) Dense Object Nets [37], which self-learns 2D (instead of 3D) feature representations with a triangulation-driven supervision similar to (i). Our experimental results can be summarized as follows: **(1)** 3D object tracking and re-identification: We show that scene representations learnt by CoCoNets can detect objects in 3D across large frame gaps better than the baselines [37, 55, 192]. **(2)** Supervised 3D object detection: Using the learnt 3D point features as initialization boosts the performance of the state-of-the-art Deep Hough Voting detector of [127]. **(3)** 3D cross-view and cross-scene object 3D alignment: We show that the learnt 3D feature representations can infer 6DoF alignment between the same object in different viewpoints, and across different objects of the same category, better than [37, 55, 192]. We further show that our model can predict image views (with or without depth as input) and 3D occupancies that outperform or are on par with the state-of-the-art view and occupancy prediction models [36, 124, 171].

In summary, the main contribution of this paper is a model that learns infinite-resolution 3D scene representations from RGB-D posed images, useful for tracking and corresponding objects in 3D, pre-training 3D object detectors, and predicting views and 3D occupancies. We set a new state-of-the-art in self-supervision of 3D feature representations.

4.2 Related work

Learning to 3D reconstruct objects and scenes Learning to infer 3D reconstructions of objects and scenes from single images or videos has been the goal of recent deep geometrical methods, that have explored a variety of explicit 3D representations, such as 3D point clouds [91, 109], 3D binary voxel occupancies [168, 195], or 3D meshes [47, 75]. Since detailed 3D supervision is only possible in large scale in simulation, many approaches attempt supervision from RGB or depth map prediction through differentiable rendering of the inferred 3D reconstruction [47, 75]. To handle limitations of spatial resolution of 3D voxel grids, recent approaches represent 3D object occupancy with implicit functions parameterized by deep neural networks trained to map continuous world 3D coordinates to the corresponding point 3D occupancy values [17, 20, 103, 115]. While these methods train one 3D shape implicit function per object, by assigning a 1D latent embedding to each object, some works [118, 124] train 3D grids of functions where each function takes care of estimating the occupancy of points in the vicinity of the corresponding 3D voxel centroid. These latter methods enjoy the generalization of 3D convolutions and can scale to multi-object scenes, while implicit functions parametrized by non-convolutional, fully connected networks are mostly limited to single object scenes [20]. Our approach also employs 3D grids of functions which predict feature embeddings for the corresponding continuous 3D world coordinates as opposed to merely occupancy.

Neural image synthesis with 3D inductive biases Deep image generative networks have shown compelling results in generating photorealistic images that match the image statistics of the unlabelled image collections they are trained on [48]. They are based on variational autoencoders [80], generative adversarial networks [6], generative flows [79], or autoregressive image pixel generators [173]. However, these 2D generative models learn to parameterize the manifold of 2D natural images, and struggle to generate images that are multi-view consistent, since the underlying 3D scene structure cannot be exploited. Generative models trained from multi-view data can render arbitrary views of an input scene [74, 139, 147, 161, 164]. Such multi-view generative models often restrict themselves to single-object scenes [108, 146, 149] or to a single complex scene without aiming at cross-scene generalization [105, 133], with the goal of generating high fidelity photorealistic images, replacing hand-engineered graphics engines. Their architectures incorporate many inductive biases of graphics engines, such as 3D-to-2D rendering modules [164] and explicit feature transformations to handle viewpoint changes [171]. Their lack of cross-scene generalization or their limitation to single object scenes makes it hard to adopt their inferred feature representations for visual recognition.

Learning visual feature representations by self-supervised view prediction Recent methods learn neural scene representations by predicting views of a scene under known egomotion [36, 54, 147, 149]. View prediction, as opposed to the related and very effective objective of feature learning via triangulation [192], results in explicitly or implicitly *amodal* representations, i.e., representations that can predict information missing from the input observations [107], as opposed to simply featurizing the visible image pixels [37]. Different view prediction methods for learning representations vary with respect to the amount of their reasoning regarding geometry and the underlying 3D structure of the scene [165]. The generative query network (GQN) of Eslami *et al.* [36] showed that it can predict alternative views of toy simulated scenes without explicit 3D structure inference, and demonstrated the usefulness of the inferred representations as pre-training for reinforcement behaviour learning tasks. Geometry-aware recurrent networks of Tung *et al.* [171] use a latent 3D feature map in their bottleneck for view prediction and demonstrate superior generalization, granted from the 3D convolutional inductive bias. Harley *et al.* [54] uses a similar 3D latent feature grid but optimizes for contrastive prediction as opposed to RGB regression, and demonstrate its usefulness for 3D object tracking and 3D moving object segmentation. Our work optimizes a contrastive view prediction objective similar to [54] but uses a 3D grid of implicit functions as its latent bottleneck. We empirically show that the emergent 3D feature representations are more accurate in 3D object tracking and visual correspondence than the features obtained from existing state-of-the-art 3D feature learning methods.

4.3 Continuous Contrastive 3D Networks (CoCoNets) for Learning Amodal Visual Representations

We consider a mobile agent that can move about the scene and has access to its egomotion. The agent has a color camera with known intrinsics, and a depth sensor registered to the camera's coordinate frame. We use groundtruth depth provided by the simulation environment, and we will show in Sec. 4.4 that the learned models generalize to the real world, where (sparser) depth is provided by a LiDAR unit.

CoCoNets learn 3D visual feature representations by collecting posed images in static scenes and doing contrastive view prediction. We describe the architecture in Sec. 4.3.1. We then evaluate the correspondability of the resulting 3D feature representations in 3D object re-identification and tracking in dynamic scenes (Section 4.4.1), as pre-training for 3D object detection (Section 4.4.2), and cross-object semantic visual correspondence (Section 4.4.3).

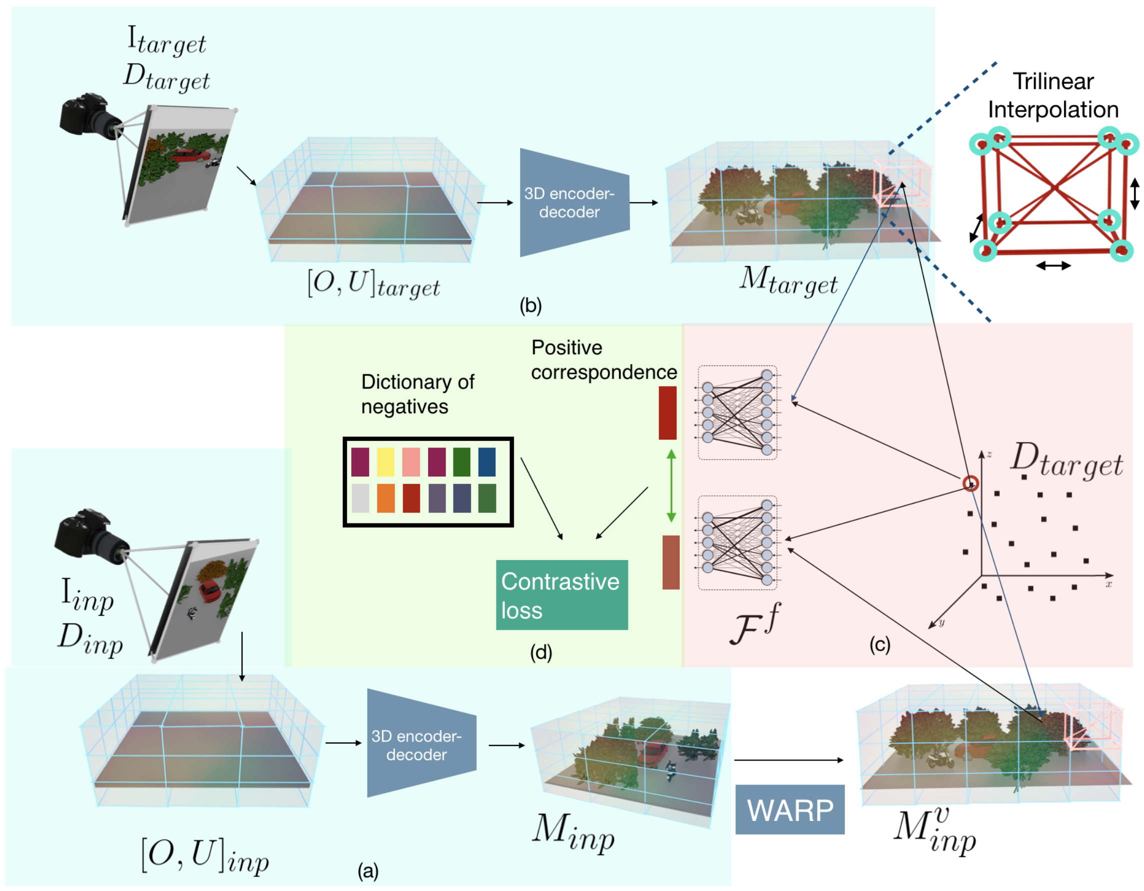

Figure 4.1: **Continuous Convolutional Contrastive 3D Networks** (CoCoNets) are trained to lift 2.5D images to 3D feature function grids of the scene by optimizing for view-contrastive prediction. (a) In the top-down path, the model encodes RGB-D images into a 3D feature map $\mathcal{M} \in \mathbb{R}^{w\times h\times d\times c}$, and uses explicit 3D feature transformations (translation and 3D rotation) to account for changes of viewpoint between the input and target views. (b) In the bottom-up path, we encode the RGB-D of the target viewpoint into a 3D feature cloud. (c) Given continuous 3D world coordinates (X, Y, Z) and its embedded code $v_{(X,Y,Z)}$ inferred via trilinear interpolation, a fully connected network maps the coordinates and the embedded code, to the feature vector of the 3D point at location (X, Y, Z). (d) Metric learning losses in 3D tie the two point cloud representations together.

4.3.1 Continuous Contrastive 3D Networks (CoCoNets)

Our model's architecture is illustrated in Figure 4.1. It is a neural network with a three-dimensional neural bottleneck $\mathcal{M} \in \mathbb{R}^{w \times h \times d \times c}$, which has three spatial dimensions (width w, height h, and depth d) and a feature dimension (c channels per grid location).

The latent state aims to capture an informative and geometrically-consistent 3D deep feature map of the world space. Therefore, the spatial extent corresponds to a large cuboid of world space, defined with respect to the camera's position at the first timestep. Each voxel in the 3D feature map $\mathcal{M}$ corresponds to a cuboid in the 3D scene depicted in the RGB-D image.

To be able to generate features at infinite spatial resolution, i.e., to featurize continuous 3D physical points within a voxel, we use implicit function parametrization. Our model's architecture of interpolating within a voxel grid resembles that of Peng *et al.* [118]. Let (X, Y, Z) denote the continuous world coordinate of a 3D point whose feature we wish to infer. First, we trilinearly interpolate the feature grid to obtain a a c-dimensional feature vector at point (X, Y, Z). Denoting this trilinearly interpolated feature vector as p for input point (X, Y, Z), we further obtain a 3d location conditioned feature vector using $\phi(p, x)$, where ϕ is a small fully-connected network. Finally we obtain our 32-dimensional embedding vector using $f_\theta(p, \phi(p, x))$, where f_θ represents a multi-block fully connected ResNet.

A similar fully-connected ResNet parametrization provides a point's binary occupancy $o_{(X,Y,Z)}$, and its RGB color value $c_{(X,Y,Z)}$. The three ResNets that predict features, occupancy and RGB values of 3D points do not share weights. During training, we use the point cloud of the target viewpoint to query our model for features, occupancies and colors in the corresponding visible 3D point locations and propagate gradients in an end-to-end manner to the same feature voxel 3D map $\mathcal{M}$. We denote the operation of obtaining point features, occupancies and colors by querying the feature map at continuous locations by $\mathcal{F}^f(\mathcal{M}, (X, Y, Z))$, $\mathcal{F}^o(\mathcal{M}, (X, Y, Z))$, and $\mathcal{F}^c(\mathcal{M}, (X, Y, Z))$. At test time, we query the model in both visible and non visible 3D point locations, to obtain an amodal completed 3D point feature cloud.

CoCoNets is made up of differentiable modules that go back and forth between 3D feature space and 2D image space. It can take as input a variable number of RGB-D images both at training time and test time. For simplicity, in our experiments we encode only a single view input at both train and test time.

2D-to-3D unprojection (Figure 4.1 (a, b)) This module converts the input RGB image $I \in \mathbb{R}^{w \times h \times 3}$ and depth map $D \in \mathbb{R}^{n \times 3}$ where, n is the size of the point-cloud) into 3D tensors using available camera intrinsics. The RGB is "unprojected" into a 3D tensor $\mathbf{U} \in \mathbb{R}^{w \times h \times d \times 3}$ by filling each 3D grid location with the RGB value of its corresponding subpixel. The pointcloud is converted to a 3D occupancy grid $\mathbf{O} \in \mathbb{R}^{w \times h \times d \times 1}$, by assigning each voxel a value of 1 or 0, depending on whether or not

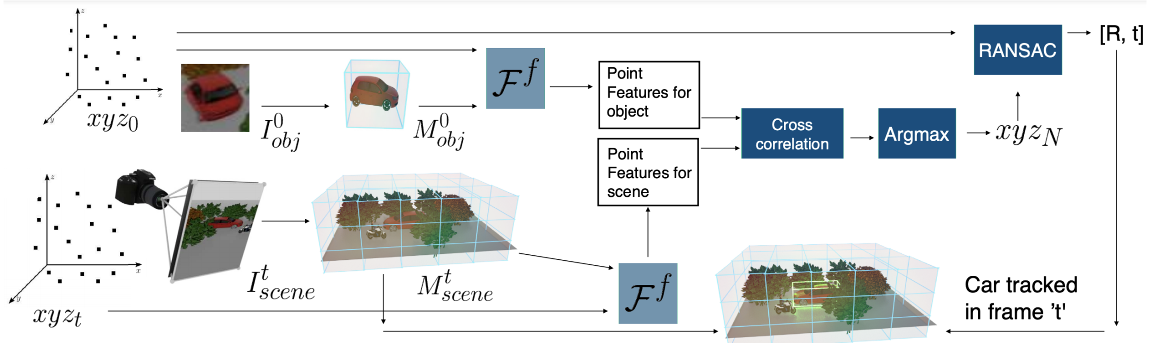

Figure 4.2: **3D object tracking using CoCoNets.** Given the cropped RGB-D image I_{obj} of the object to track at $t = 0$, our model infers the 3D object feature map $\mathcal{M}_{obj}$, and queries it using xyz_0, the point cloud of the object, to obtain object point features. Similarly, it obtains the point features of the entire scene at timestep t. Finally, it does cross correlation between these features to get xyz_N, where each i^{th} point in xyz_N is the point from the scene whose feature matched best with the feature for i^{th} point in xyz_0. We then apply RANSAC on xyz_0 and xyz_N to obtain the location of the car at timestep t.

a point lands in the voxel. We then convert the concatenation of these tensors into a 3D feature tensor $\mathcal{M} \in \mathbb{R}^{w \times h \times d \times c}$, via a 3D convolutional encoder-decoder network with skip connections. We L_2-normalize the feature in each grid cell.

3D-to-2D projection This module first warps the 3D feature map $\mathcal{M}$ to align it to the target viewpoint, yielding $\mathcal{M}^V$. It then generates a 3D feature point cloud by querying the model at the 3D point locations visible from the target viewpoint, provided by the target depth map D_{target}. It also generates a 2D feature map by computing for each visible 3D point the corresponding 2D pixel location using the camera intrinsics, $(x, y) = \mathrm{f}\frac{X}{Z}, \mathrm{f}\frac{Y}{Z}$, and copying the 3D point feature into that location in the 2D feature map. Note that each 3D feature point is mapped independently to 2D, in contrast to neural renderers that convolutionally map feature maps to image views [36, 147]. This independent point-by-point rendering is closer to the spirit of graphics operations [149].

3D contrastive learning of point features Given a pair of input RGB images (I_{inp}, I_{target}), depth maps (D_{inp}, D_{target}), and the camera pose change between input and target views V, we consider two types of representations for the target view:

- a top-down one, $\mathcal{M}^V_{inp}$ (Figure 4.1 (a)) by encoding the input RGB-D image (I_{inp}, D_{inp}) and orienting the feature map to the target viewpoint, and predicting the features for the target 3D points in D_{target} by querying the functions in $\mathcal{M}^V_{inp}$, obtaining the feature cloud $\{(X, Y, Z, \mathcal{F}(\mathcal{M}^V_{inp}, (X, Y, Z)))\}$ for $(X, Y, Z) \in$

D_{target} (Figure 4.1 (c)).

- a bottom-up one, $\mathcal{M}_{target}$ (Figure 4.1 (b)) by simply encoding the target RGB-D image I_{target}, D_{target} and predicting the features for the target 3D points in D_{target}, obtaining the feature cloud $\{(X, Y, Z, \mathcal{F}(\mathcal{M}_{target}, (X, Y, Z)))\}$ for $(X, Y, Z) \in D_{target}$ (Figure 4.1 (c)).

We use a contrastive InfoNCE loss [113] (Figure 4.1 (d)) to pull corresponding top-down and bottom-up point features close together in embedding space and push non-corresponding ones farther away:

$$\mathcal{L} = -\log \frac{\exp(\frac{q \cdot k_+}{\tau})}{\sum_{i=0}^{K} \exp \frac{q \cdot k_i}{\tau}}, \tag{4.1}$$

where τ is a temperature hyper-parameter and the sum in the denominator is over one positive and K negative samples. In the numerator, q represents $\mathcal{F}^f(\mathcal{M}_{inp}, (i, j, k))$, and k_+ is its corresponding positive sample $\mathcal{F}^f(\mathcal{M}_{target}, (m, n, o))$. The dot product computes a similarity. In practice, we randomly sample corresponding points from the top-down or bottom-up feature clouds at each training iteration. We also maintain a large pool of negative pairs in a dictionary, using the approach proposed by He *et al.*[58].

Our metric learning loss, if applied only on 3D points visible from both input and target views, coincides with the point contrastive metric learning of Xie *et al.* [192], a state-of-the-art 3D point feature learning method. CoCoNets can handle input and target views that actually have few or no points in common. We compare against Xie *et al.*'s state-of-the-art 3D feature learning model in our experiments, and demonstrate the importance of amodal completion for feature learning.

Occupancy prediction and RGB view regression We train a separate CoCoNet model to predict occupancy and RGB in novel viewpoints, which we denote CoCoNets-*OccRGB*. We use a standard binary cross entropy loss for the occupancy, and a regresion loss for RGB. The model predicts point clouds in target viewpoints and infers their occupancy and color from the implicit function grids $\mathcal{F}^o, \mathcal{F}^c$. Specifically, to predict a target RGB view and its occupancy given a source RGB-D image, after encoding and orienting the 3D feature map $\mathcal{M}$ to the target viewpoint, we predict the point occupancies of the target view, and predict the point RGB colors for the occupied points. These are then projected to the image space in the target view.

Jointly training RGB and occupancy prediction with contrastive view prediction did not improve discriminability of our feature representations. The novelty of our paper is in using voxel-implicit architectures for amodal contrastive 3D feature learning as opposed to 3D occupancy prediction [118, 124], or in addition to it.

RGB view regression without depth input Additionally, we also make use of NeRF's volumetric renderer [105], with features extracted from $\mathcal{M}_{input}$ as priors, to render novel views using just a single RGB image (no depth information) as input. We call this CoCoNets^{NoDepthRGB}. Qualitative results from this model can be seen in Figure 4.5.

4.4 Experiments

Useful visual feature representations are expected to correspond visual entities across variations in pose and appearance, as well as across intra-category variability. We evaluate the correspondability of our 3D scene representations in 3D object tracking and object re-identification, 3D pose estimation and cross-object correspondence. We further evaluate our features as pre-training for 3D object detector for the state-of-the-art method of [127]. Our experiments aim to answer the following questions:

1. Is amodal completion important for learning scene representations useful for visual correspondence, across time and across scenes? To this end, we compare 3D feature representations learned by CoCoNets against contrastive point clouds [192] that do not consider amodal completion but share a similar contrastive objective for 3D points visible in both scenes.
2. Is our 3D function parameterization important for learning discriminative visual representations? To this end, we compare with 3D contrastive neural mapping of Harley *et al.* [54] that does not consider continuous functions, but rather uses discrete 3D feature maps as the neural bottleneck.
3. Does the performance of 3D object detectors in pointclouds improve when using CoCoNets as initialization of the point features, and by how much? To this end, we compare with VoteNet [127], a model that uses PointNet++ [126] as their feature extractor and deep Hough voting for 3D object detection in point clouds.
4. Is the continous 3D convolutional bottleneck of CoCoNets useful for view prediction? To this end, we compare image views rendered from our model in novel scenes with images from GQN [36], which does not consider 3D neural bottleneck, and GRNN [171], which has a discrete 3D convolutional bottleneck.

We train and test CoCoNets in the simulated datasets of CARLA [34], ShapeNet [14] and test them further—without training—in the real-world KITTI dataset [43]. CARLA is an open-source photorealistic simulator of urban driving scenes, which permits moving the camera to any desired viewpoint in the scene. ShapeNet is a large scale 3D object repository which again permits camera placement at will. KITTI is an urban driving scene dataset collected from a camera mounted on a car.

4.4.1 Self-supervised 3D object tracking and re-indentification

Setup We evaluate the 3D feature representations of CoCoNets in their ability to track a 3D object over time, as well as across large frame gaps, a task known as object re-identification. Given the 3D bounding box of an object in the first frame, we are interested in tracking the object in subsequent frames. We first infer 3D features for object points visible in the first frame. For subsequent timesteps, we predict the amodal *complete* 3D feature cloud of the scene from a single view by querying our model in a set of visible and invisible 3D point locations. We found this uniform random sampling of query point set strategy to suffice for our tasks, though more elaborate strategies that focus on querying points close to object surfaces can be used, such as Marching Cubes [96], which has been used by previous 3D implicit shape function models for learning object occupancy to obtain a watertight mesh [44, 118, 124]. Given the object feature point cloud and the amodal scene feature cloud, we obtain a rigid body transformation (3D rotation and translation) by solving the orthogonal Procrustes problem [142], and use RANSAC to find the transformation that satisfies most inliers as shown in Figure 4.2, where inlier point matches are obtained by thresholding inner products at 0 between the object 3D point features and 3D point features of the target scene. We find the object 3D box in the target scene by warping the box b_0 using the obtained transformation. Note that we do not restrict our model to search locally only for a matching object, but search over the whole scene which makes the task harder. Such object matching across frame gaps is important for re-detecting an object after occlusions. It is also a good test for the discriminability of the features, since they are not assisted by temporal continuity or local search regions.

Evaluation and comparison with baselines We compare CoCoNets against the following baselines (we use names indicative to the differences and characteristics): i) 3DVoxContrast of Harley *et al.*[55] , ii) 3DPointContrast of Xie *et al.*[192] iii) 2.5DDenseObjectNets of Florence *et al.* [37], which uses triangulation supervision to train 2D deep feature maps, then lifts them to 3D point feature clouds using the available depth map.

We follow the experimental setup of Harley *et al.*[55], which uses video sequences of 10 timesteps. For 3DVoxContrast, we first use multiview RGB-D data with known egomotion to train a model using 3D contrastive losses. Upon training, 3DVoxContrast maps a single RGB-D image to a discrete 3D feature map. Then, 3DVoxContrast uses this model for tracking by solving orthogonal procrustes problem with RANSAC between voxel features as opposed to point features. For 3DPointContrast, upon training the model, we compute RANSAC on the cross-correlation scores between point features from the visible 3D points on the object at frame 0 and the *visible*

Method	CARLA	KITTI
CoCoNets (ours)	**0.61**	**0.54**
3DPointContrast [192]	0.55	0.48
3DVoxContrast [55]	0.37	0.23
2.5DDenseObjectNets [37]	0.24	0.19

Table 4.1: **3D object tracking IOU after 10 timesteps.**

Method	**mAP@IOU**			
	0.25	0.30	0.40	0.50
VoteNet [127]	0.32	0.26	0.24	0.20
CoCoNet-*VoteNet*(Ours)	**0.51**	**0.47**	**0.41**	**0.32**

Table 4.2: **Comparing VoteNet with CoCoNet-*VoteNet* on mAP for different IOUs**. Pre-training the 3D point cloud features with CoCoNets significantly boosts performance.

point cloud at each subsequent frame. Our method also considers amodal completed point cloud at each frame. For 2.5DDenseObjectNets, upon training the model, we apply RANSAC much the same way as in the 3DPointContrast model.

We show quantitative tracking results for our model and baselines on the CARLA and KITTI datasets in Table 4.1. Figure 4.3 visualizes 3D object trajectories over time from an overhead view. Note that we do not use any locality search, i.e., each model searches across the whole target frame for the configuration of the object. Our model achieves superior performance on 3D object tracking than all the baselines. It dramatically outperforms 3DVoxContrast, which highlights the importance of high spatial resolution for 3D feature learning and correspondence. It also outperforms 3DPointContrast, which highlights the importance of amodal completion supported by our model. Lastly, all 3D methods outperform 2.5DDenseObjectNets which highlights the effectiveness of learning features in a metric 3D feature space.

4.4.2 Supervised 3D object detection in point clouds

Setup We test the 3D feature representations obtained by CoCoNets in their ability to boost the performance of a supervised 3D object detector when used as initialization. We use VoteNet [127], a state-of-the-art 3D object detector that combines the classic ideas of Hough voting in a deep neural network architecture that computes point features, uses them to vote for locations and sizes of the 3D object boxes, and

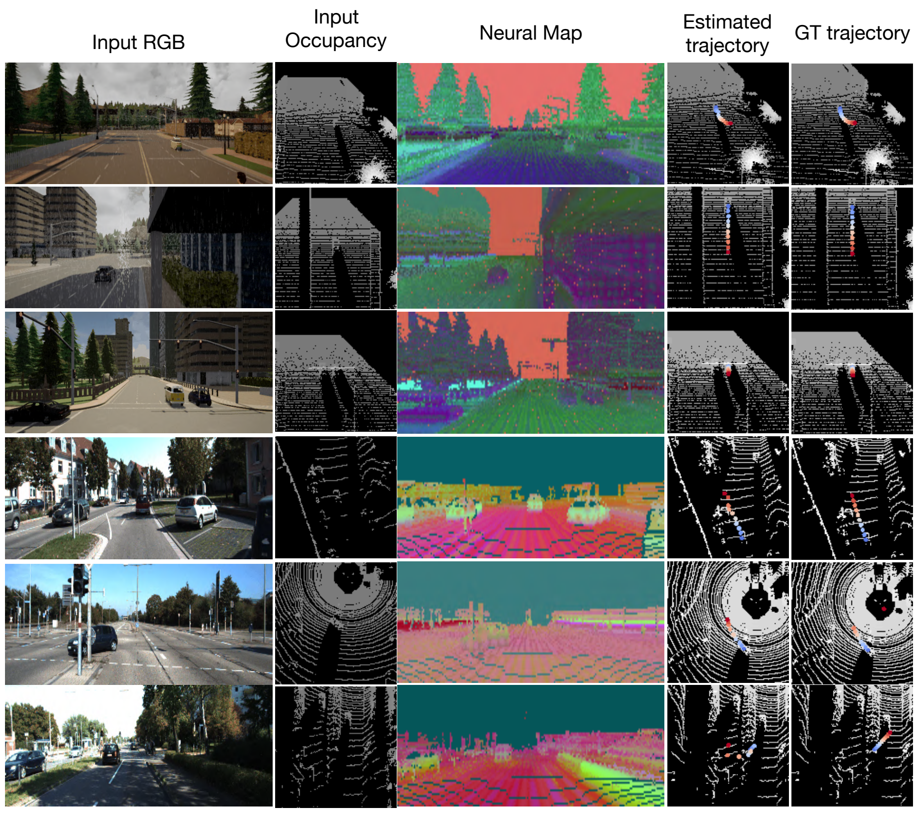

Figure 4.3: **Self-supervised 3D object tracking**. In the 1st and 2nd column we visualize the RGB and depth from the first frame, which is given as input to our model, along with a 3D box specifying the object to track. In the 3rd column we visualize our inferred point features by projecting them to the same RGB image and then doing PCA compression. In the last 2 columns we show the estimated and ground truth trajectories. The top three rows show our results on CARLA; the bottom three rows show our KITTI results.

aggregates the votes to propose 3D object bounding boxes with associated confidence, in an end-to-end differentiable framework. We modify VoteNet [127] by replacing their point cloud feature extractor backbone with a trained CoCoNet; we will refer to this as CoCoNet-*VoteNet*. The original backbone in VoteNet uses PointNet++ [126] to compute point features. Both the VoteNet backbone and CoCoNet-*VoteNet* backbone sample 1024 points from the dense point cloud using farthest point sampling. The subsequent voting and proposal modules of VoteNet then operate on these sampled point features to make the final predictions. In both cases, upon initialization, the 3D point features are trained end-to-end supervised for the 3D object detection task. We show below that our pre-training gives a significant boost to 3D object detection performance.

Method	cross-object	cross-view
3DVoxContrast [55]	**0.18**	0.21
3DPointContrast [192]	0.09	0.14
CoCoNets (Ours)	**0.18**	**0.58**

Table 4.3: **Cross-object and cross-view 3D alignment accuracies in Shapenet dataset** (mean over 4 classes: Aeroplane, Mug, Car, Chair).

Evaluation and comparison with baselines We evaluate both models on the CARLA dataset in the task of predicting 3D axis aligned car bounding boxes. We show the mean average precision (mAP) scores for both the models for different IoU thresholds in Table 4.2. CoCoNet-*VoteNet*outperforms VoteNet across all IoU thresholds.

4.4.3 Cross-view and cross-object 3D alignment

Setup We evaluate the correspondability of our 3D feature representations by testing how well they can align different views of the same instance, and different instances that belong to the same object category. Given the amodal 3D feature point clouds extracted for two objects in random viewpoints, we use orthogonal Procrustes problem with RANSAC to estimate the 3D rigid body transformation that aligns them, similar to tracking a car in Section 4.4.1. Estimating fine grained correspondences between objects, once their rigid alignment has been estimated, is obtained using iterative closest point method [2].

Evaluation and comparison with baselines We investigate two different setups: i) Given two viewpoints of the same object, our goal is to estimate the relative 3D rotation between them. ii) Given two objects of the same category in random viewpoints, our goal is to estimate a rigid 3d alignment between them. We compare our model against baselines 3DVoxContrast and 3DPointContrast, for which we use RANSAC on voxel and point features, respectively, to estimate a 3D rigid transformation for the two objects.

The inferred 3D alignment is considered correct if each of the yaw, roll, and pitch angles are within $10°$ of their respective ground truth values. In Table 4.3 we show the cross-view (same-object) and cross-object 3D alignment accuracy for our model and the baselines. Our model performs the same as 3DVoxContrast on cross-object alignment, and outperforms both models on cross-view alignment. Note that cross-object alignment is much harder than cross-view for all three models.

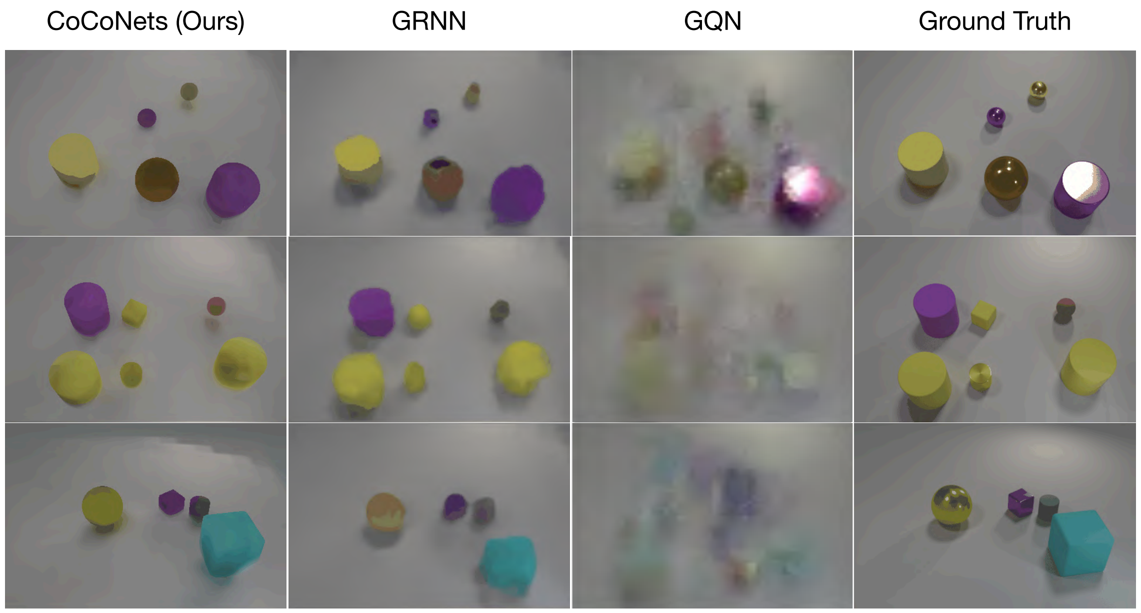

Figure 4.4: **Novel view prediction.** We compare CoCoNets, GQN [36], and GRNN [171] on the CLEVR [72] dataset.

4.4.4 Generalization in view prediction

We evaluate CoCoNets-$OccRGB$ in its ability to predict plausible images in scenes with novel number of objects, novel object appearance, and novel arrangements. We compare against state-of-the-art view prediction models. We further evaluate its ability for 3D occupancy prediction and completion.

We compare CoCoNets in RGB prediction against generative query network (GQN) [36] and geometry-aware recurrent networks (GRNN) [171]. We adapted GQN code to use posed RGB-D images as input. GQN uses a 2D latent feature space to encode the RGB-D input image, and GRNN uses a discrete latent 3D feature map of the scene. Our model uses a grid of functions and does not suffer from resolution limitations of the 3D feature map.

CoCoNets predicts RGB images for target viewpoints as described in Sec. 4.3.1. Figure 4.4 shows the view generation results for CoCoNets, GQN, and GRNN.

4.4.5 Limitations

CoCoNets might be extended to operate without the availability of depth maps at either train or test time using a differential rendering module [149], as shown in Figure 4.5.

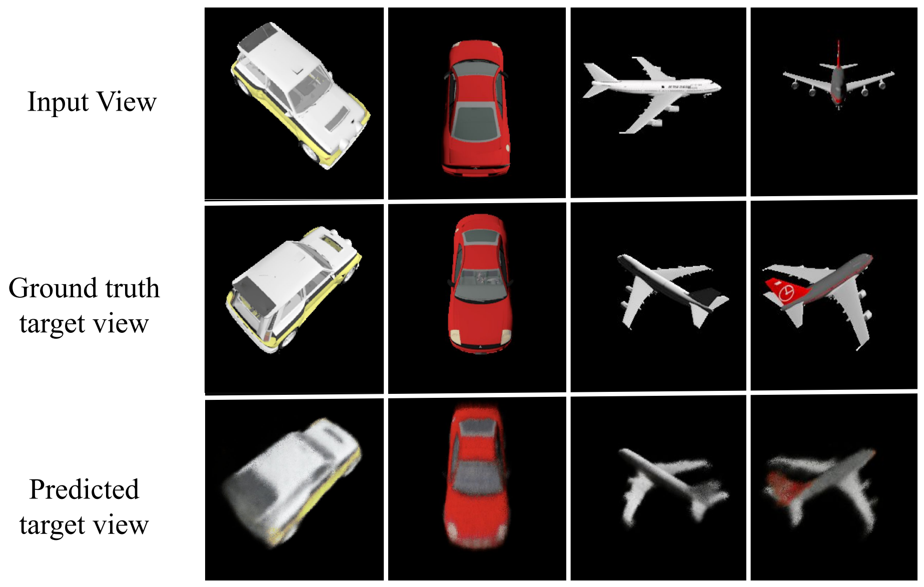

Figure 4.5: **RGB view regression** on the ShapeNet dataset using CoCoNets^{NoDepthRGB}.

4.5 Conclusion

We present a method for learning 3D visual feature representations by self-supervised view and depth prediction from posed RGB and RGB-D images. Our networks lift input 2.5D images of objects and scenes into latent thee-dimensional function grids, which can be decoded to infinite-resolution 3D occupancy and 3D feature predictions of the object or scene. Our networks are trained by predicting views using a contrastive mutual information maximization objective. We evaluate the emergent 3D visual feature representations in 3D object tracking in dynamic scenes and in cross-view and cross-object alignment estimation. We empirically demonstrate that the features are semantically meaningful and outperform popular point-based supervision [192] which does not consider 3D completion, and discrete voxel 3D latent feature maps of previous works [54, 55] which are limited by the spatial resolution of the 3D feature grid. Moreover, our models can better generalize to novel scenes with unseen number and appearance of objects than networks that do not encode 3D structure [36] or do not incorporate 3D convolutional modules [149]. They make a step towards self-supervised learning of amodal 3D feature representations, which we show are useful for 3D object tracking and correspondence. Avenues for future work include learning such representations directly from dynamic videos, and relaxing the egomotion and depth supervision.

Chapter 5

Self-Supervised Pointcloud Completion Improves Object Detection

5.1 Introduction

Object detection in point clouds is made difficult partly by the sparsity of the data collected by the sensor. Un-occluded objects a few meters from a modern Lidar unit may have hundreds of point returns, and these objects are typically detected with high accuracy, but objects that are far away, or partially occluded, may only have a few points describing them, and this sparsity of signal leads to missed detections.

Previous work has shown that if new points are added at the surfaces of objects, they can be detected more easily. Yin et al. [202] tackled this problem directly, and proposed to add points at object surfaces with the help of a 2D (image-based) detector. In contrast, Yi et al. [200] added points without detecting objects, by training a model to take a real pointcloud as input and produce a denser version as output, supervised by a Poisson surface reconstruction of the scene, using data aggregated across multiple views. A downside of multi-view aggregation is that it assumes the scene is static – moving objects produce points across their entire range of motion, and these need to be carefully eliminated to avoid artifacts in the reconstruction. Both of these methods show an improvement in detection performance, after the densification. We follow these works' strategy of isolating the completion problem from the rest of the vision pipeline, but we propose a method that requires neither detection, nor multi-view aggregation.

We propose to learn pointcloud densification self-supervised. At training time, we drop points from the pointcloud to form the model's input, and treat the original pointcloud as the model's target. At test time, the real pointcloud becomes the

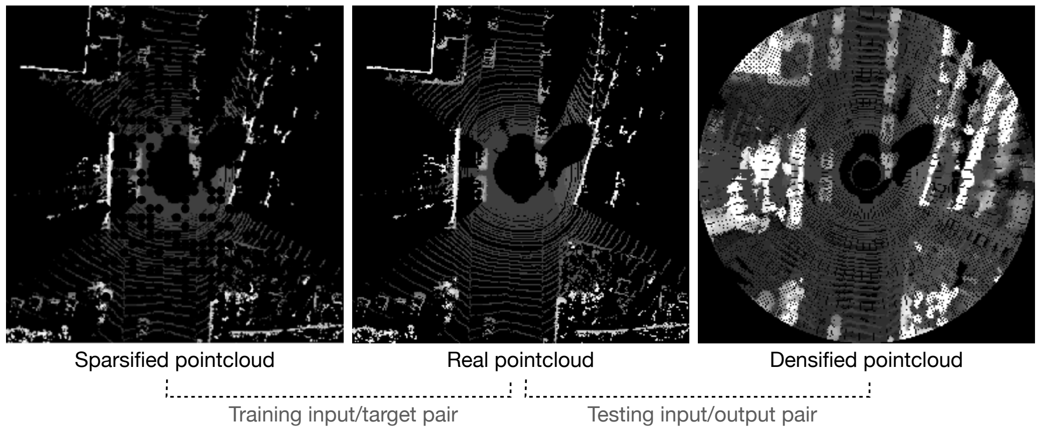

Figure 5.1: **Inputs and outputs, at training time and test time.** At training time, we drop points from the original pointcloud, and train the model to add points, using sparse supervision computed from the original points. At test time, we provide the model with the true pointcloud as input, and it hallucinates points at likely locations.

model's input, and the model generates a densified output, for which we do not have a ground-truth target. Figure 5.1 illustrates this strategy. A similar approach was recently proposed for indoor scene reconstruction [27]. In that problem setting, multi-view reconstruction of the scene geometry from depth maps leaves "holes" that need filling; the authors proposed to randomly drop frames from the reconstruction, creating additional holes for which they have ground truth, and thereby learn a prior that can fill the true holes. The current work can be seen as a more extreme implementation of the same core idea, but dropping data from a single timestep of capture, rather than dropping frames from a multi-frame capture.

Importantly, we do not prevent the model from "hallucinating" points outside the sensed set, when it is learning to complete the scene. While we treat the entire observed pointcloud as "positives" for a binary cross entropy loss, we do not treat all other locations as "negatives", but instead compute sparse negative labels by tracing the freespace ray between each point and the sensor origin, and further discard voxels that are ambiguous according to the ray trace, with a simple heuristic. This conservative supervision strategy ensures that we do not provide the model with erroneous labels (up to sensor noise), and only use the "known" part of the scene to supervise the completion.

In principle, any model can be trained for this completion task, but given the sparse supervision and the train-test gap, it is critical to avoid overfitting. Toward this goal, we take into account the fact that the pointcloud is collected by a spinning Lidar unit, and we propose a model which is rotation-invariant along the sensor's spin axis, and sensitive to distance along each ray. In other words, we force the model to use the same completion prior at every degree of rotation, but allow different strategies across the length of a ray. These choices reflect the properties of the sparsity pattern: when inference is aligned with the ray direction, locations *before* an observed point are reliably freespace, and locations *beyond* an observed point are either free or occupied; furthermore, observations *near* the camera are more likely to be dense, requiring less inpainting, and observations *far* from the camera are likely to be sparse, requiring more inpainting. This knowledge is not available, for example, to a translation-equivariant model such as a standard convolutional network, because such a model would not be informed of which locations in the space are "before" or "beyond" an observed point, or "near" or "far" from the sensor.

We demonstrate our approach in the KITTI urban scenes benchmark [43], and show that when we train and test our computer vision models on the densified pointclouds, they achieve higher accuracy in detection and semantic segmentation, compared to models trained and tested on real pointclouds. We also observe qualitatively that our completed pointclouds contain rounded convex shapes for the objects, similar to the true shapes, apparently encoding a prior of convexity [73].

5.2 Related Work

5.2.1 Pointcloud and occupancy completion

As mentioned in the introduction, prior work has tackled the pointcloud completion problem under slightly more constrained settings than considered here, with good results. Yin et al. [202] add points near object surfaces, by detecting and segmenting objects in 2D, sampling pixel coordinates within the mask, and using depth values retrieved from nearest neighbors in the projected pointcloud to create new 3D points. Our method instead adds points everywhere (including on non-object surfaces, and within object interiors), and does so unsupervised, without the use of a detector. Similar to our method, Yi et al. [200] added points everywhere, without detecting objects, but they trained their model with a dense reconstruction of the scene, created by first aggregating pointclouds from multiple timesteps and then fitting a surface mesh with a Poisson method. A dense supervision target is indeed ideal, but requires either a static scene assumption, or careful (possibly manual or heuristic-based) removal of the moving objects from the aggregated pointcloud, so as to avoid large artifacts in the surface reconstruction. In our method, we use data from individual Lidar sweeps, without aggregating, and thereby avoid the difficulty of moving objects (ignoring motion within a single sweep, which is 10hz). For indoor scenes and for objects, recent methods fit implicit functions to the points in the input 3D shape, and leverage priors learned either within a sample or across samples to estimate "complete" shapes and scenes [17, 27, 103, 115, 118]. All of these works arguably share our strategy of creating an artificial sparse-to-dense problem, forcing the model to learn a prior that fills gaps accurately, but exclusive to the context of object-centric or room-centric reconstruction settings, and without targeting a downstream semantic task. In these settings, relatively complete scans can be obtained by capturing all surfaces with a moving depth sensor, while the room/object itself is static, and later solving for the camera poses. In our setup, we use a single timestep of capture as our densest "target", and therefore do not assume the scene is static, and do not need to solve for sensor poses.

5.2.2 Super-resolution

Our strategy of "sparsifying" the available pointclouds to form input-target pairs for a "densification" model is essentially the same strategy used to train neural networks for image super-resolution [183]. These methods downsample the available images to form input-target pairs for an upsampling model. We note also that super-resolution has been shown helpful to downstream image-based vision tasks [28], which is in line with our observation that pointcloud densification is helpful to downstream pointcloud-based vision tasks.

5.2.3 Managing invariances for pointcloud data

In pointcloud processing literature, there is great interest and active search for architectures that have desirable invariances built in [24, 125]. Of particular interest here are rotation-equivariant pointcloud networks [15, 29, 89], which ensure that as the input to the model rotates, the outputs rotate by the same amount. This allows the model to take into account *invariances* across the axes made equivariant, by using the same operation across these dimensions. Note for example that the convolution operator, which is translation equivariant, exploits invariance of texture and shape across the axes of translation, by repeating the use of a 2D kernel across 2D space. The particular equivariance explored in this work is along the spin axis of the sensor, which can be called yaw equivariance. This has been achieved before with spherical projection (a.k.a. range images) followed by 2D convolution [188], as well as cylindrical or polar 3D convolutions [130, 205, 207]. Our model is similar in spirit to these, but *only* has yaw equivariance, whereas these other works typically use at least one more axis of equivariance, such as along the vertical dimension (for the 2D and 3D works), or along the forward direction (for the 3D works). Our model is instead *sensitive* (i.e., variant) to changes along the vertical and forward axes, with the reason being that the scene completion problem itself varies along these axes (at least for urban scenes sensed by Lidar): lower positions on the vertical axis are

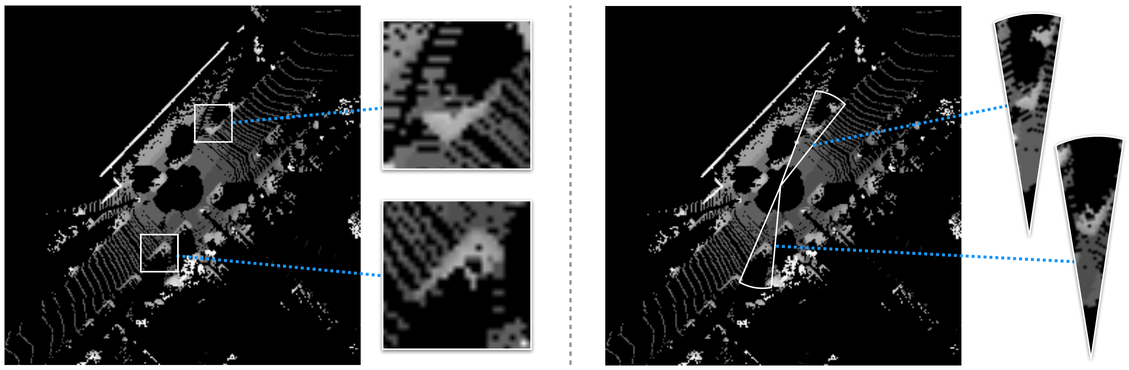

Figure 5.2: **Pointcloud completion is not translation-invariant, but rather rotation-invariant.** Standard convolutional models assume a translational invariance in the task, but this is not entirely appropriate for pointclouds captured by Lidar. Square crops of cars at different translational positions, as shown on the left, actually require *different* computation to complete their shapes. In contrast, for pie slices at different angular positions, as shown on the right, the *same* strategy can be used across crops. This is the invariance exploited by our model.

more likely ground instead of air, and distant positions on the forward axis require integrating more information from context compared to close-up positions.

5.3 Pointcloud completion model

5.3.1 Setup and overview

Our model takes as input a Lidar pointcloud with an arbitrary number of points, and produces a voxelized cylinder of occupancy estimates, of a pre-specified metric size and resolution, centered at the sensor.

We name our method a "pie slice" model, because it divides the cylinder into sectors – i.e., pie slices – and solves for each of the sectors independently. This means, for each sector of the cylinder, it orients the pointcloud so that the considered slice is at zero degrees (i.e., the canonical position), then estimates occupancy for voxels within that slice, and finally orients the outputs to their appropriate position in the cylinder.

The key motivation for dividing the scene into pie slices is illustrated in Figure 5.2. Since the pointcloud is captured by a spinning Lidar unit, the occlusion pattern is the same across rotations about the spin axis. By dealing with pie-slice crops in our model, instead of, for example, square crops extracted translationally (as done in standard convolution), we provide the model with an accurate inductive bias for the task at hand.

The model has a few major hyperparameters. First is the number of sectors in the cylinder (i.e., number of pie slices), which we denote with K. This can be equivalently defined by the number of degrees in a sector. More sectors means more yaw equivariance, but also less computation shared between slices. In principle, a model like this can be made pitch-equivariant as well, by dividing the pie slices into bins according to pitch angle, but for the current setting this seems inappropriate, due to the fact that different shapes/surfaces appear at different pitch angles.[1]

The second important hyperparameter is the metric span of the cylinder, defined by a diameter (or radius) and height. This is chosen to be the metric span of the downstream task, i.e., the span we would like the detector to operate in.

The third hyperparameter is the number and distribution of estimates within the canonical pie slice. Depending on the task, different distributions may be appropriate, such as denser sampling near the camera and sparser sampling far away. In our experiments we use approximately-uniform sampling, defined by sampling M farthest points on the floor of the pie slice, and for each of these 2D coordinates, using a constant vertical resolution of H, so that for each coordinate we produce a "pillar" of occupancy estimates.

5.3.2 Architecture

Our model takes as input a pointcloud with N points. We assume that the pointcloud is captured by a Lidar unit, and is therefore centered at the sensor origin. As output, the model produces an $M \times H$ matrix, containing M "pillars" of occupancy estimates, each of height H. Each of the M pillars corresponds to a discrete index at the base of the voxelized cylinder, within the canonical pie slice.

The main model is preceded by a "rotate" step, which puts the slice of interest at 0 degrees (the canonical orientation), and succeeded by an "un-rotate" step, which puts the output into alignment with the original input.

The architecture of our within-slice model is illustrated in Figure 5.3. The architecture is inspired by Perceiver models [67, 68], which use a learnable set of vectors to cross-attend into high-dimensional input, producing a flexible representation with dimensionality suitable for further processing. After rotating the pointcloud to the canonical orientation, we attach sinusoidal position embeddings to the points, and cross-attend with M learnable vectors with channel dimension C, using the standard query-key-value attention operator [174], producing a new matrix shaped $M \times C$, which is a compression of the input. Here, while standard Perceiver models use self-attention between the M "tokens" to process them, we instead choose to use an MLP-Mixer [166], as this led to faster convergence than self-attention in our

[1]We have also experimented with a per-ray model, where essentially $K \approx \infty$ and the M estimates lie along a single line, but it was too slow to be practical for study.

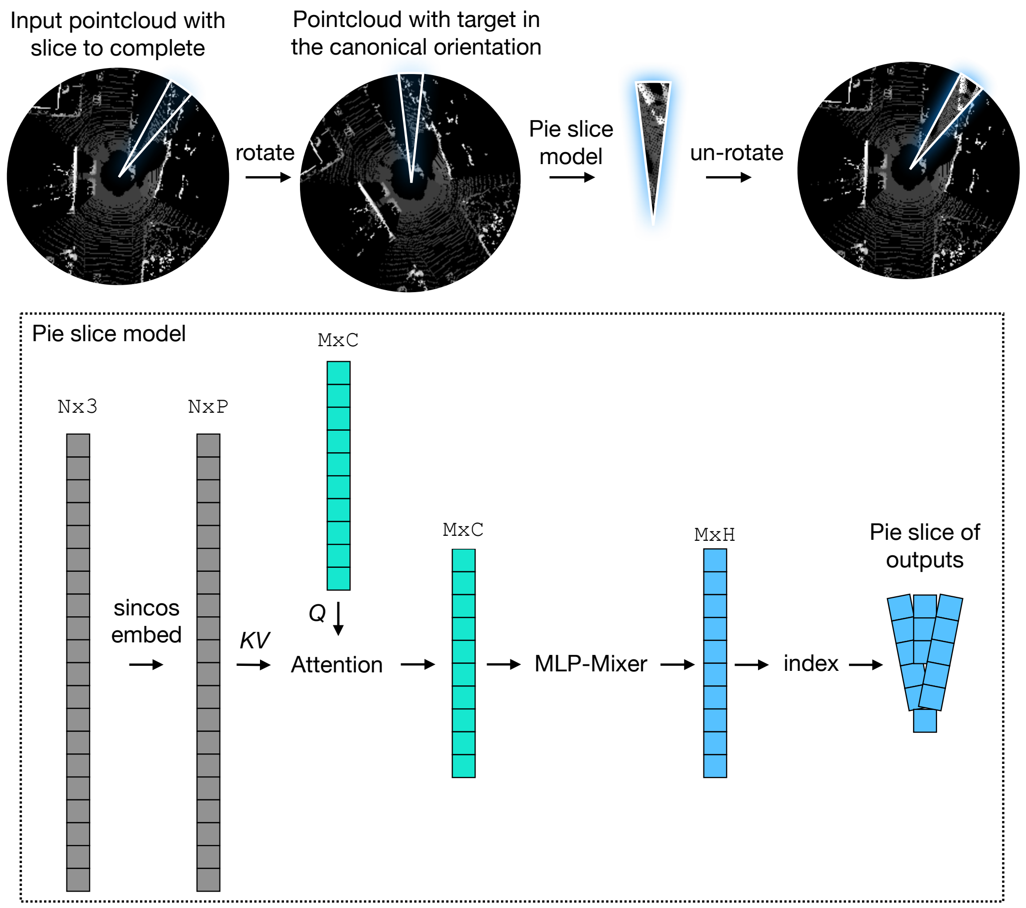

Figure 5.3: **Architecture of our "pie slice" model.** Given a pointcloud as input, the model considers the span of only a few degrees at a time (i.e., one pie slice). The model begins by orienting the full pointcloud so that the slice of interest is at 0 degrees, then tokenizes the pointcloud by attending with a learnable set of M latents, then processes these tokens with an MLP-Mixer model, which produces a set of occupancy pillars (of height H) that we can rearrange at the base of the pie slice with a pre-defined indexing operation. Finally, these outputs are "un-rotated" to put them in their correct place in the full reconstruction.

experiments. The MLP-Mixer ends with a linear layer, producing H channels per token, leading to output that we can use as occupancy logits. Note that the M learnable latents have *fixed* coordinates in the canonical pie slice, and so these tokens can each learn to attend to their relevant region of the pointcloud, and information is subsequently propagated between tokens by the MLP-Mixer, allowing the model to use multi-token context to fill gaps.

5.3.3 Training

At training time, the input to our model is a random fraction of the true pointcloud. Note that if we provided the full pointcloud as input, the model would simply estimate occupancy at all provided points and no more; providing a subset at training time forces the model to estimate additional occupied space.

We create sparse occupancy ground-truth by voxelizing the full input pointcloud. We create sparse freespace ground-truth by tracing the ray from each point to the sensor origin, but these rays must be filtered slightly before voxelization.

As illustrated in Figure 5.4, simply taking all voxels intersected by freespace rays leads to some erroneous ground truth, particularly with beams which meet their surfaces a very acute angle. For example, when tracing the ray for a ground point detected far away from the sensor, the ray may cross through several voxels which are actually at the same height as the sensed "ground" voxel, and therefore it is unlikely that these are truly freespace. We filter these with a heuristic strategy, which simply asks for the ray to travel a certain threshold distance along any axis direction before we take its voxel intersections as freespace. Since this error is essentially a voxelization error, we may choose this threshold to coincide with the length of a voxel. We qualitatively note that without this strategy, the densified pointclouds closely reflect the sampling pattern of the original sensor.

To supervise the model, we need to retrieve labels that correspond to the predicted pillars. To do this, we use the 3D coordinates of all the estimates to compute corresponding voxel indices in the ground truth, and supervise the subset of estimates which have ground truth available.

We supervise the model with a binary cross entropy loss,

$$\mathcal{L} = \sum_i y_i \log \hat{y_i} + (1 - y_i) \log(1 - \hat{y_i}), \tag{5.1}$$

where $\hat{y}$ is the labels, and y is the set of estimated logits, and the sum iterates over indices with valid labels. To prevent the model from learning an overall bias between freespace and occupied space, we compute this loss in two averages: within freespace labels, and within occupied labels, yielding a balanced loss.

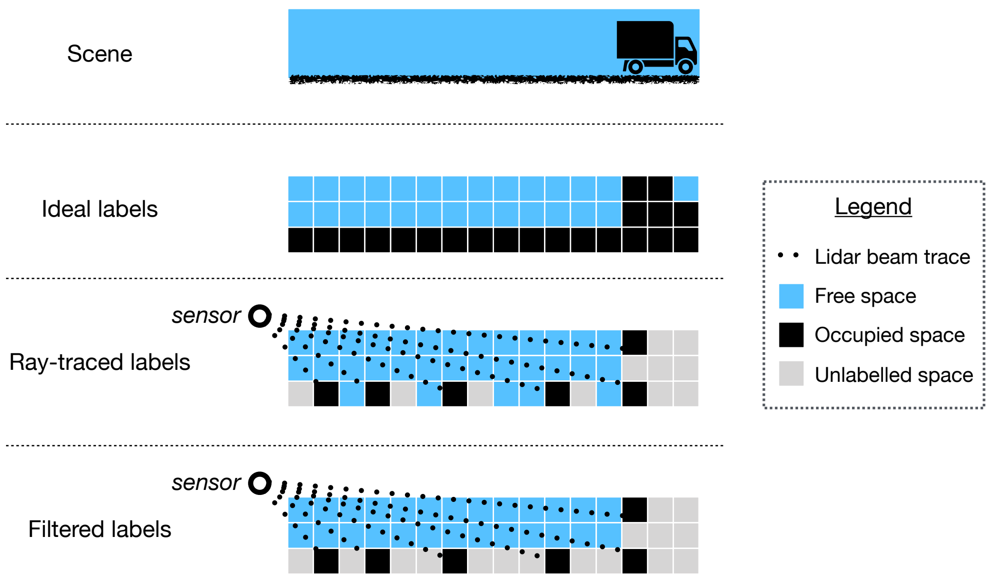

Figure 5.4: **Detailed view of label creation strategy.** We take all voxels which contain a Lidar return as occupied space. For freespace, we obtain most of the labels by taking non-occupied voxels intersected by lidar beams, but this also yields some erroneous labels. We filter out these errors with a heuristic strategy, described in the text.

5.3.4 Implementation details

At training time, we shuffle the pointcloud and take a random $\frac{1}{16}$ of the points to use as input. At test time, we use the entire pointcloud.

We encode the input pointcloud with sinusoidal position embeddings with 64 dimensions for each axis, and concatenate these embeddings with the un-normalized 3D coordinates, yielding 195 channels per point.

The learnable tokens used for cross attention have 128 channels each. The MLP-Mixer has 12 blocks, with latent dimension 128, and expansion factor 4 in each of the blocks. The last layer has channel dimension H.

We set M, the number of pillars (and the number of cross-attention tokens), to be 768. We set the angle of the pie slices to be 5 degrees, which translates to $K = 72$. At a square resolution of 256×256 around the circle, this correspond to nearly 100% density within each pie slice. At training time, we sample 128 slices at random degrees within a single pointcloud to form a batch.

At test time, for each pie slice, we collect coordinates whose occupancy estimate (after sigmoid activation) is above a threshold of 0.8, and treat these coordinates as new candidate "points". We discard candidates which intersect freespace rays computed from the original pointcloud, and finally concatenate the rest to the set of original points. This concatenation serves as input to our downstream vision models.

Our model takes approximately 0.1 seconds per pie slice. The model is extremely memory-heavy, due to the large MLP-Mixer, and due to the first cross-attention layer (which is especially expensive at test time since we no longer sub-sample the pointcloud). Producing the full cylinder of estimates simultaneously requires four V100 GPUs in parallel.

5.4 Experiments

We train and test our model in the KITTI [43] urban scenes dataset. We use annotations from the "vehicle" class in the detection benchmark.

While the model training is focused entirely on pointcloud completion, we are actually interested to see if the densified pointclouds lead to improved performance in downstream tasks, such as segmentation and detection.

Note that after densifying the pointcloud, it becomes more appropriate to use standard translational convolutions for the vision model, because the data no longer reflects the sampling pattern of the sensor, and instead is approximately uniform. We therefore follow recent related work for pointcloud vision architectures, and use a CenterPoint as our base architecture [201].

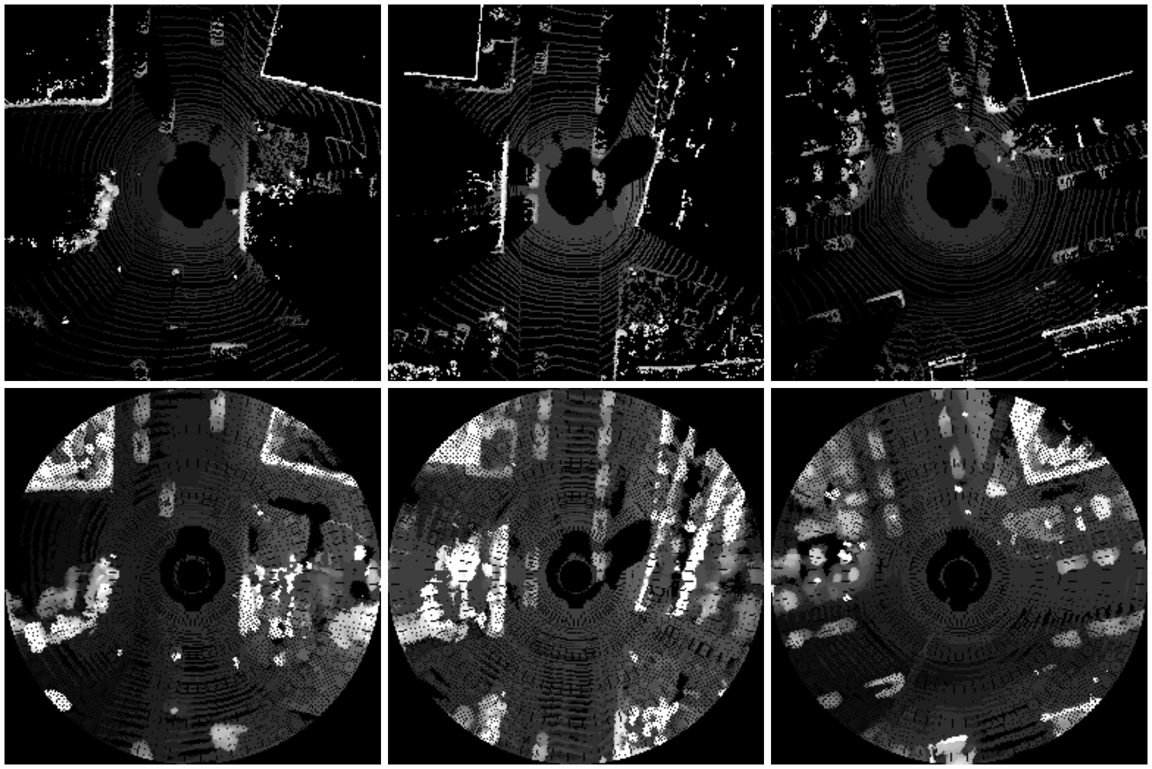

Figure 5.5: **Pointcloud inputs (top), and corresponding completions (bottom).** Our model often converts objects into full rounded convex shapes.

5.4.1 Qualitative results

We first inspect the completed pointclouds qualitatively. Figure 5.5 shows a sample of the outputs, in a bird's eye view (top down). In the project website, we provide a continuous video of these visualizations, across a sequence from the dataset.

We observe that at the object regions, our completed pointclouds show rounded convex shapes, similar to the shapes of the real objects. This is interesting, because the model never receives supervision for "full" objects, and yet the fullness emerges. This can be interpreted as a learned preference for convexity [73]. One possible reason for this emergence is that the model is implicitly recognizing the objects at some level, and it is transferring shape information from close-up well-observed cars to the far-away (or partially occluded) poorly-observed cars. A perhaps simpler explanation is that the model occasionally receives supervision to encourage a convex shape, since freespace rays may cut through flared-out shape estimates, but it will *never* receive supervision to encourage a concave shape, because concavities on the occluded side do not contribute to the set of freespace rays, much in the same way that concavities do not contribute to visual hulls [26].

Input	Detection mAP
Pointclouds from pie slice model (ours)	0.67
Pointclouds from convolutional model [200]	0.59
Real pointclouds	0.57

Table 5.1: **Detection mean average precision (mAP)** in the KITTI validation set.

Input	Segmentation IOU
Pointclouds from pie slice model (ours)	0.66
Pointclouds from convolutional model [200]	0.63
Real pointclouds	0.62

Table 5.2: **Segmentation intersection over union (IOU)** in the KITTI validation set.

5.4.2 Quantitative results

We compare our completion model against a standard convolutional model, similar to the one used by Yi et al. [200], on the downstream computer vision tasks. We are interested to see if the completions improve performance, when we use the completed pointclouds instead of the original pointclouds.

Table 5.1 shows our results in detection, evaluated in mean average precision (mAP) at an intersection-over-union threshold of 0.5. Our model delivers an absolute boost of 0.10 mAP over the baseline, while the convolutional model shows an improvement of 0.02. Inspecting the convolutional model's output, we observe that it has not learned to effectively inpaint the scene, and instead mostly copies the input to the output and adds relatively few points. Lowering the convolutional model's threshold for point addition often worsens results, since it then yields more incorrect points.

Table 5.2 shows the same experiment for pointcloud segmentation. Here, the model is segmenting the entire (potentially densified) pointcloud, but we evaluate only on "original" points. In this evaluation the models perform more similarly, but the pie slice model still slightly outperforms the baselines.

5.4.3 Limitations

We have experimented also in the Nuscenes dataset [11], with so-far negative results. We believe this is due to the lower density of points in that dataset, due to the 32-beam sensor, compared to the 64-beam sensor as in KITTI. We plan to evaluate also in the Waymo dataset [157], which has 64 beams like KITTI does.

Our model does not deal with the temporal dimension of the problem. Modern detectors typically use multi-sweep data as input [201], and this improves results over using single-sweep input, despite the fact that moving cars leave artifacts. Integrating information across time is a clear avenue for future work.

5.5 Conclusion

We propose a pointcloud completion model, which is self-supervised from single-sweep Lidar data, and exploits an rotation-invariance of the completion task caused by the Lidar sensor itself. We show that the completions appear to contain closed object contours, suggesting that some concept of objectness or at least convexity emerged through the self-supervised training. We further show that the completions lead to an improvement in downstream vision performance, as evaluated in detection and segmentation accuracy. Given that this self-supervision is “free” with Lidar data, and given that large-scale collection of unlabelled Lidar data is relatively easy (and perhaps done already by several car companies), we think this is a scalable way to improve computer vision performance for autonomous vehicles in the real world.

Chapter 6

A Baseline for Neural 3D Mapping, Without Depth or Egomotion

6.1 Introduction

There is great interest in building 3D-aware perception systems for autonomous vehicles, under various sensor platforms, which typically constitute multiple RGB cameras, multiple radar units, and optionally a Lidar unit. Vision systems which use Lidar data in combination with camera data are currently the most accurate [201, 202], but Lidar sensors are arguably too expensive for large-scale use, so attaining similar accuracy without the use of Lidar is an attractive research goal.

Recent work in this area has developed sophisticated methods for bird's eye view inference from multi-view RGB data, from one or more timesteps, with significant innovations in the techniques for "lifting" features from the 2D image plane(s) to 3D. In particular, recent methods focus on producing a "bird's eye view" representation of the 3D space around the vehicle, instead of a fully-3D one, because this suffices to capture the information required for driving-related tasks, such as navigation, obstacle detection, and moving-obstacle forecasting. In this 2D-to-2D mapping task, (mapping the image plane to the ground plane,) prior work has proposed using a homography to warp features directly onto the ground [12], using depth estimates to place features at their approximate 3D locations [119, 144], using MLPs with various geometric biases [49, 59, 138], and using geometry-aware transformers [137]. This series of innovations has of course been accompanied by rising accuracy in the considered tasks, such as vehicle and lane detection.

In this paper, we propose a simple baseline model, where the "lifting" step is a non-parametric 3D unprojection, i.e., tiling features across their corresponding 3D rays, as done in some earlier 3D scene representation work (e.g., [54]). When this simple method is paired with appropriate data augmentation and high-resolution input, it can match the performance of state-of-the-art models. We further show that if we incorporate the available data from radar, which is typically either ignored or discarded due to its sparsity and noise, we exceed the state-of-the-art. This is a pleasant surprise, because prior work has attempted to fuse information from RGB and radar with mostly negative results [59]. In light of our experiments, it appears that end-to-end training is the key to effective RGB+radar fusion, and enables the metric signal embedded in radar to contribute a large performance boost.

6.2 Related Work

A major differentiator in prior work is the precise operator for "lifting" 2D perspective-view features to 3D, or directly to the ground plane.

Parameter-free unprojection: This strategy, pursued in a variety of object and scene representation models [18, 54, 148], uses the camera geometry to define a mapping between voxels to their projected coordinates, and collects features by bilinearly sampling at the projected coordinates. This places each image feature into multiple 3D coordinates, essentially tiling the feature along the ray's extent in the volume. This method of lifting is not typically used in bird's eye view semantic tasks.

Depth-based unprojection: Several works estimate per-pixel depth with a monocular depth estimator, either pre-trained for depth estimation [92, 144] or trained simply for the end-task [64, 119, 178], and used the depth to place features at their estimated 3D locations. This is an effective strategy, but note that if the depth estimation is perfect, it will only place "truck" features at the front visible surface of the truck, rather than fill the entire truck volume with features. We believe this detail is one reason that naive unprojection performs competitively with depth-based unprojection.

Homography-based unprojection: Some works estimate the ground plane instead of per-pixel depth, and use the homography that relates the image to the ground to create a warp [12, 93], transferring the features from one plane to another. This operation tends to produce poor results when the scene itself is non-planar (e.g., tall objects get spread out over a wide area).

MLP-based unprojection: A popular approach is to convert a vertical-axis strip of image features to a forward-axis strip of ground-plane features, with an MLP [59, 88, 114]. An important detail here is that the initial ground-plane features are considered aligned with the camera frustum, and they are therefore warped into a rectilinear space using the camera intrinsics. Some works in this category use multiple MLPs, dedicated to different scales [135, 138], or to different categories [49]. As this MLP is parameter-heavy, Yang et al. [199] propose a cycle-consistency loss (mapping backward to the image-plane features) to help regularize it.

Geometry-aware transformer-like models: An exciting new trend is to

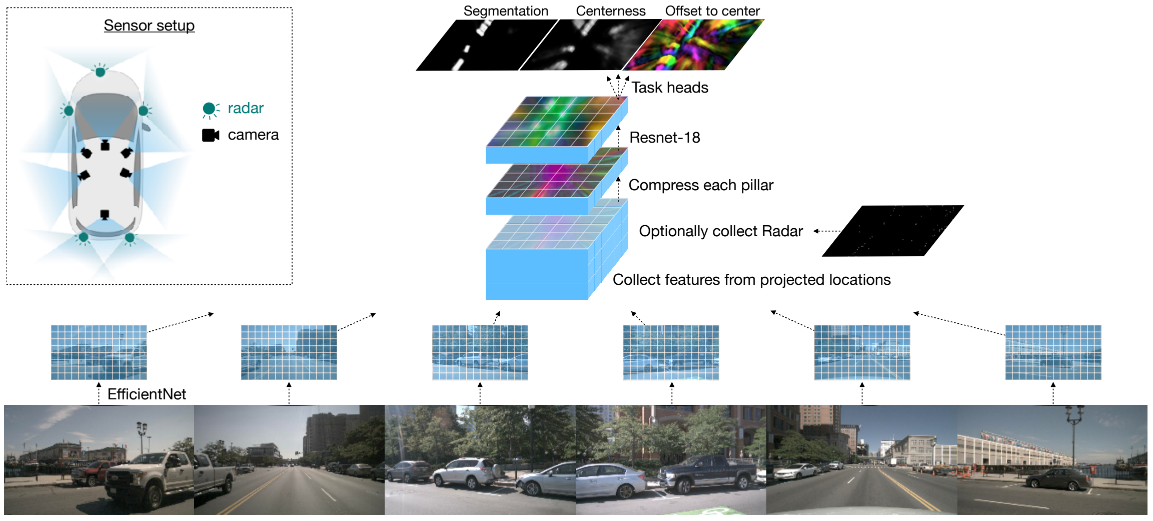

Figure 6.1: **A baseline for neural 3D mapping**. We begin by featurizing each input view, with an EfficientNet. Using a set of 3D coordinates around the ego-vehicle, we bilinearly sample features from all the views at the projected locations, yielding a 3D volume of features. We optionally add radar information, then reduce the vertical dimension of the volume, and process it with a Resnet-18 (which is 2D). Finally, independent task heads produce the quantities of interest, such as segmentation, centerness scores, and an offset field.

transfer features using model components taken from transformer literature. Saha et al. [137] begin by defining a relationship between each vertical scan-line of an image, and the ground-plane line that it projects to, and show that transformer self-attentions can learn an effective "translation" function between the two coordinate systems. Defining this transformer at the line level helps provide inductive bias to the model, since the lifting task should be similar across lines. BEVFormer [90], which is concurrent with our work, proposes to use deformable attention operations to collect image features for a pre-defined grid of 3D coordinates. This is similar to the bilinear sampling operation in the parameter-free unprojection, but with approximately 10× more samples, learnable offsets for the sampling coordinates, and a learnable kernel on their combination.

Implicit models: Another recent trend is to define an implicit function over the metric span of interest, provide the function access to the information computed from the perspective views, and query the function during inference to obtain either an intermediate representation [21] or final estimates [136]. These have not yet been evaluated on bird's eye view semantic tasks, but they appear to be effective for driving command prediction and 3D reconstruction.

6.3 Neural 3D Mapping Model

6.3.1 Setup and overview

Our model takes as input multi-view camera data, as captured by an autonomous vehicle, and optionally radar data as well. We assume that the data is synchronized across sensors. We assume that the intrinsics and extrinsics of each sensor are known.

The model has a 3D metric span, with the larger axes being forward/backward and left/right. Following the baselines in this task, we set this span to $100m \times 100m$. We set the up/down span to $10m$. This volume is centered and oriented according to a reference camera, which is typically the front camera. We denote the left-right axis with X, the up-down axis with Y, and the forward-backward axis with Z.

The model can use as input a flexible number of cameras and radar units, and optionally a Lidar unit. In the dataset used in the experiments, we have 6 cameras, pointing front, front-left, front-right, back-left, back, and back-right, and 5 radar units, pointing front, left, right, back-left, and back-right. The sensor setup is shown in Figure 6.1-left. We use the Lidar unit only for comparison, and focus on the RGB and RGB+radar cases.

Our model computes features from each camera view, then uses the intrinsics and extrinsics to cast the features along their 3D rays, creating a 3D volume of features. If radar is provided, we rasterize its returns into a bird's eye view image, and concatenate this with the feature volume. We then compress the vertical dimension, and process the resulting featuremap with a 2D convolutional net. Finally, task heads produce quantities of interest, such as segmentation, centerness scores, and an offset map, all in a 2D bird's eye view. The segmentation map contains a categorical distribution over semantic categories. The centerness map indicates probabilities of a grid cell being the center of an object. The offset map contains a vector field, where each vector points to the nearest object center. An overview of this model is illustrated in Figure 6.1.

6.3.2 Model

We begin by processing the RGB images, shaped $3 \times H \times W$, into informative featuremaps. For this, we use an EfficentNet-B4 [160]. We upsample the output of the fourth "reduction layer" and concatenate it with the third "reduction layer" output, and apply two convolution layers with instance normalization and ReLU activations [31], arriving at feature maps with shape $C \times H/8 \times W/8$ (one eighth of the image resolution).

We define a voxel resolution for our 3D span, yielding a set of discrete 3D coordinates. We project all coordinates into all featuremaps, and bilinearly sample features there, yielding a 3D feature volume from each camera. We compute a binary "valid" volume per camera at the same time, indicating if the 3D coordinate landed within the camera frustum. We then take a valid-weighted average across the set of volumes, reducing our representation down to a single 3D volume of features, shaped $C \times Z \times Y \times X$.

We next incorporate radar information, if available, and reduce the vertical dimension of the volume, yielding a bird's eye view plane of features. When radar is not available, we simply take a sum across the vertical dimension, which achieves a change in tensor dimensions $C \times Z \times Y \times X \rightarrow C \times Z \times X$. When radar is available, we first rasterize it to a bird's eye view plane, creating a binary image that matches the resolution of the 3D volume. We then rearrange the feature volume dimensions, so that the vertical dimension extends the channel dimension, as in $C \times Z \times Y \times X \rightarrow (C \cdot Y) \times Z \times X$. We then concatenate the radar, and apply a 3×3 convolution kernel to compress the extended channels back down to a dimensionality of C, which achieves $(C \cdot Y + 1) \times Z \times X \rightarrow C \times Z \times X$. If Lidar is provided, we use it the same way as radar.

At this point, we have a single plane of features, representing a bird's eye view of the scene. We process this with three blocks of a Resnet-18 [57], then use additive skip connections with bilinear upsampling to gradually bring the deeper features to the input resolution, and finally apply task-specific heads. Each head is two convolution layers with instance normalization, and ReLU after the norm in the first layer.

We use three task heads: segmentation, centerness, and offset. The segmentation head produces per-pixel vehicle/background segmentation. The centerness head produces a heatmap where high values indicate high probability that the grid cell is an object center. The offset head produces a vector field where, within each object mask, each vector points to the center of that object.

We train the segmentation head with a cross entropy loss, and supervise the centerness and offset fields with an L1 loss. We compute the ground truth from 3D box annotations. We use an uncertainty-based learnable weighting [77] to balance the three losses.

Compared to related models, the architecture choices for the perspective-view and bird's-eye-view encoders are most similar to those in Lift-Splat [119] and FIERY [64]. Our 2D-to-3D lifting step is the same as Neural 3D Mapping [54, 55], and is similar to Lift-Splat but *without any depth estimation*. Our multi-task setup is the same as FIERY [64]. In some experiments, we will compare against methods that integrate information across time, and we note that these make use of egomotion to put features from multiple timesteps into alignment on the bird's eye view map, while our model performs inference on each timestep independently and therefore does not require egomotion input.

6.3.3 Implementation details

We vary the input resolution across experiments, but our main model uses an RGB input resolution of 448×960. After the EfficientNet, the resulting feature maps are 56×120. We use a feature dimension (i.e., channel dimension) of 128.

Our 3D resolution is $200 \times 8 \times 200$, and our final output resolution is 200×200. Our 3D metric span is $100m \times 10m \times 100m$. This corresponds to voxel lengths of $0.5m \times 1.25m \times 0.5m$ (in Z, Y, X order).

The EfficientNet is pre-trained for image classification on ImageNet [30]. The Resnet-18 is trained from scratch.

At training time, we randomly sample 5 out of the 6 cameras. We also randomly select a camera to be the "reference" camera, which randomizes the orientation of the 3D volume (as well as the rasterized annotations). We apply random cropping on the RGB input, by resizing the original images to 558×992, and taking a random 448×960 crop inside (and updating the intrinsics accordingly). At test time, we use all 6 cameras, we use the "front" camera as the reference camera, and we take a center crop.

We train the model end-to-end for 300,000 iterations, with a batch size of 8, with the Adam-W optimizer [97] at a constant learning rate of 3e-4. We distribute the batch across four V100 GPUs.

The model runs at 5hz. The main bottleneck to speed is the EfficientNet, due to the high RGB resolution.

6.4 Experiments

We train and test our model in the NuScenes [11] urban scenes dataset. We use annotations from the "vehicle" superclass, which includes bicycle, bus, car, construction vehicle, emergency vehicle, motorcycle, trailer, and truck. We evaluate on vehicle/background segmentation, using the intersection-over-union (IOU) metric in a bird's eye view. (We treat centerness and offset as auxiliary tasks, and do not evaluate them on the test set.)

6.4.1 How much does each modality contribute?

We first experiment with different modality combinations, to see how much each can contribute to performance. In these experiments, our RGB resolution is 224×480.

We summarize the results in Table 6.1. We also include results from FISHING [59], which is a multi-modal model similar to ours, except that the 2D-to-3D lifting step is managed by an MLP rather than being parameter-free, and each modality is trained separately, rather than fused into a single model trained end-to-end.

Method	Input	IOU
Ours	radar	19.1
Ours	RGB	34.5
Ours	Lidar	44.5
Ours	RGB+radar	42.1
Ours	RGB+Lidar	**51.7**
FISHING [59]	RGB	30.0
FISHING [59]	Lidar	44.3
FISHING [59]	RGB+Lidar	40.9

Table 6.1: **Segmentation IOU** in the NuScenes validation set, over different modality combinations, compared to another multi-modal model. Note that in this evaluation, the RGB resolution for our model is 224×480, and the resolution for FISHING is 192×320.

Within single modalities, radar-alone does badly, RGB-alone does better, and Lidar-alone does best. Interestingly, RGB+radar improves by 8 points over RGB-alone, which suggests that the radar contains information that is difficult to extract from the RGB. We believe this is, precisely, metric information about the scene layout. The RGB+Lidar model performs the best overall, which is consistent with related work [201].

Compared to FISHING, our model achieves similar accuracy for the single-modality models, but obtains substantially better results for the multi-modality models, likely due to our end-to-end training, which better fuses the information from the sensors. While the FISHING model includes a radar branch, the authors did not report its NuScenes results in the paper, claiming that the data is "too sparse" [59]; we believe this sparsity is compensated for by end-to-end training.

6.4.2 How much does input resolution contribute?

Methods in the literature vary in terms of their input resolution. We believe this is an important factor for performance, and so we evaluate our RGB-only model across different input resolutions, and note the resolutions of the baselines. In methods cases where the paper does not list the resolution, we have retrieved the information from the public code (if available). Table 6.2 summarizes the results.

Our model obtains a sizeable 5 point boost from doubling the input resolution. In the related work, we see that in general, higher resolution is accompanied by better performance, and chronologically there has been an increase in resolution. BEVFormer[90] is concurrent with our work, and uses the full input resolution without

Method	RGB resolution	IOU
Ours	224 × 480	34.5
Ours	448 × 960	39.1
View Parsing Network [114]	224 × 400	23.9
Pyramid Occupancy Networks [135]	600 × 800	31.4
FISHING [59]	192 × 320	30.0
Lift, Splat [119]	224 × 480	32.1
FIERY [64]	448 × 960	35.8
Translating Images into Maps [137]	-	38.9
BEVFormer [90]	900 × 1600	**43.2**

Table 6.2: **Segmentation IOU** in the NuScenes validation set, over different input resolutions, for models using RGB alone. Related works are sorted in chronological order. *The IOU for View Parsing Network was retrieved from Roddick et al. [135], who re-implemented it to compute this metric.

any downsampling, and achieves the best results.

6.4.3 How much do augmentations contribute?

Prior work has used random cropping in the RGB input, and random camera dropout. As far as we are aware, prior work has not randomized the camera selected to be the "reference" camera, which dictates the orientation of the 3D coordinate system. This yields a boost of 1.7 points in performance, as shown Table 6.3. We believe that randomizing this factor helps reduce overfitting in the bird's eye view module. We have observed anecdotally that without this augmentation, the segmented cars have a slight bias for certain orientations in certain positions; with the augmentation added, this bias disappears.

We have also experimented with color, contrast, and blur augmentations, and found that these worsened results. This is surprising, because such augmentations are well-known to be beneficial in image classification. It may be that the model benefits from sensitivity to these factors in the current data.

6.4.4 How high can we push performance, with everything together?

Using high-resolution input, all augmentations, and fusing radar with RGB, we obtain a new best result of **49.5**. We compare this against the state of the art, including also

Method	**Augmentations**	**IOU**
Ours	Random crop, random 5/6 cameras	40.0
Ours	Random crop, random 5/6 cameras, random refcam	41.7

Table 6.3: **Segmentation IOU** in the NuScenes validation set, over different augmentation settings, for models using RGB alone, with resolution 448×960.

Method	**Inputs**	**IOU**
FIERY [64]	RGB	35.8
FIERY [64]	RGB+Time	38.2
Translating Images into Maps [137]	RGB	38.9
Translating Images into Maps [137]	RGB+Time	41.3
BEVFormer [90]	RGB	43.2
BEVFormer [90]	RGB+Time	46.7
Ours	RGB	41.7
Ours	RGB+radar	**49.5**

Table 6.4: **Segmentation IOU** in the NuScenes validation set, for recent high-performing models.

methods that integrate information across time. Our model exceeds the rest, including the concurrent BEVFormer [90] when it uses temporal information, although that model does better than our RGB-alone model. Integrating over time is orthogonal to the strengths of our model, so it should be possible to push results higher with time. Note also that integrating over time is accomplished in these works using egomotion information (which we do not use), to put features from multiple timesteps into alignment on the bird's eye view map.

We visualize a random sample of results in Figure 6.2. We also visualize corresponding Lidar and radar returns, to show the scene structure as captured by those sensors. Qualitatively, the radar is indeed sparse as noted in related work, but we believe it gives valuable hints about the metric scene structure, which, when fused with information acquired from RGB, enables higher-accuracy semantic segmentation in the bird's eye view.

6.5 Conclusion

Bird's eye view estimation of driving-relevant information is a rapidly-advancing area. Recent works have developed sophisticated methods for transferring data from the 2D image views onto the 2D ground plane, typically using cameras alone or cameras in combination with Lidar. In this paper, we explore a simple baseline model, with a non-parametric 2D-to-3D step, and surprisingly show that it performs competitively with the state-of-the-art, and outperforms most methods when certain augmentations are added. We then reconsider the assumption that radar data is too sparse to be useful, and propose a simple strategy to fuse its noisy returns with the 3D representation acquired from RGB, and exceed the performance of all published models, including ones that integrate information across time. We hope that this simple model will serve as a useful baseline in the future.

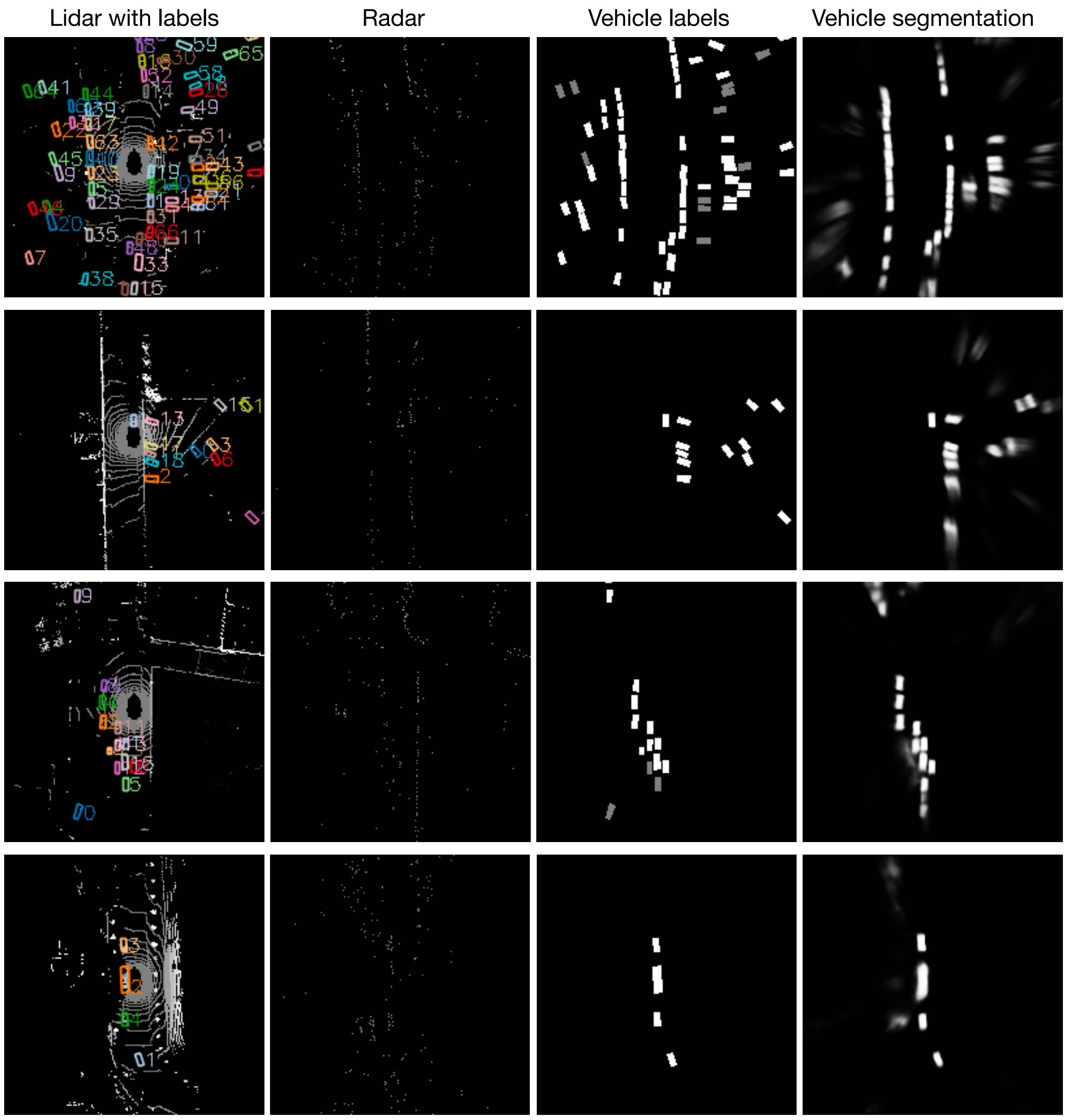

Figure 6.2: **Visualization of Lidar vs radar, and results.** From left to right: voxelized lidar in a bird's eye view, with ground truth boxes overlaid; voxelized radar in a bird's eye view, vehicle labels, and vehicle segmentation estimates.

Part II

Geometry-free amodal scene representations

Chapter 7

Particle Videos Revisited: Tracking Through Occlusions Using Point Trajectories

7.1 Introduction

In 2006, Sand and Teller [141] wrote that there are two dominant approaches to motion estimation in video: feature matching and optical flow. This is still true today. In their paper, they proposed a new motion representation called a "particle video", which they presented as a middle-ground between feature tracking and optical flow. The main idea is to represent a video with a set of particles that move across multiple frames, and leverage long-range temporal priors while tracking the particles.

Methods for feature tracking and optical flow estimation have greatly advanced since that time, but there has been relatively little work on estimating long-range trajectories at the pixel level. Feature correspondence methods [13, 181] currently work by matching the features of each new frame to the features of one or more source frames [83], without taking into account temporal context. Optical flow methods today produce such exceedingly-accurate estimates within pairs of frames [162] that the motion vectors can often be chained across time without much accumulation of error, but as soon as the target is occluded, it is no longer represented in the flow field, and tracking fails.

Particle videos have the potential to capture two key elements missing from feature-matching and optical flow: (1) persistence through occlusions, and (2) multi-frame temporal context. If we attend to a pixel that corresponds to a point on the world surface, we should expect that point to exist across time, even if appearance and position and visibility all vary somewhat unpredictably. Temporal context is of course widely known to be relevant for flow-based methods, but prior efforts to

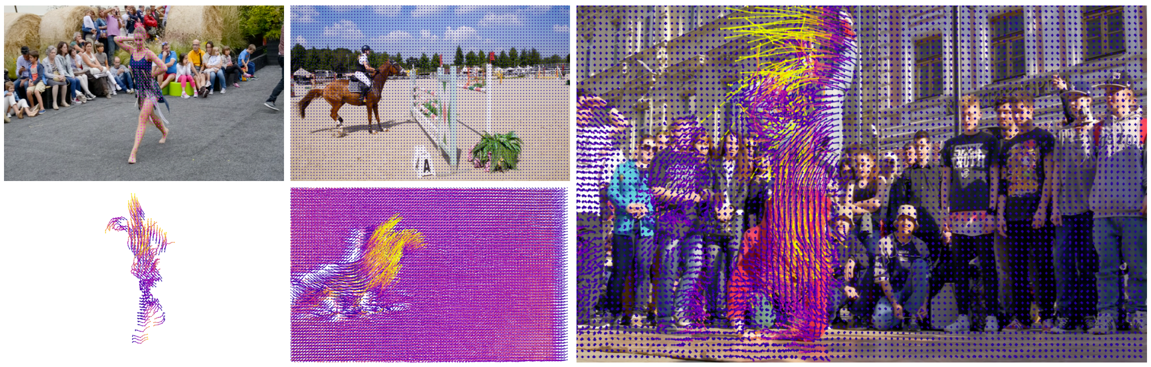

Figure 7.1: **Deep particle video.** Our method takes an RGB video as input, and estimates trajectories for any number of target pixels. Top-left: target pixels are marked with dots; bottom-left: estimated trajectories. Right: estimated trajectories overlaid on the first frame of the input video.

take multi-frame context into account have yielded only small gains. Flow-based methods mainly use consecutive pairs of frames, and occasionally leverage time with a simple constant-velocity prior, which weakly conditions the current flow estimate on previous frames' flow [132, 162].

We propose Deep Particle Video (DPV), a new particle video method, which takes a T-frame RGB video as input, along with the (x, y) coordinate of a target to track, and produces a $T \times 2$ matrix as output, representing the positions of the target across the given frames. The model can be queried for any number of particles, at any positions within the first frame's pixel grid. A defining feature of our approach, which differentiates it from both the original particle video method and modern flow methods, is that it makes an extreme trade-off between spatial awareness and temporal awareness. *Our model estimates the trajectory of every target independently.* Computation is shared between particles within a video, which makes inference fast, but each particle produces its own trajectory, without inspecting the trajectories of its neighbors. This extreme choice allows us to devote the majority of parameters into a module that simultaneously learns (1) temporal priors, and (2) an inference mechanism that *searches* for the target pixel's location in all input frames. The value of the temporal prior is that it allows the model to *fail* its correspondence task at multiple intermediate frames. As long as the pixel is "found" at some sparse timesteps within the considered temporal span, the model can use its prior to estimate plausible positions for the remaining timesteps. This is helpful because appearance-based correspondence is impossible in some frames, due to occlusions, moving out-of-bounds, or difficult lighting.

We train our model entirely in synthetic data, which we call FlyingThings++, based on the FlyingThings [100] optical flow dataset. Our dataset includes multi-frame amodal trajectories of various lengths, with challenging synthetic occlusions caused by moving and static objects. In our experiments on both synthetic and real video data, we demonstrate that our particle trajectories are more robust to occlusions than flow trajectories—they can pick up an entity upon re-appearance—and also provide smoother and finer-grained correspondences than current feature-matching methods, thanks to its temporal prior. We also propose a method to link the model's moderate-length trajectories into arbitrarily-long trajectories, relying on a simultaneously-estimated visibility cue. Figure 7.1 displays sample outputs of our model on RGB videos from the DAVIS benchmark [123]. We plan to publicly release our code, model weights, and data.

7.2 Related Work

7.2.1 Optical flow

While earlier optical flow methods use optimization techniques to estimate motion fields between two consecutive frames [10, 158], recent methods learn such displacement fields supervised from synthetic datasets [33, 65]. Many recent works use iterative refinements for flow estimation by leveraging coarse-to-fine pyramids [156]. Instead of coarse-to-fine refinements, RAFT [162] mimics an iterative optimization algorithm, and estimates flow through iterative updates of a high resolution flow field based on 4D correlation volumes constructed for all pairs of pixels from per-pixel features. Inspired by RAFT, we also perform iterative updates of the position estimations using correlations as an input, but unlike RAFT we additionally update features.

Ren *et al.* [132] propose a fusion approach for multi-frame optical flow estimation. The optical flow estimates of previous frames are used to obtain multiple candidate flow estimations for the current timestep, which are then fused into a final prediction by a learnable module. In contrast, DPV explicitly reasons about multiframe context, and iteratively updates its estimates across all frames considered. Note that without using multiple frames, it is impossible to recover an entity after occlusion. Janai et al. [69] is closer to our method, since it uses 3 frames as multiframe context, and explicitly reasons about occlusions. That work uses a constant velocity prior [140] to estimate motion during occlusion. In contrast, DPV devotes a large part of the model capacity to learning an accurate temporal prior, and iteratively updates its estimates across all frames considered, in search of the object's re-emergence from occlusion. Note that without using multiple frames, it is impossible to recover an entity after occlusion. Additionally, our model is the only work that aims to recover *amodal* trajectories that do not terminate at occlusions but rather can recover and re-connect with a visual entity upon its re-appearance.

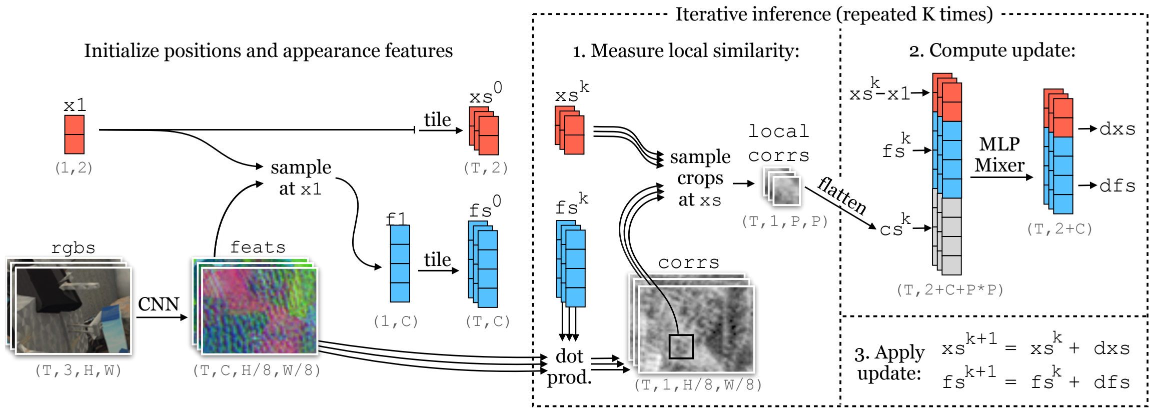

Figure 7.2: **Architecture of Deep Particle Video (DPV).** Given an RGB video as input, along with a location in the first frame indicating what to track, our model initializes a multi-frame trajectory, then computes features and correlation maps, and iteratively updates the trajectory and its corresponding sequence of features, with a deep MLP-Mixer model.

7.2.2 Feature matching

Wang and Jabri et al. [66, 181] leverage cycle consistency of time for feature matching. This allows unsupervised learning of features by optimizing a cycle consistency loss on the feature space across multiple time steps in unlabelled videos. Lai et al. [82, 83] and Yang et al. [196] learn feature correspondence through optimizing a proxy reconstruction objective, where the goal is to reconstruct a target frame (color or flow) by linearly combining pixels from one or more reference frames. The combination weights are obtained by computing affinities between the features from the target frame and features from the reference frame(s).

Instead of using proxy tasks, supervised approaches [45, 71, 179, 186] directly train models using ground truth correspondences across images. Features are usually extracted per-image and a transformer-based processor locates correspondences between images. In this work, we reason about point correspondences over a long temporal horizon, incorporating motion context, instead of using pairs of frames like these works.

7.3 Deep Particle Video (DPV)

7.3.1 Setup and overview

We name our model Deep Particle Video (DPV), taking conceptual inspiration from Particle Video [141]. DPV takes as input an RGB video with T frames, and the (x_1, y_1)

coordinate of a single point on the first frame, indicating the target to track. As output, the model produces a $T \times 3$ matrix, containing T tracked coordinates (x_t, y_t) across time, along with visibility/occlusion estimates $v_t \in [0, 1]$. The model can be queried for N target points in parallel, and some computation will be shared between them, but the model does not share information between the targets' trajectories.

At training time, we query the model with points for which we have ground-truth trajectories and visibility labels. We supervise the model's (x_t, y_t) outputs with a regression objective, and supervise v_t with a classification objective. At test time, the model can be queried for the trajectories of any number of points.

We use the words "point" and "particle" interchangeably to mean the things we are tracking, and use the word "pixel" more broadly to indicate any discrete cell on the image grid. Note that although the tracking targets are specified with single pixel coordinates, tracking successfully requires (at least) taking into account the local spatial context around the specified pixel, and therefore the model is somewhat sensitive to scale and resolution.

Our overall approach has four stages, somewhat similar to the RAFT optical flow method [162]: extracting visual features (Section 7.3.2), initializing a list of positions and features for each target (Section 7.3.3), locally measuring appearance similarity (Section 7.3.4), and repeatedly updating the positions and features for each target (Section 7.3.5). Figure 7.2 shows an overview of the method.

7.3.2 Extracting features

We begin by extracting features from every frame of the input video. In this step, each frame is processed independently with a 2D convolutional network (i.e., no temporal convolutions). The network produces features at 1/8 resolution.

7.3.3 Initializing each target

After computing feature maps for the video frames, we compute a feature vector for the target, by bilinearly sampling inside the first feature map at the first (given) coordinate, obtaining a feature vector f_1.

We use this sampled feature to initialize a trajectory of features, by simply tiling the feature across time, yielding a matrix $\mathcal{F}^0$ sized $T \times C$, where C is the channel dimension. This initialization implies an appearance constancy prior.

We initialize the target's trajectory of positions in a similar way. We simply copy the initial position across time, yielding a matrix $\mathcal{X}^0$, shaped $T \times 2$. This initialization implies a zero-velocity prior, which essentially assumes nothing about the target's motion.

During inference, we will update the trajectory of features, tracking appearance changes, and update the trajectory of positions, tracking motion.

7.3.4 Measuring local appearance similarity

We would like to measure how well our trajectory of positions, and associated trajectory of features, matches with the pre-computed feature maps. We compute visual similarity maps by correlating each feature f^t with the feature map of the corresponding timestep, and then obtain "local" scores by bilinearly sampling a crop centered at the corresponding position (x_t, y_t). This step returns patches of un-normalized similarity scores, where large positive values indicate high similarity between the target's feature and the convolutional features at this location. We denote the initial set of scores as $\mathcal{C}^0$, shaped $T \times P \cdot P$, where P is the size of the patch extracted from each correlation map.

Similar to RAFT [162], we find it is beneficial to create a spatial pyramid of these score patches, to obtain similarity measurements at multiple scales. This makes our score matrix $T \times P \cdot P \cdot L$, where L is the number of levels in the pyramid.

7.3.5 Iterative updates

The main inference step for our model involves updating the sequence of positions, and updating the sequence of features. To perform this update, we take into account all of the information we have computed thus far: the position matrix $\mathcal{X}^k$, the feature matrix $\mathcal{F}^k$, and the correlation matrix $\mathcal{C}^k$, indicated here with superscript k to emphasize that these inputs will change across iterative inference steps.

Instead of using absolute positions $\mathcal{X}^k$, we subtract the given position (x_1, y_1) from each element of this matrix, making it into a matrix of displacements. (On the first iteration, it will in fact be all zeros, since $\mathcal{X}^0$ is initialized from (x_1, y_1).) Using displacements instead of absolute positions makes all input trajectories appear to start at $(0, 0)$, which makes our model translation-invariant. We additionally concatenate sinusoidal position encodings of the displacements [174], motivated by the success of these encodings in vision transformers [35].

To process this broad set of inputs, we concatenate them all on the channel dimension, yielding a new matrix shaped $T \times D$, and process this with a 12-block MLP-Mixer [166], which is a parameter-efficient all-MLP architecture with design similarities to a transformer. As output, this module produces updates for the sequence of positions and sequence of features, $d\mathcal{X}$ and $d\mathcal{F}$, which we apply with:

$$\begin{aligned} \mathcal{F}^{k+1} &= \mathcal{F}^k + d\mathcal{F}, \\ \mathcal{X}^{k+1} &= \mathcal{X}^k + d\mathcal{X}. \end{aligned} \tag{7.1}$$

After each update, we compute new correlation pyramids at the updated coordinates, using the updated features.

The update module is iterated K times. After the last update, the positions $\mathcal{X}^K$ are treated as the final trajectory, and the features $\mathcal{F}^K$ are sent to a linear layer to estimate per-timestep visibility logits $\mathcal{V}$.

7.3.6 Supervision

We supervise the model using the L_1 distance between the ground-truth trajectory and the estimated trajectory (across iterative updates), with exponentially increasing weights, similar to RAFT [162]:

$$\mathcal{L}_{\text{main}} = \sum_{k}^{K} \gamma^{K-k} ||\mathcal{X}^k - \mathcal{X}^*||_1, \tag{7.2}$$

where K is the number of iterative updates, and we set $\gamma = 0.8$. Note that this loss is applied even when the target is occluded, or out-of-bounds, which is possible since we are using synthetically-generated ground truth. This is the main loss of the model, and the model can technically train using only this, although it will not learn visibility estimation and convergence will be slow.

On the model's visibility estimates, we apply a cross entropy loss:

$$\mathcal{L}_{\text{ce}} = \mathcal{V}^* \log \mathcal{V} + (1 - \mathcal{V}^*) \log(1 - \mathcal{V}). \tag{7.3}$$

We find it accelerates convergence to directly supervise the score maps to peak in the correct location (i.e., the location of the true correspondence) when the target is visible:

$$\mathcal{L}_{\text{score}} = -\log(\exp(c_i) / \sum_j \exp(c_j)) 1\{\mathcal{V}^* \neq 0\}, \tag{7.4}$$

where c_j represents the match score at pixel j, and i is pixel index with the true correspondence.

We average all losses across all targets within a batch. We set the coefficient of every loss to 1.

7.3.7 Test-time trajectory linking

At test time, it is often desirable to generate correspondences over longer timespans than the training sequence length T. To generate these longer trajectories, we may repeat inference starting from any timestep along the estimated trajectory, treating (x_t, y_t) as the new (x_1, y_1), and thereby "continuing" the trajectory up to (x_{t+T}, y_{t+T}). However, doing this naively (e.g., always continuing from the last timestep), can quickly cause tracking to drift.

It is first of all crucial to avoid continuing the trajectory from a timestep where the target is occluded. Otherwise, the model will switch to tracking the occluder. To avoid these identity switches, we make use of our visibility estimates, and seek the farthest timestep whose visibility score is above a threshold. Note that seeking farthest visible timestep allows the model to skip past frames where the target was momentarily occluded, as long as the temporal span of the occlusion is less than the temporal span of the model (T). Visibility estimates always begin near 1, since the model is trained assuming the provided (x_1, y_1) indicates the true target, so we exclude the first several timesteps from this selection process, forcing the model to choose a timestep from the later part of the trajectory. We initialize the threshold conservatively at 0.99, and decrease it in increments of 0.01 until a valid selection is found.

Even with visibility-aware trajectory linking, the model can "forget" what it was originally tracking, since the target initialization strategy involves bilinearly sampling at the new (x_1, y_1) location whenever a new link is added to the chain. We therefore employ a second strategy, which is simply to always initialize $\mathcal{F}^0$ using the features found at the very first timestep. Intuitively, this locks the model into tracking the "original" target. This does create a slight mismatch between the target features and the convolutional feature maps on the current frame, but we find that this is effectively compensated for by the internal update mechanism. We have also tried a strategy of initializing with the last $\mathcal{F}^K$ output by the model, but this leads to unstable behavior.

7.4 Implementation details

CNN: Our CNN uses the "BasicEncoder" architecture from the official RAFT codebase [163]. This architecture has a 7×7 convolution with stride 2, then 6 residual blocks with kernel size 3×3, then a final convolution with kernel size 1×1. The CNN has an output dimension of $C = 256$.

Local correlation pyramids: We use three levels in our correlation pyramids, with radius 4. This translates to three 9×9 correlation patches per timestep.

MLP-Mixer: The input to the MLP-Mixer is a sequence of relative coordinates, features, and correlation pyramids. The coordinates are made "relative" by subtracting the first position, meaning the first element of this sequence is $(0, 0)$, and other elements of the sequence can be interpreted as flow vectors originating from the first timestep's absolute position. The per-timestep inputs are flattened, then treated as a sequence of vectors (i.e., "tokens") for the MLP-Mixer. We use the MLP-Mixer architecture exactly as described in the original paper; at the end of the model there is a mean over the sequence dimension, followed by a linear layer that maps to a channel size of $T \cdot (C + 2)$.

Updates: We reshape the MLP-Mixer's outputs into a sequence of feature updates and a sequence of coordinate updates. We apply the coordinate updates directly (with addition). We use separate linear layers to apply the updates for the instance-level and pixel-level features (i.e., ϕ in Equation 1 from the main paper). We train and test with 8 updates, but we find that performance is similar if we train with 4 (and still test with 8).

Visibility: We use a linear layer to map the last update iteration's pixel-level feature sequence into visibility logits.

Optimization: We train with a batch size of 4, distributed across four GPUs. At training time, we use a resolution of 368×496. For each element of the batch, we randomly sample 32 trajectories whose initial timestep is marked "visible" in the ground truth (so that the tracking targets always begin within-bounds and un-occluded). We train for 500,000 steps, with a learning rate of 1e-4 with a 1-cycle schedule [150], using the AdamW optimizer and clipping gradients to $[-1, 1]$. Training takes approximately 5 days on four GeForce RTX 2080s.

Hyperparameters: We use $T = 8$ (timesteps considered by the MLP-Mixer), and $K = 8$ (update iterations). The model can in general be trained for any T, but we found that the model was more difficult to train at $T = 16$ and $T = 32$, likely because the complexity of trajectories grows rapidly with their length under our model, as there is no weight sharing across time. On the other hand, the temporal sensitivity allows our model to learn more complex temporal priors. We found that $K > 8$ performs similar to $K = 8$, and so use $K = 8$ because it is faster.

Complexity: *Speed:* When the number of targets is small enough to fit on a GPU (e.g., 128 targets for a 12G GPU), our model is faster than RAFT (340ms vs. 2000ms at 480×1024). RAFT is comparatively slow because (1) it is too memory-heavy to compute all frames' flows in parallel, so we must run it $T - 1$ times, and (2) much computation is spent on non-target pixels. *Memory:* Our model's memory scales with $T \cdot N$, where N is the number of particles being tracked, due to the iterated MLP-Mixer which consumes a T-length sequence of features per particle.

Code: DPV is implemented in Pytorch [116]. We will publicly release the code for the model and training procedure.

7.5 Experiments

We train our model in a modified version of FlyingThings [100], which we name FlyingThings++ (discussed more below).

We evaluate our model on tracking objects in FlyingThings++, tracking vehicles and pedestrians in KITTI [43], tracking heads in a crowd in CroHD [159], and finally, propagating keypoints in animal videos in BADJA [5]. We visualize trajectory estimates in DAVIS videos in Figure 7.1, to illustrate the method's generality, and visualize the estimates against ground truth in Figures 7.4 and 7.3.

7.5.1 Training data: FlyingThings++

We created a synthetic dataset based on FlyingThings [100], which is a dataset typically used to train optical flow models (usually in combination with other datasets). We chose FlyingThings because (1) its visuals and motions are extremely complex, which gives hope of generalizing to other data, and (2) it provides 10-frame videos with ground-truth forward and backward optical flow (as opposed to 2-frame videos), from which we can mine multi-step trajectory ground truth.

To create ground truth multi-frame trajectories, we chain the ground-truth flows forwards, and discard chains that fail a forward-backward consistency check (e.g., by landing on occluders). Through this process we create a sparse set of 4-frame and 8-frame trajectories. To this set we also add 2-frame trajectories from the raw flow fields.

The forward-backward consistency check ensures that the trajectories are accurate, but it leaves us with a library of trajectories where the target is visible on every timestep. Therefore, it is necessary to add *new occlusions* on top of the video. We do this on-the-fly during batching: for each FlyingThings video in the batch, we randomly sample an object from an alternate FlyingThings video, paste it directly on top of the current video, overwriting the pixels within its mask on each frame. We then update the ground-truth to reflect the new visibility in the occluded area on each frame, as well as update the trajectory list to include the trajectories of the added object.

Combining all videos with at least 32 valid 8-frame trajectories, we obtain a total of 4311 training videos, and 734 test videos. To expand the breadth of the training set, we augment the data on-the-fly with color and brightness changes, random scale changes, random crops which randomly shift across time, random Gaussian blur, and random horizontal and vertical flips.

7.5.2 Baselines

In our experiments we consider the following baselines.

Recurrent All-Pairs Field Transforms (RAFT) [162] represents the state-of-the-art in optical flow estimation, where a high resolution flow field is refined through iterative updates, based on lookups from a 4D cost volume constructed between all pairs of pixels. Note that similar to our method, RAFT has been trained on FlyingThings (including occlusions and out-of-bounds motions), but only has a 2-frame temporal span. To generate multi-frame trajectories with RAFT at test time, we compute flow with all consecutive pairs of frames, and then compute flow chains at the pixels queried on the first frame. To continue chains that travel out of bounds, we clamp the coordinates to the image bounds and sample at the edge of the flow map.

DINO [13] is a vision transformer (ViT-S [35] with patch size 8) trained on ImageNet with a self-supervision objective based on a knowledge distillation setup that builds invariance to image augmentations. To use this model for multi-frame correspondence, we use the original work's code for instance tracking, which uses nearest neighbor between the initial frame and the current frame, as well as nearest-neighbor between consecutive frames, and a strategy to restrict matches to a local neighborhood around previous matches. We report results with and without this "windowing" strategy.

TimeCycle [181] learns correspondences between pixels from different frames by optimizing an objective that encourages correspondences to be cycle-consistent across time (i.e., forward-backward consistency), including across frame skips. This method tracks the same way as DINO, by computing feature affinity across frames and reporting nearest neighbors.

Contrastive Random Walk (CRW) [66] treats the video as a space-time graph, with edges containing transition probabilities of a random walk, and computes long-range correspondences by walking across the graph. Similar to TimeCycle [181], the model is trained with cycle-consistency.

Memory-Augmented Self-supervised Tracker (MAST) [83] learns correspondences between features by reconstructing the target frame with linear combinations of reference frames. At test time the correspondences are predicted autoregressively. The model is trained on OxUvA [172] and YouTube-VOS [194]

Video Frame-level Similarity (VFS) [193] learns an encoder that produces frame-level embeddings which are similar within a video, and dissimilar across videos. This model is trained on Kinetics-400 [76].

ImageNet ResNet [57] is a ResNet50 supervised for classification with ImageNet labels, and evaluated the same way as DINO.

7.5.3 Trajectory estimation in FlyingThings++

Using 8-frame videos from the FlyingThings++ test set as input, we estimate trajectories for all pixels for which we have ground-truth, and evaluate the Euclidean distance between each estimated trajectory and its corresponding ground truth, averaging over all 8 timesteps. We are especially interested in measuring how our model's performance compares with the baselines when the target gets occluded or flies out of frame. To create controlled out-of-bounds and occlusion tests, we first take a central crop of each video, sized 368×496, providing us with some trajectories that fly out of the crop. Second, we implement a simple controllable occlusion strategy, where we replace a 200×200 square in frames 2-5 with gray pixels.

We compare our model against DINO [13], representing the state-of-the-art for feature matching, and RAFT [162], representing the state-of-the-art for flow. Table 7.1 shows the results across the different evaluations on the test set. DINO struggles

Method	Vis.	OOB	Occ
DINO [13]	32.4	75.4	68.13
RAFT [162]	9.05	19.41	36.30
DPV (ours)	**7.19**	**18.48**	**18.41**

Table 7.1: **Trajectory estimation error in FlyingThings++.** DPV is more robust to targets moving out-of-bounds or becoming occluded.

Method	Veh.	Ped.
DINO [13]	12.5	11.5
RAFT [162]	5.6	5.4
DPV (ours)	**5.0**	**5.3**

Table 7.2: **Trajectory estimation error in KITTI.** DPV performs slightly better than RAFT; DINO lags behind both.

across all splits and performs worse on occluded pixels than on visible ones. RAFT obtains high accuracy for visible pixels, but its errors increase as the occlusions become more difficult. Our model performs similarly to RAFT on visible pixels, but it is somewhat robust to occlusions. Inspecting the results manually, we see that RAFT's trajectories become "stuck" in the region of the added occluder, which makes sense because flows there do not reflect the motion of the targets. Our model, in contrast, is able to locate the targets after they re-emerge from the occluder, and inpaint the missing portions of the trajectories.

7.5.4 Trajectory estimation in KITTI

We additionally evaluate on an 8-frame point trajectory dataset that we created from the "tracking" subset of the KITTI [43] urban scenes benchmark. To create 8-frame trajectories, we sample a 3D box annotation that has at least 8 valid timesteps, select a LiDAR point within the box on the first timestep, transform it in 3D to its corresponding location on every other step, and project this location into pixel coordinates.

In Table 7.2 we see that RAFT and our method perform approximately on par with one another (with our method winning by less than one point), but DINO's error is nearly twice this. We evaluate separately on vehicles and pedestrians since occlusions are more common with pedestrians, but we note that pedestrians also move more slowly, so the average errors are similar. Qualitative results for our model are shown in Figure 7.3-middle.

Method	Vis.	Occ.
DINO [13]	21.84	43.34
RAFT [162]	12.24	21.16
DPV (ours)	**8.26**	**15.93**

Table 7.3: **Trajectory estimation error in CroHD.** DPV achieves better accuracy, for both visible and occluded targets.

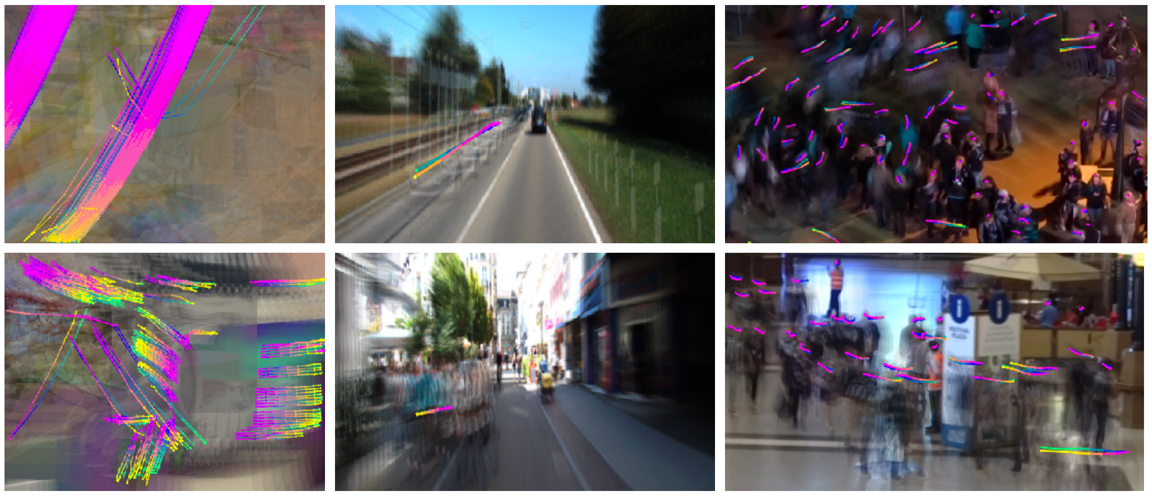

Figure 7.3: **Qualitative results in FlyingThings++ (left), KITTI (middle), and CroHD (right).** We visualize a video with the mean of its RGB. We trace the estimates with pink-to-yellow trajectories, and show ground truth in blue-to-green. FlyingThings++ is chaotic, but training on this data allows our model to generalize.

7.5.5 Trajectory estimation in CroHD

We additionally evaluate on the Crowd of Heads Dataset (CroHD) [159], which consists of high-resolution (1920 x 1080) videos of crowds, with annotations tracking the heads of people in the crowd. We evaluate on 8-frame sequences extracted from the dataset, using an FPS of 12.5. We filter out targets whose motion is below a threshold distance, and split the evaluation between targets that are visible and those that undergo occlusions. The results are shown in Table 7.3. In this data, DPV outperforms RAFT and DINO by a wide margin, both visibility settings. DINO struggles overall, likely because the motions in this dataset are small, and DINO is only able to track at a coarse resolution. Qualitative results for our model are shown in Figure 7.3-right.

Method	bear	camel	cows	dog-a	dog	horse-h	horse-l	Avg.
Win. DINO [13]	**77.9**	69.8	**83.7**	<u>17.2</u>	46.0	29.1	50.8	<u>53.5</u>
Win. ResNet [57]	70.7	65.3	71.7	6.9	27.6	20.5	49.7	44.6
Win. TimeCycle [181]	13.6	10.0	8.0	3.4	9.8	7.9	13.1	9.4
Win. CRW [66]	63.2	<u>75.9</u>	77.0	6.9	32.8	20.5	22.0	42.6
Win. VFS [193]	63.9	74.6	76.2	6.9	35.1	27.2	40.3	46.3
DINO [13]	<u>75.0</u>	59.2	70.6	10.3	**47.1**	35.1	**56.0**	50.5
ImageNet ResNet [57]	65.4	53.4	52.4	0.0	23.0	19.2	27.2	34.4
TimeCycle [181]	13.6	8.4	14.4	3.4	5.7	13.2	13.6	10.3
CRW [66]	66.1	67.2	64.7	6.9	33.9	25.8	27.2	41.7
VFS [193]	64.3	62.7	71.9	10.3	35.6	33.8	33.5	44.6
MAST [83]	51.8	52.0	57.5	3.4	5.7	7.3	34.0	30.2
RAFT [162]	64.6	65.6	69.5	13.8	39.1	<u>37.1</u>	29.3	45.6
DPV (ours)	72.7	**84.9**	<u>78.6</u>	**34.2**	<u>46.6</u>	**46.7**	<u>51.4</u>	**59.3**

Table 7.4: **PCK-T in BADJA.** In this evaluation, keypoints are initialized in the first frame of the video, and are propagated to the end of the video; PCK-T measures the accuracy of this propagation. In each column, we bold the best result, and underline the second-best. Above the middle bar, we give methods a spatial window (marked "Win.") to constrain how they propagate labels, which encodes domain knowledge about the span of plausible motions in the domain (which is a common strategy in existing work). Below the bar, we run each method in the unconstrained setting. Our method wins in most videos, but DINO performs well also.

7.5.6 Keypoint propagation in BADJA

BADJA [5] is a dataset of animal videos with keypoint annotations. These videos overlap with the DAVIS dataset [123], but include keypoint annotations. Keypoint annotations exist on approximately 1/5 frames, and the standard evaluation is Percentage of Correct Keypoint-Transfer (PCK-T), where keypoints are provided on a reference image, and the goal is to propagate these annotations to other frames. A keypoint transfer is considered correct if it is within a distance of $0.2\sqrt{A}$ from the true pixel coordinate, where A is the area of the ground-truth segmentation mask on the frame.

We note that some existing methods test on a simplified version of this keypoint propagation task, where the ground-truth segmentation is available on every frame of the video (e.g., [197, 198]). Here, we focus on the harder setting, where the ground-truth mask is unknown. Similarly, we have found that feature-matching methods (e.g.,

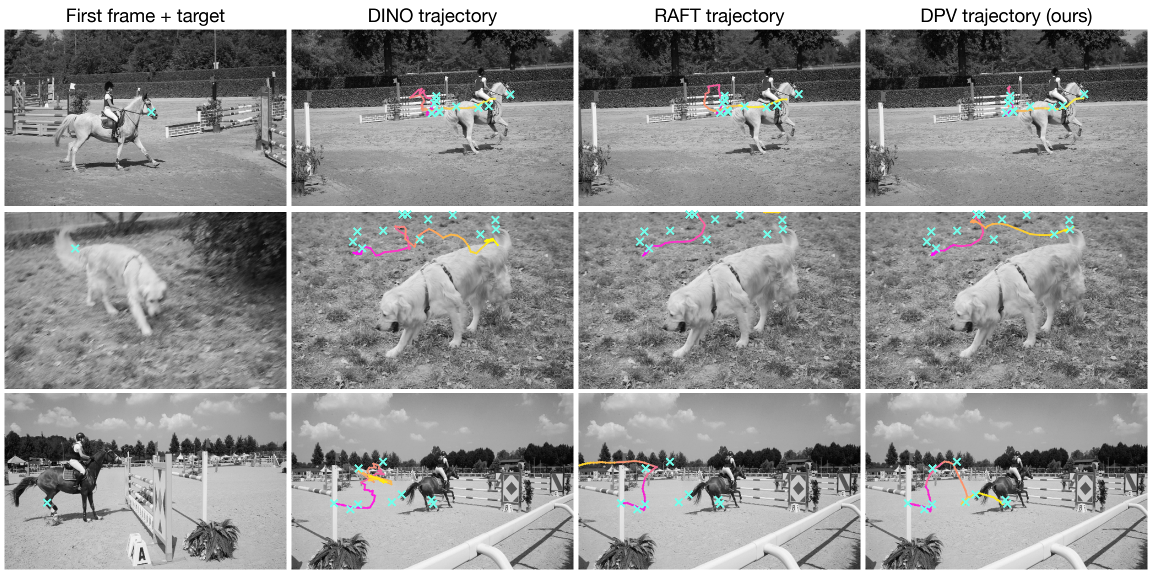

Figure 7.4: **Comparison with baselines in BADJA, on videos with occlusions.** For each method, we trace the estimated trajectory with a pink-to-yellow colormap. The sparse ground truth is visualized with cyan x's. In the video on the first row, all methods perform fairly well, though DINO and RAFT drift slightly toward the horse's body. In the second video, the target (on the dog's tail) leaves the image bounds then returns into view. In the third video, the target (on the horse's leg) is momentarily occluded, causing RAFT to lose track entirely.

[13]) constrain their correspondences to a local spatial window around the previous frame's match. We report results for these methods with the qualifier "Windowed", but focus again on the un-constrained version of the problem, where keypoints need to be propagated from frame 1 to every other frame, with no other knowledge about motion or position.

Table 7.4 shows the results of the BADJA evaluation. On four of the seven videos, our model produces the best keypoint tracking accuracy, as well as the best on average, by a margin of 9 points. DINO [13] obtains the best accuracy in the remaining videos, though its widest margin over our model is just 5 points. Interestingly, windowing helps DINO (and other baselines) in some videos but not in others, perhaps because of the types of motions in DAVIS. We note that DAVIS has an object-centric bias (i.e., the target usually stays near the center of the frame), which translation-sensitive methods like DINO can exploit, since their features encode image position embeddings; RAFT and DPV track more generally. In Figure 7.4 we visualize DINO, RAFT, and DPV trajectories on targets that undergo momentary occlusions, illustrating how DINO tracks only coarsely, and how RAFT loses track after the occlusion.

7.6 Ablation on visibility-aware linking

In this section we evaluate the effect of visibility-aware trajectory linking instead of naive linking.

Our trajectory linking strategy relies on visibility estimates produced by the model. Without using these estimates, we may still link trajectories naively (i.e., greedily), by chaining the 8-frame trajectories end-to-end. We evaluate this choice in BADJA keypoint propagation, and show the result in Table 7.5. Naive linking indeed gives worse PCK-T, but the margin is only 0.2 points. This may suggest that the visibility estimator is not generalizing well to the new video domain. Alternatively, this may be showing a small effect because the metric is dominated by trajectories that stay visible for the full duration of the video, as evidenced by DINO obtaining 50.5 PCK-T with no strategy for managing occlusions at all (see Table 4 from main paper).

Method	Average PCK-T
Without visibility-aware linking	59.1
With visibility-aware linking	**59.3**

Table 7.5: **Effect of visibility-aware linking on keypoint propagation in BADJA.** Linking trajectories from locations estimated to be "visible" yields a small improvement in average PCK-T.

7.6.1 Limitations

Our model has two main limitations. First is our unique extreme tradeoff, of spatial awareness for temporal awareness. Although this maximizes the power of the temporal prior in the model, it sacrifices potentially valuable information that could be shared between trajectories. We are indeed surprised that single-particle tracking performs as well as it does, considering that spatial smoothness is known to be essential for accurate optical flow estimation. Extending our architecture to concurrent estimation of multiple point trajectories is a direct avenue for future work.

Our second main limitation stems from the MLP-Mixer. Due to this architecture choice, our model is not recurrent across time. Although longer trajectories can be produced by re-initializing our inference at the tail of an initial trajectory, our model will lose the target if it stays occluded beyond the model's temporal window. We have tried models that are convolutional across time, and that use self-attention across the sequence length, but these did not not perform as well as the MLP-Mixer on our FlyingThings++ tests. Taking advantage of longer and potentially varying temporal context would help the model track through longer periods of ambiguity, and potentially leverage longer-range temporal priors.

7.7 Conclusion

We propose Deep Particle Video (DPV), a method for multi-frame point trajectory estimation through occlusions. Our method combines cost volumes and iterative inference with a multi-frame temporal deep network, which jointly reasons about location and appearance of visual entities across multiple timesteps. We argue that optical flow, particle videos, and feature matches cover different areas in the spectrum of pixel-level correspondence tasks. Particle videos benefit from temporal context, which matching-based methods lack, and can also survive multi-frame occlusions, which is missing in flow-based methods. Given how tremendously useful optical flow and feature matching have been for driving progress in video understanding, we hope the proposed multi-frame trajectories will spark interest in architectures and datasets designed for longer-range fine-grained correspondences.

Chapter 8

Conclusion and future work

This thesis proposes methods for computer vision architectures to estimate amodal scene representations, modelling space and time in a scene-centric space, as opposed to modelling in pixel-space. The underlying idea is that computer vision models are more effective when they assume certain basic facts about the world: it is 3D, it obeys physical and geometric constraints, and it is largely made up of entities which persist across time. By incorporating inductive biases and supervision signals derived from these basic facts, we can deliver improvements in object detection and tracking, and provide a platform for building increasingly useful structured representations of the world.

In our ongoing and future work, we are actively considering the following problems:

- For the geometry-based 3D mapping work, the primary goal is to scale up. Chapter 5 is a first step toward practical large-scale learning from unlabelled data, by learning from real data captured by a sensor platform on a moving vehicle. It should be possible to push this much further, since such sensor platforms quickly capture terabytes of diverse data simply by driving around. It should be possible to merge the Lidar-completion-based learning with the contrastive learning strategy pursued in the other chapters. In particular, we are interested in using the multi-modal aspect of the data, and temporal information, to drive prediction tasks across modalities and across time.
- For driving-based representations, such as the one explored in Chapter 6, it would be interesting to learn representations which optimize a behavioral end-task, rather than a vision-based one. For example, Philion et al. [51, 120] recently proposed a metric to evaluate vision systems based on how *plans* generated from the estimates differ from plans that would be generated by ground-truth information. End-task learning, with scene/object/point representations at the bottleneck, would be very exciting future work.
- For the geometry-free tracking work, the goals are to track particles for arbitrary

timespans, improve individual point tracking by tracking supporters [50], and track simultaneously at multiple levels of abstraction. Besides the direct benefits of fine-grained correspondence, part of the motivation here is that higher level abstractions become easier when the low-level correspondences are worked out. For example, this work could extend to grouping pixels into objects, grouping object instances into higher-level groups, and even distinguishing "stuff" categories from "thing" categories.

- The geometry-free "particle" representation can be naturally merged with the geometry-based scene representations, to serve as a tracker and forecaster, for entities of interest. An excellent outcome here would be a fully persistent representation, where the background is propagated across time using egomotion, and all detected non-background elements are propagated with forecasters of varying complexity (depending on the complexity and importance of each target).

Overall, we argue that computer vision models should treat their input data as mere projections of a physical 3D world—where things have complete shapes, and persist over time—and seek to describe that complete world, instead of merely describing the input. From this perspective, "describing what we see" is only a small part of the vision problem; we also need to describe what we cannot see.

Bibliography

[1] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *Proceedings of the IEEE international conference on computer vision*, pages 37–45, 2015. 25

[2] Leopoldo Armesto, Javier Minguez, and Luis Montesano. A generalization of the metric-based iterative closest point technique for 3d scan matching. In *2010 IEEE International Conference on Robotics and Automation*, pages 1367–1372. IEEE, 2010. 51

[3] Keni Bernardin, Alexander Elbs, and Rainer Stiefelhagen. Multiple object tracking performance metrics and evaluation in a smart room environment. In *Sixth IEEE International Workshop on Visual Surveillance, in conjunction with ECCV*, volume 90, page 91. Citeseer, 2006. 35

[4] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*, pages 850–865. Springer, 2016. 33

[5] Benjamin Biggs, Thomas Roddick, Andrew Fitzgibbon, and Roberto Cipolla. Creatures great and SMAL: Recovering the shape and motion of animals from video. In *Asian Conference on Computer Vision*, pages 3–19. Springer, 2018. 91, 96

[6] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2019. 41

[7] Alla Brodski, Georg-Friedrich Paasch, Saskia Helbling, and Michael Wibral. The faces of predictive coding. *Journal of Neuroscience*, 35(24):8997–9006, 2015. 7, 25

[8] Rodney A Brooks. Intelligence without reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991. 10

[9] Thomas Brox and Jitendra Malik. Object segmentation by long term analysis of point trajectories. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *ECCV*, pages 282–295, 2010. 10, 14, 25

[10] Thomas Brox and Jitendra Malik. Large displacement optical flow: Descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:500–513, 2011. 85

[11] 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:1903.11027*, 2019. 66, 74

[12] Yigit Baran Can, Alexander Liniger, Ozan Unal, Danda Paudel, and Luc Van Gool. Understanding bird's-eye view of road semantics using an onboard camera. *IEEE Robotics and Automation Letters*, 7(2):3302–3309, 2022. 69, 70

[13] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021. 83, 93, 94, 95, 96, 97

[14] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. *arXiv:1512.03012*, 2015. 47

[15] Haiwei Chen, Shichen Liu, Weikai Chen, Hao Li, and Randall Hill. Equivariant point network for 3d point cloud analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14514–14523, 2021. 58

[16] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020. 28

[17] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019. 39, 41, 58

[18] Ricson Cheng, Ziyan Wang, and Katerina Fragkiadaki. Geometry-aware recurrent neural networks for active visual recognition. In *NIPS*, 2018. 70

[19] Anil Cheriyadat and Richard J. Radke. Non-negative matrix factorization of partial track data for motion segmentation. In *ICCV*, 2009. 25

[20] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion, 2020. 39, 41

[21] Kashyap Chitta, Aditya Prakash, and Andreas Geiger. Neat: Neural attention fields for end-to-end autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15793–15803, 2021. 71

[22] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau,

Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014. 10

[23] Peng Chu, Jiang Wang, Quanzeng You, Haibin Ling, and Zicheng Liu. Spatial-temporal graph transformer for multiple object tracking. *arXiv*, 2021. 4

[24] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016. 58

[25] J. Costeira and T. Kanade. A multi-body factorization method for motion analysis. *ICCV*, 1995. 25

[26] Geoffrey Cross and Andrew Zisserman. Surface reconstruction from multiple views using apparent contours and surface texture. In *Confluence of computer vision and computer graphics*, pages 25–47. Springer, 2000. 65

[27] Angela Dai, Christian Diller, and Matthias Nießner. Sg-nn: Sparse generative neural networks for self-supervised scene completion of rgb-d scans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 849–858, 2020. 56, 58

[28] Dengxin Dai, Yujian Wang, Yuhua Chen, and Luc Van Gool. Is image super-resolution helpful for other vision tasks? In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9. IEEE, 2016. 58

[29] Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas J Guibas. Vector neurons: A general framework for so (3)-equivariant networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12200–12209, 2021. 58

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009. 74

[31] Victor Lempitsky Dmitry Ulyanov, Andrea Vedaldi. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *CVPR*, 2017. 72

[32] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, 2015. 16

[33] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *ICCV*, 2015. 85

[34] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017. 8, 16, 24, 30, 47

[35] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 88, 93

[36] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360 (6394):1204–1210, 2018. xiii, 8, 9, 10, 17, 25, 39, 40, 42, 45, 47, 52, 53

[37] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *CoRL*, 2018. xii, 24, 32, 35, 40, 42, 48, 49

[38] Katerina Fragkiadaki and Jianbo Shi. Detection free tracking: Exploiting motion and topology for segmenting and tracking under entanglement. In *CVPR*, 2011. 25

[39] Katerina Fragkiadaki, Pablo Arbelaez, Panna Felsen, and Jitendra Malik. Learning to segment moving objects in videos. In *CVPR*, 2015. 10

[40] Steven L Franconeri and Daniel J Simons. Moving and looming stimuli capture attention. *Perception & psychophysics*, 65(7):999–1010, 2003. 23

[41] Karl Friston. Learning and inference in the brain. *Neural Networks*, 16(9): 1325–1352, 2003. 2, 7, 9, 25

[42] Dashan Gao, Vijay Mahadevan, and Nuno Vasconcelos. On the plausibility of the discriminant center-surround hypothesis for visual saliency. *Journal of vision*, 8, 2008. 15

[43] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. 16, 17, 24, 30, 47, 57, 64, 91, 94

[44] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape, 2020. 39, 48

[45] Hugo Germain, Vincent Lepetit, and Guillaume Bourmaud. Visual correspondence hallucination: Towards geometric reasoning. In *arXiv Preprint*, 2021. 86

[46] James J. Gibson. *The Ecological Approach to Visual Perception.* Houghton Mifflin, 1979. 21, 25

[47] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh R-CNN. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9785–9795, 2019. 41

[48] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-

Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *NIPS*, pages 2672–2680, 2014. 41

[49] Nikhil Gosala and Abhinav Valada. Bird's-eye-view panoptic segmentation using monocular frontal view images. *IEEE Robotics and Automation Letters*, 2022. 69, 70

[50] Helmut Grabner, Jiri Matas, Luc Van Gool, and Philippe Cattin. Tracking the invisible: Learning where the object might be. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1285–1292. IEEE, 2010. 104

[51] Yiluan Guo, Holger Caesar, Oscar Beijbom, Jonah Philion, and Sanja Fidler. The efficacy of neural planning metrics: a meta-analysis of pkl on nuscenes. *arXiv preprint arXiv:2010.09350*, 2020. 103

[52] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, pages 2616–2625, 2017. 10

[53] Tengda Han, Weidi Xie, and Andrew Zisserman. Video representation learning by dense predictive coding, 2019. 39

[54] Adam W Harley, Shrinidhi K Lakshmikanth, Fangyu Li, Xian Zhou, Hsiao-Yu Fish Tung, and Katerina Fragkiadaki. Learning from unlabelled videos using contrastive predictive neural 3d mapping. In *ICLR*, 2020. 4, 24, 25, 27, 32, 39, 40, 42, 47, 53, 69, 70, 73

[55] Adam W Harley, Shrinidhi K Lakshmikanth, Paul Schydlo, and Katerina Fragkiadaki. Tracking emerges by looking around static scenes, with neural 3d mapping. In *ECCV*, 2020. 4, 39, 40, 48, 49, 51, 53, 73

[56] Adam W Harley, Zhaoyuan Fang, and Katerina Fragkiadaki. Particle videos revisited: Tracking through occlusions using point trajectories. *arXiv:2204.04153*, 2022. 4

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 13, 14, 73, 93, 96

[58] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020. 28, 46

[59] Noureldin Hendy, Cooper Sloan, Feng Tian, Pengfei Duan, Nick Charchut, Yuesong Xie, Chuang Wang, and James Philbin. Fishing net: Future inference of semantic heatmaps in grids. *arXiv preprint arXiv:2006.09917*, 2020. 69, 70, 74, 75, 76

[60] J. F. Henriques and A. Vedaldi. MapNet: An allocentric spatial memory for mapping environments. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. 10

[61] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 10

[62] Michael Hornacek, Andrew Fitzgibbon, and Carsten Rother. SphereFlow: 6 DoF scene flow from RGB-D pairs. In *CVPR*, 2014. 15

[63] Jun-Ting Hsieh, Bingbin Liu, De-An Huang, Li F Fei-Fei, and Juan Carlos Niebles. Learning to decompose and disentangle representations for video prediction. In *NIPS*, pages 517–526, 2018. 9, 19

[64] Anthony Hu, Zak Murez, Nikhil Mohan, Sofía Dudas, Jeffrey Hawke, Vijay Badrinarayanan, Roberto Cipolla, and Alex Kendall. Fiery: Future instance prediction in bird's-eye view from surround monocular cameras. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15273–15282, 2021. 70, 73, 76, 77

[65] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *arXiv preprint arXiv:1612.01925*, 2016. 85

[66] Allan Jabri, Andrew Owens, and Alexei A Efros. Space-time correspondence as a contrastive random walk. *Advances in Neural Information Processing Systems*, 2020. 86, 93, 96

[67] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, pages 4651–4664. PMLR, 2021. 60

[68] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. In *ICLR*, 2022. 60

[69] Joel Janai, Fatma G"uney, Anurag Ranjan, Michael J. Black, and Andreas Geiger. Unsupervised learning of multi-frame optical flow with occlusions. In *European Conference on Computer Vision (ECCV)*, volume Lecture Notes in Computer Science, vol 11220, pages 713–731. Springer, Cham, September 2018. 85

[70] Dinesh Jayaraman and Kristen Grauman. Learning image representations equivariant to ego-motion. In *Proc. ICCV*, 2015. 25

[71] Wei Jiang, Eduard Trulls, Jan Hosang, Andrea Tagliasacchi, and Kwang Moo

Yi. COTR: Correspondence Transformer for Matching Across Images. In *ICCV*, 2021. 86

[72] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017. xiii, 52

[73] Gaetano Kanizsa. Convexity and symmetry in figure-ground organization. *Vision and artifact*, 1976. 57, 65

[74] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. In *NIPS*, 2017. 9, 10, 17, 25, 41

[75] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018. 25, 41

[76] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv:1705.06950*, 2017. 93

[77] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018. 73

[78] Anna Khoreva, Rodrigo Benenson, Eddy Ilg, Thomas Brox, and Bernt Schiele. Lucid data dreaming for object tracking. In *CVPR Workshops*, 2017. 10

[79] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions, 2018. 41

[80] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013. 41

[81] Adam R. Kosiorek, Hyunjik Kim, Ingmar Posner, and Yee Whye Teh. Sequential attend, infer, repeat: Generative modelling of moving objects. In *NIPS*, 2018. 9, 19

[82] Z. Lai and W. Xie. Self-supervised learning for video correspondence flow. In *BMVC*, 2019. 86

[83] Zihang Lai, Erika Lu, and Weidi Xie. MAST: A memory-augmented self-supervised tracker. *arXiv:2002.07793*, 2020. 32, 83, 86, 93, 96

[84] Shamit Lal, Mihir Prabhudesai, Ishita Mediratta, Adam W Harley, and Katerina Fragkiadaki. CoCoNets: Continuous contrastive 3d scene representations. In *CVPR*, 2021. 4

[85] Hoang-An Le, Thomas Mensink, Partha Das, and Theo Gevers. Novel view synthesis from single images via point cloud transformation, 2020. 39

[86] Hsin-Ying Lee, Jia-Bin Huang, Maneesh Singh, and Ming-Hsuan Yang. Unsupervised representation learning by sorting sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 667–676, 2017. 25

[87] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *arXiv:1504.00702*, 2015. 10

[88] Qi Li, Yue Wang, Yilun Wang, and Hang Zhao. Hdmapnet: A local semantic map learning and evaluation framework. *arXiv preprint arXiv:2107.06307*, 2021. 70

[89] Xiaolong Li, Yijia Weng, Li Yi, Leonidas J Guibas, A Abbott, Shuran Song, and He Wang. Leveraging se (3) equivariance for self-supervised category-level object pose estimation from point clouds. *Advances in Neural Information Processing Systems*, 34, 2021. 58

[90] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Qiao Yu, and Jifeng Dai. Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers. *arXiv preprint arXiv:2203.17270*, 2022. 71, 75, 76, 77

[91] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. Learning efficient point cloud generation for dense 3d object reconstruction, 2017. 41

[92] Buyu Liu, Bingbing Zhuang, Samuel Schulter, Pan Ji, and Manmohan Chandraker. Understanding road layout from videos as a whole. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4414–4423, 2020. 70

[93] Buyu Liu, Bingbing Zhuang, and Manmohan Chandraker. Weakly but deeply supervised occlusion-reasoned parametric layouts. *arXiv preprint arXiv:2104.06730*, 2021. 70

[94] John C Loehlin. *Latent variable models: An introduction to factor, path, and structural analysis.* Lawrence Erlbaum Associates, Inc, 1987. 9

[95] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. Smpl: A skinned multi-person linear model. *ACM Trans. Graph.*, 34(6):248:1–248:16, 2015. 25

[96] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987. 48

[97] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 74

[98] Vijay Mahadevan and Nuno Vasconcelos. Spatiotemporal saliency in dynamic scenes. *TPAMI*, 32, 2010. 15

[99] Lain Matthews, Takahiro Ishikawa, and Simon Baker. The template update problem. *IEEE transactions on pattern analysis and machine intelligence*, 26 (6):810–815, 2004. 33

[100] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *CVPR*, 2016. 84, 91, 92

[101] James L McClelland and David E Rumelhart. An interactive activation model of context effects in letter perception: I. an account of basic findings. *Psychological review*, 88(5):375, 1981. 7, 25

[102] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *CVPR*, 2015. 24

[103] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 39, 41, 58

[104] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv:1301.3781*, 2013. 9

[105] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. *arXiv preprint arXiv:2003.08934*, 2020. 39, 41, 46

[106] Ishan Misra, C. Lawrence Zitnick, and Martial Hebert. Unsupervised learning using sequential verification for action recognition. In *ECCV*, 2016. 25

[107] Bence Nanay. The importance of amodal completion in everyday perception. *i-Perception*, 9(4):2041669518788887, 2018. doi: 10.1177/2041669518788887. PMID: 30109014. 1, 42

[108] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7588–7597, 2019. 41

[109] David Novotny, Diane Larlus, and Andrea Vedaldi. Learning 3d object categories by looking around them. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5218–5227, 2017. 41

[110] David Novotny, Roman Shapovalov, and Andrea Vedaldi. Canonical 3d deformer maps: Unifying parametric and non-parametric methods for dense weakly-supervised category reconstruction, 2020. 39

[111] P. Ochs and T. Brox. Object segmentation in video: A hierarchical variational approach for turning point trajectories into dense regions. In *ICCV*, 2011. 10, 14, 25

[112] Bruno Olshausen. Perception as an inference problem. In *The Cognitive Neurosciences*. MIT Press, 2013. 25

[113] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv:1807.03748*, 2018. 9, 25, 28, 39, 46

[114] Bowen Pan, Jiankai Sun, Ho Yin Tiga Leung, Alex Andonian, and Bolei Zhou. Cross-view semantic segmentation for sensing surroundings. *IEEE Robotics and Automation Letters*, 5(3):4867–4873, 2020. 70, 76

[115] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019. 39, 41, 58

[116] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 91

[117] Aftab E Patla. Visual control of human locomotion. In *Advances in psychology*, volume 78, pages 55–97. Elsevier, 1991. 2, 7, 11, 27

[118] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *ECCV*, 2020. 39, 40, 41, 44, 46, 48, 58

[119] Jonah Philion and Sanja Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. In *European Conference on Computer Vision*, pages 194–210. Springer, 2020. 69, 70, 73, 76

[120] Jonah Philion, Amlan Kar, and Sanja Fidler. Learning to evaluate perception models using planner-centric metrics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14055–14064, 2020. 103

[121] Yair Pinto, Simon van Gaal, Floris P de Lange, Victor AF Lamme, and Anil K Seth. Expectations accelerate entry of visual stimuli into awareness. *Journal of Vision*, 15(8):13–13, 2015. 7, 25

[122] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv:1704.00675*, 2017. 25

[123] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool. The 2017 DAVIS challenge on video object segmentation. *arXiv:1704.00675*, 2017. 85, 96

[124] Stefan Popov, Pablo Bauszat, and Vittorio Ferrari. Corenet: Coherent 3d scene reconstruction from a single rgb image, 2020. 39, 40, 41, 46, 48

[125] Adrien Poulenard, Marie-Julie Rakotosaona, Yann Ponty, and Maks Ovsjanikov. Effective rotation-invariant point cnn with spherical harmonics kernels. In *2019 International Conference on 3D Vision (3DV)*, pages 47–56. IEEE, 2019. 58

[126] Charles R. Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 47, 50

[127] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9277–9286, 2019. 40, 47, 49, 50

[128] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008. 33

[129] Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79, 1999. 2, 7, 9, 25

[130] Meytal Rapoport-Lavie and Dan Raviv. It's all around you: Range-guided cylindrical network for 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2992–3001, 2021. 58

[131] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015. 13

[132] Zhile Ren, Orazio Gallo, Deqing Sun, Ming-Hsuan Yang, Erik B Sudderth, and Jan Kautz. A fusion approach for multi-frame optical flow estimation. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2019. 84, 85

[133] Gernot Riegler and Vladlen Koltun. Free view synthesis, 2020. 39, 41

[134] Lawrence Roberts. *Machine perception of three-dimensional solids*. PhD thesis, MIT, 1965. 10, 25

[135] 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. xvi, 70, 76

[136] Thomas Roddick, Benjamin Biggs, Daniel Olmeda Reino, and Roberto Cipolla. On the road to large-scale 3d monocular scene reconstruction using deep implicit functions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2875–2884, 2021. 71

[137] Avishkar Saha, Oscar Mendez Maldonado, Chris Russell, and Richard Bowden. Translating images into maps. *arXiv preprint arXiv:2110.00966*, 2021. 69, 71, 76, 77

[138] Avishkar Saha, Oscar Mendez, Chris Russell, and Richard Bowden. Enabling spatio-temporal aggregation in birds-eye-view vehicle estimation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5133–5139. IEEE, 2021. 69, 70

[139] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2304–2314, 2019. 9, 41

[140] Agustín Salgado and Javier Sánchez. Temporal constraints in large optical flow estimation. In *International Conference on Computer Aided Systems Theory*, pages 709–716. Springer, 2007. 85

[141] P. Sand and S. Teller. Particle video: Long-range motion estimation using point trajectories. In *CVPR*, volume 2, pages 2195–2202, 2006. 3, 83, 86

[142] Peter H Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966. 48

[143] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *CVPR*, 2015. 14

[144] Samuel Schulter, Menghua Zhai, Nathan Jacobs, and Manmohan Chandraker. Learning to look around objects for top-view representations of outdoor scenes. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 787–802, 2018. 69, 70

[145] Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997. 7, 25

[146] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *arXiv preprint arXiv:2007.02442*, 2020. 41

[147] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. DeepVoxels: Learning persistent 3D feature embeddings. In *CVPR*, 2019. 9, 10, 39, 41, 42, 45

[148] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019. 70

[149] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *arXiv*, 2020. 39, 41, 42, 45, 52, 53

[150] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 1100612. International Society for Optics and Photonics, 2019. 91

[151] Kihyuk Sohn. Improved deep metric learning with multi-class N-pair loss objective. In *NIPS*, pages 1857–1865, 2016. 14, 28

[152] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *CVPR*, 2016. 14

[153] Kasey C Soska and Scott P Johnson. Development of three-dimensional object completion in infancy. *Child development*, 79(5):1230–1236, 2008. 7

[154] Elizabeth S Spelke, J Mehler, M Garrett, and E Walker. Perceptual knowledge of objects in infancy. In NJ: Erlbaum Hillsdale, editor, *Perspectives on mental representation*, chapter 22. Erlbaum, 1982. 7

[155] D. Sun, S. Roth, and M. J. Black. Secrets of optical flow estimation and their principles. In *CVPR*, 2010. 15

[156] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *CVPR*, 2018. 13, 14, 18, 85

[157] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020. 67

[158] Narayanan Sundaram, Thomas Brox, and Kurt Keutzer. Dense point trajectories by GPU-accelerated large displacement optical flow. In *ECCV*, 2010. 85

[159] Ramana Sundararaman, Cedric De Almeida Braga, Eric Marchand, and Julien Pettre. Tracking pedestrian heads in dense crowd. In *CVPR*, pages 3865–3875,

2021. 91, 95

[160] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 72

[161] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Single-view to multi-view: Reconstructing unseen views with a convolutional network. In *ECCV*, 2016. 9, 25, 41

[162] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *European Conference on Computer Vision*, pages 402–419. Springer, 2020. 83, 84, 85, 87, 88, 89, 92, 93, 94, 95, 96

[163] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. `https://github.com/princeton-vl/RAFT`, 2020. 90

[164] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, Rohit Pandey, Sean Fanello, Gordon Wetzstein, Jun-Yan Zhu, Christian Theobalt, Maneesh Agrawala, Eli Shechtman, Dan B Goldman, and Michael Zollhöfer. State of the art on neural rendering, 2020. 41

[165] Josh Tobin, OpenAI Robotics, and Pieter Abbeel. Geometry-aware neural rendering, 2019. 42

[166] Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-mixer: An all-mlp architecture for vision. In *NeurIPS*, 2021. 60, 88

[167] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: A factorization method. *Int. J. Comput. Vision*, 9(2):137–154, 1992. 25

[168] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2626–2634, 2017. 25, 39, 41

[169] Shubham Tulsiani, Saurabh Gupta, David F Fouhey, Alexei A Efros, and Jitendra Malik. Factoring shape, pose, and layout from the 2d image of a 3d scene. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 302–310, 2018. 39

[170] Hsiao-Yu Fish Tung, Adam W Harley, William Seto, and Katerina Fragkiadaki. Adversarial inverse graphics networks: Learning 2d-to-3d lifting and image-to-image translation from unpaired supervision. In *International Conference on*

Computer Vision (ICCV), pages 4364–4372. IEEE, 2017. 25

[171] Hsiao-Yu Fish Tung, Ricson Cheng, and Katerina Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2595–2603, 2019. xiii, 8, 9, 10, 11, 16, 17, 18, 20, 24, 25, 27, 40, 41, 42, 47, 52

[172] Jack Valmadre, Luca Bertinetto, Joao F Henriques, Ran Tao, Andrea Vedaldi, Arnold WM Smeulders, Philip HS Torr, and Efstratios Gavves. Long-term tracking in the wild: A benchmark. In *ECCV*, pages 670–685, 2018. 93

[173] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *NIPS*, pages 4790–4798, 2016. 9, 41

[174] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 60, 88

[175] Sudheendra Vijayanarasimhan, Susanna Ricco, Cordelia Schmid, Rahul Sukthankar, and Katerina Fragkiadaki. Sfm-net: Learning of structure and motion from video. *arXiv:1704.07804*, 2017. 25

[176] Carl Vondrick, Abhinav Shrivastava, Alireza Fathi, Sergio Guadarrama, and Kevin Murphy. Tracking emerges by colorizing videos. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 391–408, 2018. 24, 25, 29, 30, 32

[177] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *European Conference on Computer Vision*, pages 835–851. Springer, 2016. 25

[178] Hengli Wang, Peide Cai, Yuxiang Sun, Lujia Wang, and Ming Liu. Learning interpretable end-to-end vision-based motion planning for autonomous driving with optical flow distillation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13731–13737. IEEE, 2021. 70

[179] Qianqian Wang, Xiaowei Zhou, Bharath Hariharan, and Noah Snavely. Learning feature descriptors using camera pose supervision. In *Proc. European Conference on Computer Vision (ECCV)*, 2020. 86

[180] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2015. 24, 25

[181] Xiaolong Wang, Allan Jabri, and Alexei A. Efros. Learning correspondence from the cycle-consistency of time. In *CVPR*, 2019. 83, 86, 93, 96

[182] Xiaolong Wang, Allan Jabri, and Alexei A Efros. Learning correspondence from the cycle-consistency of time. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2566–2576, 2019. 24

[183] Zhihao Wang, Jian Chen, and Steven CH Hoi. Deep learning for image super-resolution: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3365–3387, 2020. 58

[184] Frank M White. *Fluid mechanics*. Tata McGraw-Hill Education, 1979. 2

[185] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. SynSin: End-to-end view synthesis from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 39

[186] Olivia Wiles, Sebastien Ehrhardt, and Andrew Zisserman. Co-attention for conditioned image matching. In *CVPR*, 2021. 86

[187] Laurenz Wiskott and Terrence J Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770, 2002. 25

[188] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893. IEEE, 2018. 58

[189] Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. Sampling matters in deep embedding learning. In *ICCV*, pages 2840–2848, 2017. 14

[190] Jiajun Wu, Tianfan Xue, Joseph J. Lim, Yuandong Tian, Joshua B. Tenenbaum, Antonio Torralba, and William T. Freeman. Single image 3d interpreter network. In *ECCV*, pages 365–382, 2016. 25

[191] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920. IEEE Computer Society, 2015. 25

[192] Saining Xie, Jiatao Gu, Demi Guo, Charles R. Qi, Leonidas J. Guibas, and Or Litany. Pointcontrast: Unsupervised pre-training for 3d point cloud understanding, 2020. 40, 42, 46, 47, 48, 49, 51, 53

[193] Jiarui Xu and Xiaolong Wang. Rethinking self-supervised correspondence learning: A video frame-level similarity perspective. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10075–10085, October 2021. 93, 96

[194] Ning Xu, Linjie Yang, Yuchen Fan, Jianchao Yang, Dingcheng Yue, Yuchen Liang, Brian Price, Scott Cohen, and Thomas Huang. Youtube-vos: Sequence-to-sequence video object segmentation. In *ECCV*, pages 585–601, 2018. 93

[195] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision, 2017. 41

[196] Charig Yang, Hala Lamdouar, Erika Lu, Andrew Zisserman, and Weidi Xie. Self-supervised video object segmentation by motion grouping. In *ICCV*, 2021. 86

[197] Gengshan Yang, Deqing Sun, Varun Jampani, Daniel Vlasic, Forrester Cole, Huiwen Chang, Deva Ramanan, William T Freeman, and Ce Liu. LASR: Learning articulated shape reconstruction from a monocular video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15980–15989, 2021. 96

[198] Gengshan Yang, Deqing Sun, Varun Jampani, Daniel Vlasic, Forrester Cole, Ce Liu, and Deva Ramanan. Viser: Video-specific surface embeddings for articulated 3d shape reconstruction. In *NeurIPS*, 2021. 96

[199] Weixiang Yang, Qi Li, Wenxi Liu, Yuanlong Yu, Yuexin Ma, Shengfeng He, and Jia Pan. Projecting your view attentively: Monocular road scene layout estimation via cross-view transformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15536–15545, 2021. 70

[200] Li Yi, Boqing Gong, and Thomas Funkhouser. Complete & label: A domain adaptation approach to semantic segmentation of lidar point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15363–15373, 2021. 55, 57, 66

[201] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11784–11793, 2021. 64, 67, 69, 75

[202] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Multimodal virtual point 3d detection. *Advances in Neural Information Processing Systems*, 34, 2021. 55, 57, 69

[203] Jason J. Yu, Adam W. Harley, and Konstantinos G. Derpanis. Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness. In *ECCV*, 2016. 15

[204] A. Yuille and D. Kersten. Vision as Bayesian inference: analysis by synthesis? *Trends in Cognitive Sciences*, 10:301–308, 2006. 25

[205] Yang Zhang, Zixiang Zhou, Philip David, Xiangyu Yue, Zerong Xi, Boqing Gong, and Hassan Foroosh. Polarnet: An improved grid representation for online lidar point clouds semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9601–9610,

2020. 58

[206] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, 2017. 25

[207] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. Cylindrical and asymmetrical 3d convolution networks for lidar segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9939–9948, 2021. 58