Learning Object-Oriented Models of the Visual World

Eric Crawford

School of Computer Science

McGill University, Montreal

April, 2021

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Abstract

Objects are a central abstraction in human mental life. Indeed, developmental psychology tells us that humans and many animals have modular, developmentally inevitable systems for discovering, detecting, and tracking objects. In building artificially intelligent agents, we would do well to find ways to equip our agents with a similar propensity for constructing object-oriented representations of the world. In particular, to maximize autonomy of future systems, we should seek ways of learning object-oriented representations of the world *without supervision*; this restriction rules out the plethora of supervised object-detection and tracking techniques that have emerged in recent years. To meet the challenge of unsupervised object discovery, detection and tracking, we formalize the concept of an Object-Oriented World Model (OOWM), and thereby unify a range of similar models from the recent literature. OOWMs are a class of temporal latent variable models which permit learning to detect and track objects, and even to predict future object trajectories, all without object-level supervision. Our primary contribution in this thesis is to significantly expand the scope of OOWMs. We first show how to build OOWMs that are able to handle large, densely packed scenes containing many objects. We then go on to show how to build OOWMs which, unlike the majority of past models, properly model the positions of objects in 3D space. In the end, this thesis greatly expands the range and complexity of environments in which OOWMs can usefully be applied.

Abrégé

Les objets sont au coeur du processus d'abstraction mentale pour l'être humain. En effet, des études sur la psychologie du développement montrent que les humains et beaucoup d'animaux ont un système modulaire, conçus pour découvrir, détecter et reconnaître des objets. En développant des agents artificiels intelligents nous devrions les équiper de la même propension à construire la représentation des objets orientés dans le monde. En particulier, pour optimiser l'autonomie des futurs systèmes, nous devrions trouver un moyen d'apprendre la représentation des objets orientés du monde en *absence de signal supervisé*; cette restriction est à la base de la logique de pléthore de méthodes de détection et reconnaissance d'objets qui sont devenues très populaires récemment. Pour répondre à ce challenge de découverte, détection et reconnaissance d'objets non supervisés, nous formalisons le concept de "Object-Oriented World Model" (OOWM), et ainsi unifions un éventail de modèles similaires dans la littérature. Les OOWMs sont des modèles de variables latentes temporelles qui permettent d'apprendre à détecter et suivre les objets, et même de prédire leur trajectoire future, tout cela sans supervision à l'échelle de l'objet. Notre principale contribution dans cette thèse est d'étendre la portée d'application des OOWMs. Nous montrons d'abord comment construire un OOWM capable de traiter des lieux denses et de grandes dimensions contenant beaucoup d'objets. Nous montrons ensuite comment construire les OOWMs qui, contrairement à la majorité d'anciennes méthodes, modélisent spécifiquement les positions de l'objet dans un espace trois dimensions. Finalement, cette thèse étend grandement l'éventail et complexité des environnements dans lesquels le OOWM peut être appliqué.

Contribution to Original Knowledge

This thesis is about developing techniques for modeling videos as collections of objects. My primary modeling tool is a kind of temporal Variational Autoencoder equipped with an object-oriented latent representation which I call an Object-Oriented World Model (OOWM). Under certain conditions, OOWMs are able to learn to build object-oriented representations of perceptual data without supervision, to predict future object trajectories, and to render collections of objects into scenes.

At the time this research began, OOWMs were a relatively novel field of study, and the range of environments in which they would yield useful results was relatively narrow. Indeed, most models were restricted to small black-and-white images containing few objects. My project in this thesis is primarily one of extending the scope of OOWMs: first to images and videos that are spatially large and contain many objects, and then to modeling 3D worlds. My contributions can be summarized as follows:

1. I use insights from supervised object detection to design an image-only OOWM which is significantly better than past models at handling large cluttered scenes containing many objects.

2. I use those same insights from supervised object detection to develop a full-fledged OOWM for videos which achieves significantly improved performance on large, many-object videos compared with past approaches.

3. I develop one of the first OOWMs capable of learning 3D object-oriented representations of 3D scenes without supervision.

Contribution of Authors

All research presented herein was carried out by me, with the help of valuable technical guidance from my supervisor Joelle Pineau. Several chapters in this thesis are based on previously published papers:

- Chapter 4 is based on a paper presented at AAAI 2019 [28].
- Chapter 5 is based on a paper presented at AAAI 2020 [29].
- Chapter 6 is based in part on a workshop paper presented at the Object-Oriented Learning Workshop at ICML 2020 [30], and on a subsequent full-length paper under review at ICML 2021.

Acknowledgements

This thesis would not have been possible without the support of a large cast of friends and family. I am grateful, in particular, to the following people: to my wife Erika for her constant love and support, and for putting up with my bad habit of working through the night around conference deadlines; to my parents Janet and Kevin and brother Michael, for their wise advice and unconditional support in any number of different aspects of life; to my friends from Dundas and Waterloo, especially Colin Davidson, Bill, Johnny, Matt and Theresa Valeriote, Kevin Emerson, Adam Carter, Ian Dransutavicius, Kyle Dillane, and Shuang Chen, for keeping my spirits high and keeping me connected to the world outside of academia; to John Morris and Rosa Rodriguez-Mota for being the world's best roommates; and to my sports pals in Montreal, especially Nick Bohn, Rob Keyes, Matt Rigby, Kunal Tiwari, Aaron Moscaro, and Volker Hofmann, without whom I would have gone insane long before the end.

This thesis also owes its existence to a number of people in the RLLab at McGill. First and foremost, I want to thank my supervisor Joelle Pineau, for giving me the freedom to explore a wide range of research topics (and having faith that I would one day settle on a topic), for an unending stream of sage wisdom and technical guidance, and for simply being a wonderful human being. I am also extremely thankful for the array of talented people I have had the pleasure of working alongside, including Thang Doan, Yue Dong, Srinivas Venkattaramanujam, Gautam Bhattacharya, Boyu Wang, and Angus Leigh.

Table of Contents

Abstract i

Abrégé ii

Contribution to Original Knowledge iii

Contribution of Authors iv

Acknowledgements v

1 Introduction 1

1.1 Deep Learning, Representations and Models . . . 3
1.2 Inductive Biases in Computer Vision . . . 4
1.3 An Inductive Bias for Objects . . . 5
1.4 Expanding the Scope of Object-Oriented World Models . . . 8
1.5 Thesis Structure . . . 10

2 Technical Background 11

2.1 Deep Learning . . . 11
2.2 Neural Components . . . 13
2.2.1 Multi-layer Perceptrons . . . 13
2.2.2 Convolutional Neural Networks . . . 16
2.2.3 Spatial Transformers . . . 22
2.3 Variational Autoencoders . . . 25
2.3.1 Overview . . . 26
2.3.2 Parameterizing Distributions with Neural Networks . . . 30
2.3.3 The Reparameterization Trick . . . 31
2.3.4 Reparameterizing Discrete Latent Variables . . . 32
2.4 Supervised Object Detection with Deep Convolutional Neural Networks . . . 34

2.4.1 Problem Definition . . . 35
2.4.2 Measuring Performance: Average Precision . . . 36
2.4.3 Proposal–based Object Detection . . . 38
2.4.4 Single Shot Object Detection . . . 39
2.4.5 Architectural Concepts for Object Detection . . . 40
2.5 Conclusion . . . 44

3 Object-Oriented World Models **46**
3.1 Optimizing Agent Performance . . . 46
3.2 World Models . . . 47
3.3 Object-Oriented World Models . . . 50
3.3.1 Objects . . . 51
3.3.2 Object-Oriented Representations and World Models . . . 53
3.3.3 The Advantages of Object-Oriented Representations . . . 55
3.3.4 Strategies for Learning Object-Oriented Representations . . . 58
3.4 Literature Review . . . 62
3.4.1 Scene Mixture Models . . . 63
3.4.2 Spatial Attention Models. . . . 66
3.4.3 Other Approaches . . . 68
3.5 Outstanding Issues . . . 69
3.5.1 Scaling to Large, Many-Object Scenes . . . 69
3.5.2 3D Object-Oriented World Models . . . 70
3.6 Conclusion . . . 72

4 Scaling Unsupervised Object Detection Through Spatial Invariance **73**
4.1 Introduction . . . 73
4.2 Unsupervised Object Detection . . . 75
4.3 Related Work: Attend, Infer, Repeat . . . 76
4.4 Spatially Invariant Attend, Infer, Repeat . . . 78
4.4.1 Object Representation . . . 78
4.4.2 Convolutional, Object-Detecting Inference Network . . . 79
4.4.3 Object-Rendering Generative Network . . . 87
4.4.4 Prior Distribution . . . 90
4.4.5 Training . . . 91
4.5 Scaling Properties of SPAIR . . . 91
4.5.1 Comparison with AIR . . . 91

4.5.2 Out-of-Distribution Generalization . . . 93
4.5.3 Spatial Invariance vs Spatial Equivariance . . . 94
4.6 Experiments . . . 95
4.6.1 Baseline Algorithm: ConnComp . . . 95
4.6.2 Comparison with AIR . . . 95
4.6.3 Generalization . . . 97
4.6.4 Downstream Tasks . . . 98
4.6.5 Space Invaders . . . 101
4.7 Discussion . . . 103
4.7.1 Grid Cell and Receptive Field Sizes . . . 103
4.7.2 Sample Efficiency . . . 105
4.7.3 Objects and Backgrounds . . . 106
4.8 Retrospective . . . 108

5 Exploiting Spatial Invariance for Scalable Unsupervised Object Tracking 109
5.1 Introduction . . . 109
5.2 Spatially Invariant, Label-free Object Tracking . . . 111
5.2.1 Object Representation . . . 112
5.2.2 Overview . . . 113
5.2.3 Discovery . . . 114
5.2.4 Propagation . . . 119
5.2.5 Selection . . . 127
5.2.6 Rendering . . . 128
5.2.7 Prior Distribution . . . 128
5.2.8 Training . . . 129
5.3 Experiments . . . 130
5.3.1 Metrics . . . 130
5.3.2 Baseline Algorithm: ConnComp . . . 131
5.3.3 Scattered MNIST . . . 131
5.3.4 Scattered Shapes . . . 134
5.3.5 Atari . . . 136
5.4 Discussion . . . 137
5.5 Retrospective . . . 138

6 Unsupervised Object Detection and Tracking in 3D 139
6.1 Introduction . . . 139

6.2 Related Work . . . 141
6.3 Problem Definition . . . 142
6.4 Learning to Model Static Scene Elements . . . 144
6.5 Learning to Model Dynamic Objects . . . 146
6.5.1 3D Object Representation . . . 146
6.5.2 Overview . . . 148
6.5.3 Discovery . . . 148
6.5.4 Propagation . . . 155
6.5.5 Selection . . . 159
6.5.6 Rendering . . . 162
6.5.7 Training the Depth Predictor . . . 164
6.5.8 Prior Propagation . . . 166
6.6 Training . . . 167
6.7 Experiments . . . 168
6.7.1 Metrics . . . 169
6.7.2 Simulated Object Picking . . . 169
6.7.3 First-Person Maze Navigation . . . 172
6.8 Discussion . . . 179
6.8.1 Depth Ambiguity and Phantom Object Motion . . . 183
6.8.2 Handling Stochastic Objects . . . 185
6.8.3 Alternative Models of Object Motion . . . 187
6.8.4 Modeling 3D Object Shape . . . 188
6.9 Retrospective . . . 190

7 Conclusion **191**
7.1 Summary of Contributions . . . 191
7.2 Limitations . . . 195
7.3 Open Questions . . . 196

A Prior for *pres* Latent Variables **201**
A.1 SPAIR . . . 201
A.2 SILOT . . . 203
A.3 3DOM . . . 203

B Experiment Details for SPAIR **205**
B.1 Base Model Details . . . 205
B.1.1 AIR / DAIR . . . 205

B.1.2 SPAIR . . . 206
B.1.3 ConnComp . . . 208
B.2 Experiment Details . . . 209
B.2.1 Comparison with AIR . . . 209
B.2.2 Generalization . . . 209
B.2.3 Addition . . . 209
B.2.4 SET . . . 210

C Experiment Details for SILOT **212**
C.1 Base Model Details . . . 212
C.1.1 Training Details . . . 212
C.1.2 SQAIR . . . 212
C.1.3 SILOT . . . 213
C.1.4 ConnComp . . . 213
C.2 Experiment Details . . . 216
C.2.1 Scattered MNIST . . . 216
C.2.2 Scattered Shapes . . . 217

D Experiment Details for 3DOM **218**
D.1 Base Model Details . . . 218
D.1.1 Scene Representation Networks . . . 218
D.1.2 3DOM . . . 220
D.1.3 2DOM . . . 221
D.2 Experiment Details . . . 221
D.2.1 Metrics . . . 221
D.2.2 Simulated Object Picking . . . 225
D.2.3 First Person Maze Navigation . . . 226

Chapter 1

Introduction

The overarching goal of computer vision is to build systems that can behave adaptively in physical environments, basing their actions and decisions only on raw visual input (i.e. light bouncing off surfaces in the world). Three crucial abilities for such a system are: constructing an internal representation of the physical scene based on the perceptual data, updating that representation as new perceptual data becomes available, and predicting how the scene (and the system's representation thereof) will unfold in the future. In designing a system with these abilities, it is important to ensure that the representation itself, as well as the components that extract, update and forecast the representation, are optimized for the structure of the physical world. In deciding which aspects of the physical world are most relevant, it may be instructive to look for guidance in intelligent systems in nature that perform similar tasks, and one of the most conspicuous examples of such a system is the human brain.

The human brain appears to take significant advantage of the fact that the physical world can be divided up into discrete objects. Through a complex process of grouping and inference, our brains transduce the blooming, buzzing confusion of raw visual data into highly structured representations that correspond to individual objects in the world. These "object-oriented" representations are central to human mental life, and serve as input to a

wide-range of downstream cognitive processes. The brain's use of objects is summed up well by (Carey, 2009) [19, p. 70]:

> Sensory input is continuous. The array of light on the retina is not segregated into individual objects. Yet distinct individuals are provided by visual cognition as input into many other perceptual and cognitive processes. It is individuals we categorize into kinds; it is individuals we reach for; it is individuals we enumerate; it is individuals among which we represent spatial relations such as "behind" and "inside"; and it is individuals that enter into our representations of causal interactions and events.

Beyond simply extracting objects from visual input, we are also able to update our object-oriented representations over time (i.e. object tracking) [127, 128] and to use them in prediction tasks [141]. Object-oriented representations also support certain invaluable skills such as object permanence, the ability to represent objects as existing even while they are not visible [7].

This thesis is about building artificial systems that can learn to extract, maintain, and predict internal representations which, like the representations used in the human brain, are object-oriented. In order to build systems that are both highly flexible and can be trained from data, we operate in the framework of deep learning, the dominant paradigm in machine learning for the past decade. Deep neural networks that maintain an internal representation of the world are typically called *world models* [64]. We focus on world models whose internal representations are constrained to be object-oriented (i.e. have separate dimensions corresponding to different objects in the world), a class of model that we call *Object-Oriented World Models* (OOWMs). In particular, our goal in this thesis is to greatly expand the range of environments in which OOWMs can be usefully applied. We pay special attention to two aspects of performance that have largely been neglected by past OOWMs: their ability to deal with scenes containing large numbers of densely packed and overlapping objects, and their ability to properly model objects as existing in 3D space. In this chapter we expand on several of the topics that we have touched on so far, fully

fleshing out the motivation behind building OOWMs and going into greater depth on the ways in which existing OOWMs can be improved.

1.1 Deep Learning, Representations and Models

The past decade has witnessed a giant leap forward in humankind's ability to train many-layered, heavily-parameterized differentiable function approximators, commonly known as deep neural networks [55]. As a result of this progress, we have seen a proliferation of artificial systems that can outperform their human counterparts at a wide-range of tasks that were previously considered well beyond machine capabilities. And while the deep learning revolution has touched a huge variety of domains, nowhere has it been felt more acutely than in the field of computer vision. Differentiable function approximators constitute the state-of-the-art on essentially every visual task, including image classification [155], object detection [151], semantic segmentation [152], monocular depth estimation [104], optical flow estimation [168], and visual question answering [106], to name just a few[1].

Two of the central goals of deep learning, especially as it applies to computer vision, are *representation learning* [11], the pursuit of an efficient means of encoding the current state of the world, and *model building* [64], the project of learning to update representations in the face of new perceptual data and to predict how the world will unfold in the future. These goals can be unified in the deep learning framework of *world models* [64].

A world model is a trainable system which maintains an internal representation of the current state of the world, and includes components for constructing an internal representation and updating it based on new perceptual data, predicting how the world state will evolve in the future given the current state, and reconstructing perceptual data corresponding to a given world state. These components are implemented as deep neural networks so that the model as a whole can be trained end-to-end, using stochastic gradient

[1]State-of-the-art according to `paperswithcode.com/sota`, extracted on July 30, 2020.

descent to maximize the model's ability to reconstruct perceptual data based on its internal representation. Through this training, it is expected that the network learns a compact, efficient representation of the state of the world, and the various model components become effective at their assigned tasks.

1.2 Inductive Biases in Computer Vision

World models as described so far are very general, and can be used to model any kind of sequential data. However, that generality can come at the cost of poor sample efficiency. In machine learning, we can often improve sample efficiency by making use of *inductive biases*. An inductive bias is a way of exploiting any prior knowledge we may have about a learning task, and amounts to influencing the learning algorithm's search process so that it is more likely to select hypotheses that are favored by our prior knowledge.

In the case of computer vision, we have access to a great deal of prior knowledge about how to go about interpreting the visual world. For instance, we have the benefit of years of vision science investigating the computations used by the eye and brain to make sense of visual input [123], and a comparable amount of work hand-designing artificial computer vision systems [150]. Together these two sources of prior knowledge, possibly combined with insights gleaned through introspecting into our own visual processes (though it is well known that introspection is not always reliable[2]), should provide us with ample material for identifying inductive biases that are useful for training world models on visual data.

Convolutional neural networks, the driving force behind much of the success of deep neural networks in computer vision [100], are one example of a highly effective inductive bias for vision [103]. Their motivation starts with a pair of insights (i.e. prior knowledge) about the structure of natural images: important visual features tend to be spatially local,

[2]For example, we do not have conscious access to the complex cascade of processes through which our brains perform stereopsis, mining discrepancies between the different images falling on our two eyes in order to estimate the depth of a 3D scene [77].

and can appear anywhere in an image. Convolutional networks exploit these insights by using spatially local neural receptive fields and replicating their processing spatially across the image. This translates into a massively reduced parameter space (compared to, for example, a multi-layer perceptron with the same number of layers and neurons per layer) without sacrificing expressiveness in any important way, which in turn greatly reduces sample complexity and speeds up learning.

1.3 An Inductive Bias for Objects

The convolutional inductive bias operates at the relatively low level of visual features in images. Can we come up with a visual inductive bias that operates at a higher level of abstraction? We can start by trying to identify prior knowledge about the physical world that might prove useful. One fact that is intimately familiar to all humans is that the world is largely composed of discrete objects moving through 3D space. Indeed, physical objects are so important that humans and many animals seem to be hard-wired to form internal representations that are about particular objects in the world, and use those representations extensively in downstream tasks [19]. Infants as young as 2 months old (and possibly even younger) already interpret the world in terms of objects, evinced by their ability to complete objects that are significantly occluded by a barrier (amodal completion) [87, 88] and their expectation that objects will continue to exist even after they have moved out of sight (object permanence) [2, 3]; there is even evidence that amodal completion is available from birth [157]. Meanwhile, chickens are capable of both amodal completion and object permanence from birth [133], a fact which provides proof that innate object-oriented systems are at least possible in principle.

Given that evolution has apparently gone out of its way to equip humans and other animals with machinery for discovering, tracking, and reasoning about objects, we can then ask what it is about objects that makes them worth representing explicitly. One attractive feature of objects is that they constitute a particularly *modular* way of partitioning

the physical world [61], which means that they are highly structured, largely independent of the scenes they occur in, and tend to recur often in different contexts. The high degree of within-object structure ensures that objects are stable over time, making it worthwhile to form explicit representations of them. The relative independence of objects from the scenes in which they occur (for example, we can often move or change any object in a scene without affecting the rest of the scene) means that object-oriented representations are able to efficiently represent the probable changes to a scene. Finally, any particular type of object will tend to recur in many different contexts, a property which ensures that any effort spent learning about one kind of object (e.g. how it behaves) is likely to pay off as similar objects are encountered repeatedly in the future.

A human's internal representation of an object is called an *object file* [89], i.e. a mental file or data structure storing information about an object out in the world. Object files support a wide range of useful abilities such as maintaining representations of objects even once they have passed out of view, and re-identifying objects when they become visible again. An object file is usually understood as comprising two components [58]:

1. A *visual index* which picks out the object's spatial location, updated as the object moves around the scene.

2. Some amount of *content*, storing perceived information about the real world object picked out by the object file.

The content can include a wide range of information such as shape, color, size, mass, or even abstract properties such as kind (e.g. dog, person) or identity (e.g. President Obama).

We can impose an inductive bias for objects onto world models by making them use a style of representation similar to object files. Whereas vanilla world models use an unstructured representation of the environment, such as a flat vector z, world models equipped with an inductive bias for objects have their representation split into separate "files" or "slots", $\mathcal{Z} = \{z_k\}_{k=1}^{K}$, where $K \in \mathbb{N}$ is the maximum number of objects. Each slot z_k stores the location and known properties of an object in the world (some models

explicitly separate location from content within a slot, while others allow content and location to be tangled together). In this thesis, we call world models equipped with this kind of internal representation *Object-Oriented World Models* (OOWMs). One key advantage of OOWMs is that they enable the use of neural components that are designed to take advantage of the object-oriented structure of the latent representation, often resulting in improvements in sample complexity, interpretability, and generalization. Moreover, when an OOWM model is trained on images or video from a physical environment, our hope is that the object-oriented structure of the internal representation forces the network to learn how to decompose scenes into physical objects. If training is successful, the neural components that extract and update the object-oriented representation will essentially have become a competent object detector and tracker.

Note that the OOWM framework is defined at an abstract level, and the only property of objects that it relies on is their being modular components of the physical world. This allows us to avoid providing a more refined definition of what constitutes an object, and each instantiation of the OOWM framework is free to use its own object definition and representation, so long as their chosen definition still carves the world into discrete modular components. For example, in later chapters of this thesis in which we present novel instantiations of the OOWM framework, we employ relatively simple object representations. In particular, when modeling 2D images or video (see Chapters 4 and 5), the location and size of each object are represented as a simple bounding box, and all other features of the object (such as its appearance) are encoded together into a single vector. In later work (see Chapter 6) in which we build OOWMs for 3D environments, we add to this bounding box representation a 3D vector indicating the location the object in 3D space. These simple object representations are used in part because they are known to be compatible with deep learning (bounding boxes, for example, feature prominently in deep supervised object detection). However, future work would do well to take up some of the more sophisticated object definitions and/or representations that have been presented in the computer vision literature [32, 97]. For example, future representations should

explicitly represent the material properties of objects, should enforce physical rules such as distinct objects not being permitted to occupy the same regions of space, and should provide support for complex objects with articulating parts. One significant challenge along this path will be incorporating these more sophisticated representations in a way that meets the differentiabilty requirements imposed by the deep learning framework.

1.4 Expanding the Scope of Object-Oriented World Models

The task facing an OOWM is in some ways equivalent to the well-studied tasks of supervised object detection [44] and tracking [119]: the goal is to end up with a robust system capable of building representations of the objects in an image, and updating those representations over time as objects move and/or change appearance. However, unlike those other tasks, an OOWM is generally trained *without* the benefit of object-level annotations. Making progress without access to annotations is difficult because, in addition to the relatively straightforward task of learning how to detect and track objects, the network also has to discover what exactly counts as an object in the environment of interest.

Discovering objects in an unfamiliar environment is challenging. There is much philosophical debate about what even counts as an object [57], and an appropriate definition of an object may depend on the task one is interested in performing [61, 118]. As a reasonable first approximation, we could define a physical object to be a maximally large set of strongly connected parts. Here, a teacup counts as an object because it cannot be broken into parts without a degree of physical violence, but no subset of the teacup counts (because it would not be maximally large) nor does the conjunction of the teacup and its saucer (because the parts would not be strongly connected). However, discovering objects is difficult even with this relatively crisp definition, since, in an unfamiliar environment, it is not generally possible to tell which parts are strongly connected to one another based purely on images or (non-interactive) videos. While motion can be a strong cue about

objectness (human infants make heavy use of motion cues for segmenting objects [145]), relying solely on motion means we will miss objects that do not move.

Thus, while the study of OOWMs is progressing quickly, these conceptual issues remain problematic, and much work remains before OOWMs can be used to discover high-quality object-oriented representations in arbitrary environments. Indeed, when the research in this thesis was first undertaken, most OOWMs depended on additional, often quite restrictive, assumptions about the environment. In particular, many OOWMs restricted themselves to small 2-dimensional scenes with few objects and monochrome backgrounds colored so as to contrast starkly with the objects. For example, in Attend, Infer, Repeat (AIR) [42], a seminal paper that introduced one of the first OOWMs for discovering objects in static images, the main dataset consisted of images containing at most 3 white hand-written digits rendered on a black background.

The work in this thesis is primarily aimed at eliminating some of these limitations, with the end goal of significantly expanding the set of environments in which OOWMs can be usefully applied. We pay particular attention to two directions of improvement:

1. Handling larger scenes with many objects and a high degree of inter-object occlusion.

2. Moving from modeling 2D objects on simple backgrounds to 3D objects in complex environments.

To improve in the first direction, we use AIR as a starting point, and import insights from supervised object detection to greatly improve its scalability, resulting in an OOWM that is able to learn high-quality object-oriented representations in domains with large numbers of heavily overlapping objects. We then we take those same insights and apply them to a successor of AIR that operates on videos, Sequential Attend, Infer, Repeat [98], resulting in a full-fledged OOWM that is able to discover, track and forecast objects in videos containing large numbers of moving objects that often heavily occlude one another. To address the second direction of improvement, we formulate an OOWM that explicitly models the positions of objects in 3D space with respect to a known camera pose. We

discover objects by fitting a static 3D model to a scene, and consider anything that cannot be captured by the model to be an object. This allows us to discover objects in significantly more complex environments than past approaches.

1.5 Thesis Structure

This thesis is structured as follows. In Chapter 2 we present technical background, introducing a number of concepts necessary for building OOWMs; these concepts include deep learning with stochastic gradient descent, various kinds of deep neural networks, an important class of deep latent variable model called a Variational Autoencoder, as well as some high-level background on supervised object detection. In Chapter 3 we motivate and concretely define both standard world models and Object-Oriented World Models. We also review the literature on existing OOWMs (limiting ourselves to work that precedes the work presented in this thesis) and discuss in greater detail the two OOWM-related limitations introduced in the previous section (scaling to large numbers of objects and properly handling objects in 3D).

In Chapter 4 we begin to address the question of scalability, using techniques from supervised object detection to build an OOWM for static images that can handle scenes containing many objects and can effortlessly generalize to scenes containing more objects than were present in training scenes. In Chapter 5 we re-use those same insights from supervised object detection, this time using them to construct a full-fledged OOWM for video capable of simultaneously tracking dozens of objects in densely packed scenes. In Chapter 6 our focus switches from scalability to the question of properly modeling 3D scenes; we introduce an OOWM which, unlike previous OOWMs, properly models objects as existing in a 3D environment. In Chapter 7, we conclude with a discussion of what this thesis has achieved, and identify a number of important research questions left open for future work.

Chapter 2

Technical Background

In this section we discuss a number of machine learning concepts that will serve as the building blocks for the original research contributions presented later in the thesis.

2.1 Deep Learning

The models proposed in this thesis are trained using the framework of *deep learning*. In the standard formulation of deep learning, one has a differentiable function f_θ, parameterized by a vector θ, and the aim is to find a value of θ which approximately minimizes some differentiable loss function L. Typically f_θ is a function of an input data point x, which may be paired with a label y that can be used by L to assess the quality of $f_\theta(x)$. Concretely, assuming pairs (x, y) drawn from some distribution $P(x, y)$, we have:

$$\theta^* = \arg\min_\theta E_{(x,y)\sim P(x,y)}[L(f_\theta(x), y)] \, .$$

In most cases of practical interest one does not have access to the true data distribution $P(x, y)$, and instead must make do with a finite dataset D of samples drawn from $P(x, y)$:

$$\theta^* \approx \arg\min_\theta E_{(x,y)\sim D}[L(f_\theta(x), y)] \, .$$

Since L and f_θ are differentiable, we may perform this minimization using gradient descent; that is, by repeatedly computing the gradient of the objective, and moving θ some distance in the opposite direction of that gradient (opposite since we are *minimizing* the loss).

Since D is finite, we could compute the exact gradient by summing over all examples in D. However, this would be prohibitively expensive for even modestly large datasets. Instead, the approach taken in *stochastic gradient descent* (SGD) is to estimate the required gradient using a relatively small batch of samples drawn randomly from D:

$$\begin{aligned}\nabla_\theta E_{(x,y)\sim D}\left[L(f_\theta(x), y)\right] &= E_{(x,y)\sim D}\left[\nabla_\theta L(f_\theta(x), y)\right] \\ &\approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta L(f_\theta(x_i), y_i) \qquad\qquad (x_i, y_i) \sim D\,,\end{aligned}$$

where $N \in \mathbb{N}$ is the batch size. The SGD algorithm is presented in Algorithm 1.

Algorithm 1 Stochastic Gradient Descent

function SGD(Dataset: D, Batch size: N, Function: f_θ, Loss: L, Learning rates: $\{\alpha_t\}_{t=0}^{\infty}$)
    Initialize θ randomly
    $t \leftarrow 0$
    while not converged **do**
        Sample batch $\mathcal{B} = \{(x_i, y_i)\}_{i=1}^{N}$, $(x_i, y_i) \sim D$
        $\Delta\theta \leftarrow \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta L(f_\theta(x_i), y_i)$
        $\theta \leftarrow \theta - \alpha_t \Delta\theta$
        $t \leftarrow t + 1$
    return θ

SGD requires the specification of a sequence of learning rates $\{\alpha_t\}_{t=0}^{\infty}$ controlling the size of successive gradient update steps, and has been shown to converge provided that this sequence satisfies the stochastic approximation conditions [135]:

$$\alpha_t > 0\,, \qquad \sum_{t=0}^{\infty} \alpha_t = \infty\,, \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty\,.$$

A number of variants of SGD have been presented in recent years which aim to speed up the optimization and/or improve the final result. One such variant is momentum, in which θ is moved in the direction of a moving average of recent gradients [148]; this can

help to dampen oscillations which can be harmful to the optimization process. Another variant is RMSProp, which updates θ using an adaptive per-parameter learning rate [154]; this can help the optimization quickly escape from "plateau" regions in which gradient is very small. A third variant, ADAM [94], elegantly combines momentum and RMSProp, and is used to train most of the models presented in this thesis.

2.2 Neural Components

In this section we review a number of different kinds of differentiable, parameterized functions, all of which will be used as components when building some of the larger models proposed later in this thesis.

Before we dive in, a brief note on syntax. Neural networks make extensive use of multidimensional arrays (e.g. for collections of neural activations and learnable parameters), and we will often find it useful to be able to refer to slices of these arrays. We will use a `numpy`-like syntax for this purpose[1]; however, following mathematical convention, we assume array indexing starts from 1 rather than 0, and that array slicing is inclusive on both ends (as opposed to standard numpy, wherein the slice is inclusive at the start but exclusive at the end).

2.2.1 Multi-layer Perceptrons

Multi-layer perceptrons (MLPs) [137] are one of the most basic and widely used classes of learnable functions. An MLP is composed of a series of layers, each layer taking in a vector and yielding another vector as output. These layers are indexed and arranged in a feedforward fashion, so that the output of layer $n-1$ is fed as input to layer n.

Let o_n denote the dimensionality of the vector yielded by layer n, with o_0 denoting the dimensionality of the input vector. The function computed by layer n is composed of a learnable affine transformation, parameterized by a weight matrix $W_n \in \mathbb{R}^{o_n \times o_{n-1}}$ and a

[1] `https://numpy.org/doc/stable/reference/arrays.indexing.html`

bias vector $b_n \in \mathbb{R}^{o_n}$, followed by a fixed nonlinear function, or *nonlinearity*, g_n, applied elementwise. Concretely, the function computed by an MLP with N layers, given input vector x, is:

$$\begin{aligned} \ell_0 &= x \ , \\ \ell_n &= g_n(W_n \ell_{n-1} + b_n), \ n \in \{1, \ldots, N\} \ . \end{aligned}$$

The total set of parameters for the MLP is $\theta = \{W_1, b_1, \ldots, W_N, b_N\}$.

As for the particular choice of nonlinearity, $g_n(\cdot)$, there are a wide range of possibilities in common use, and more are invented every day. Standard choices include $\mathrm{sigmoid}(x) = (1 + e^{-x})^{-1}$, $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$, and rectified linear $\mathrm{ReLU}(x) = \max(x, 0)$. Throughout this thesis ReLU is used as a default, as it has been shown to have a number of desirable properties such as encouraging sparse activation vectors [53] and avoiding saturation [100], and has been adopted by the deep learning community as standard. One additional consideration is that it is generally desirable to allow the MLP to yield any real vector as output; meanwhile, most nonlinearities have restricted ranges (e.g. ReLU cannot yield negative values). For this reason, it is common to require that the nonlinearity for the final layer be the identity function, and we follow that convention throughout this work.

The computations performed in the layers of an MLP were originally inspired by the computations performed by neurons in the brain [137]. To see this, note that we can write entry i of ℓ_n as:

$$\ell_n[i] = g_n \left(\sum_{j=1}^{o_{n-1}} W_n[i, j] \cdot \ell_{n-1}[j] + b_n[i] \right) \ .$$

This computation roughly models the computation performed by a neuron. Each weighted term in this expression is analogous to a synaptic input from an upstream neuron, where the weight corresponds to the synaptic strength. The summation of the weighted terms reflects the pooling of these input signals by the neuron's dendrites and soma. Finally,

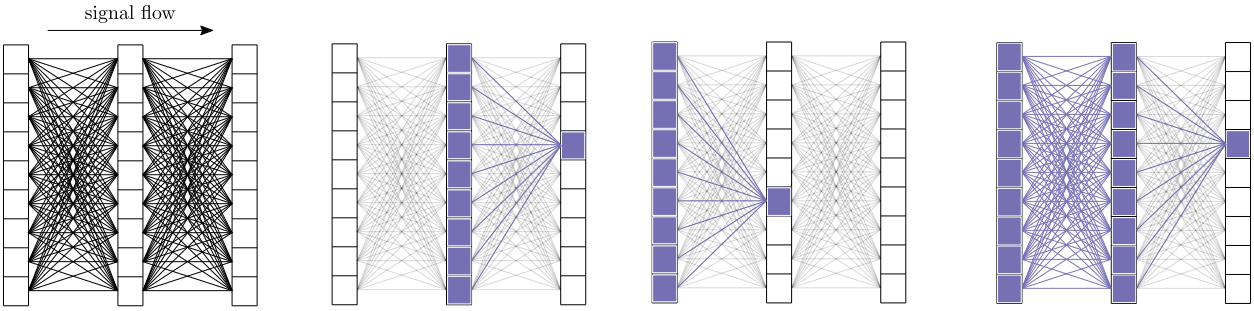

Figure 2.1. Schematic showing the connectivity structure and receptive fields of a Multi-layer Perceptron. In each diagram, 2 MLP layers are shown; each cell corresponds to a neuron, and lines between neurons in different layers indicate that the downstream neuron is a direct function of the upstream neuron. Left: Showing the fully-connected nature of MLPs. Every neuron in a given layer receives input from every neuron in the preceding layer. Center left/right: For neurons in two different layers, showing their receptive fields in the immediately preceding layer. Right: For a neuron in the most downstream layer, showing its receptive field in both of the preceding layers. For any given neuron, every neuron in every upstream layer is in its receptive field.

the nonlinearity corresponds to the fact that if the somatic input is large enough, an action potential or spike is generated in the soma and propagates down the axon, after which it will go on to enervate downstream neurons. In early Perceptron models, the nonlinearity was a threshold function [137], simulating the all-or-nothing character of a neural spike. Most modern MLPs use differentiable nonlinearities for compatibility with backpropagation, and these can be viewed as giving the *firing rate* of the neuron. Due to the neural analogy, it is standard to say that an MLP layer n with output size o_n is composed of o_n "neurons" or "units", and to say that the layer output ℓ_n is a vector of neural activation values, or simply "activations".

One useful concept for understanding neural architectures is that of a neural receptive field [113]. The receptive field of a target neuron is the set of upstream neurons whose activation values can, according to the connectivity structure of the network, affect the target neuron's activation value. MLPs are said to be *fully-connected*, meaning that in each pair of consecutive layers, every neuron in the downstream layer receives input from every neuron in the upstream layer. One consequence is that for any target neuron, its receptive field consists of every neuron in every layer upstream of it. Figure 2.1 depicts the connectivity structure and receptive fields of MLPs.

An MLP is a general-purpose learnable function, and constitutes a reasonable default choice when the function to be learned has no known structure. When there *is* known structure, other network architectures may be more appropriate, as we will see in the next section.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of neural network with connectivity structure optimized for visual tasks [103]; that is, tasks in which the network takes an image or video as input. An excellent introduction to CNNs may be found in [105].

Convolutional Volumes

As with MLPs, CNNs are generally feedforward, with each layer producing an array of activations by applying a set of learned weights to the array of activations produced by the previous layer. Unlike MLPs, wherein the activations were organized into vectors (1-dimensional arrays), CNNs organize their neural activations into 3-dimensional arrays called *convolutional volumes*. For a general CNN, let V_n be the volume of activations produced by the n-th convolutional layer, and let V_n have shape (H_n, W_n, C_n) where H_n is the volume height, W_n is the volume width and C_n is the number of channels. The initial volume is the input image, so $(H_0, W_0, C_0) = (H_{img}, W_{img}, 3)$, assuming images are encoded in RGB format (3 values per pixel).

The first two dimensions of V_n, the height and width dimensions, are spatial. Let (h, w) be a pair of indices along the height and width dimensions, $h \in \{1, \ldots, H_n\}, w \in \{1, \ldots W_n\}$; we will call such a pair a *spatial location*. The vector at spatial location (h, w), namely $V_n[h, w, :]$, stores information about a specific region of the input image. The reason for this spatial correspondence will become clear after we define the convolution operation in the next section. The third dimension of V_n, often called the *channel* dimension, is non-spatial; moving along this dimension for a fixed (h, w) merely accesses a different aspect of the information stored about the region covered by spatial location (h, w).

The set of all activations in a volume with spatial location (h, w) is given by $V[h, w, :]$, and is called a *feature column*. Meanwhile, the set of all activations with channel index c is given by $V[:, :, c]$, and is called a *feature map*.

Convolutional Layers

In the most elementary case, the n-th convolutional layer is parameterized by a set of C_n-many filters, or kernels, organized into a 4-dimensional array $F_n \in \mathbb{R}^{K_n \times K_n \times C_{n-1} \times C_n}$. Integer $K_n \geq 1$ is the filter size and specifies the spatial extent of the filter. A learnable bias $b_n \in \mathbb{R}^{C_n}$ may also be included. Then for a unit in the n-th volume with spatial location (h, w) and channel index c, we compute its activation as:

$$V_n[h, w, c] = g_n \left(\sum_{i=1}^{K_n} \sum_{j=1}^{K_n} \sum_{k=1}^{C_{n-1}} F_n[i, j, k, c] \cdot V_{n-1}[h + i, w + j, k] + b_n[c] \right) . \tag{2.1}$$

Here g_n is a nonlinearity which plays the same role as in MLPs.

In words, for the c-th filter $F_n[:, :, :, c]$, we begin by spatially aligning the filter with the top-left of the input volume, elementwise-multiply the filter with the aligned entries from V_{n-1}, sum the result, and finally pass it through the nonlinearity. We then spatially shift the filter to the right, and repeat. When we get to the end of the volume horizontally, we go back to the left side of the filter and move down one unit. Performing this operation for all locations that the filter can occupy yields the activations for the c-th feature map of the output volume, $V_n[:, :, c]$. This operation (minus the nonlinearity) is essentially a 2-dimensional *convolution* between the input volume and (a flipped version of) the filter, and it is this concept which gives convolutional networks their name. Performing the convolution for all C_n-many filters, and stacking the results along the channel dimension, yields the full output volume V_n. In the most basic scenario, we apply the filter at every location at which it "fits", and the spatial shape of the output volume will thus be $(H_n, W_n) = (H_{n-1} - K_n + 1, W_{n-1} - K_n + 1)$.

Convolutional layers have two additional structural parameters: stride and padding. The stride for layer n, s_n, is a positive integer. When performing the convolution, we skip every $(s_n - 1)$-many locations where the filter could be applied. Because of these skips, the spatial size of the output volume is reduced by a factor of s_n. The padding for layer n, p_n, is a non-negative integer. Before performing the convolution we concatenate p_n-many feature columns containing all zeros to the input volume. We do this along both spatial dimensions, both "before" and "after" the input volume. Padding has the effect of creating additional locations at which the filter may be applied, which increases the spatial shape of the output volume. Note that stride and padding can use different values for each spatial dimension, and for padding we can also pad different numbers of zeros "before" and "after" the input volume, though we have omitted full treatments of these cases for brevity. Taking the stride and padding into account, we have:

$$V_n[h, w, c] = g_n\left(\sum_{i=1}^{K_n}\sum_{j=1}^{K_n}\sum_{k=1}^{C_{n-1}} F_n[i, j, k, c] \cdot V_{n-1}[s_n \cdot h - p_n + i, s_n \cdot w - p_n + j, k] + b_n[c]\right) .$$

Padding is implemented here by stipulating that when accessing V_{n-1} using an index that is out-of-bounds, a value of zero is returned. The spatial size of the output volume with stride and padding is:

$$(H_n, W_n) = \left(\left\lfloor \frac{H_{n-1} + 2p_n - K_n}{s_n} \right\rfloor + 1, \left\lfloor \frac{W_{n-1} + 2p_n - K_n}{s_n} \right\rfloor + 1\right) . \tag{2.2}$$

The Benefits of the Convolutional Architecture for Processing Images

Here we discuss a number of key ways in which the convolutional architecture just described lends itself well to processing images. In particular, convolutional neural networks have *local receptive fields* and employ a large degree of *weight sharing* [55, 103]. Importantly, they achieve these properties in a way that is computationally efficient. Also,

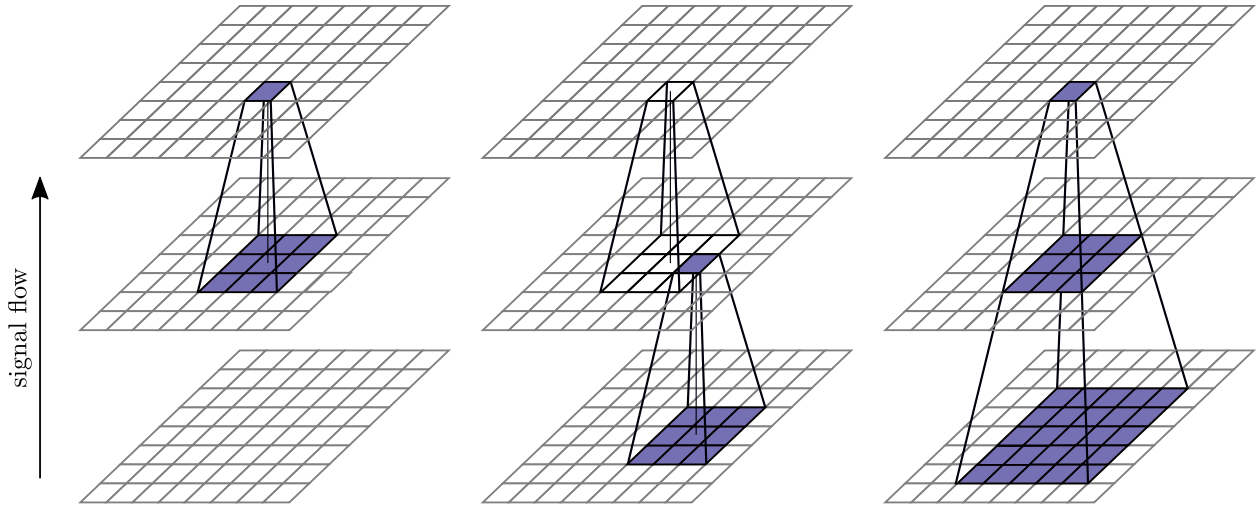

Figure 2.2. Schematic showing the locality of receptive fields in Convolutional Neural Networks. In each diagram, 2 convolutional layers are shown, and each cell corresponds to a spatial location in a convolutional volume. The channel dimensions of the convolutional volumes are not shown here. For both layers, $K_n = 3$, $s_n = 1$ and $p_n = 1$. The padding is not shown here, but $p_n = 1$ is required so that all layers have the same spatial shapes. Left/Center: Receptive fields of two different spatial locations in their immediately preceding layers. Right: For a target spatial location in the most downstream layer, showing its receptive fields in both upstream layers. In all cases, the receptive fields are rectangular regions.

together these properties have the added bonus of allowing the network to operate on images of different sizes.

Local Receptive Fields. In convolutional neural networks, the receptive fields are significantly more sparse and structured than in MLPs, and in particular are *spatially local*. Consider the neurons at spatial location (h, w) in the n-th volume, whose activations are $V_n[h, w, :]$. In general, the receptive field of these neurons in an earlier layer with index m, where $0 \leq m < n$, will be a rectangular region; an example is shown in Figure 2.2. In the simplest case, where $m = n - 1$, we can immediately see from Equation (2.1) that only spatial locations in a K_n-by-K_n square in the previous layer V_{n-1} are able to affect $V_n[h, w, :]$, and hence it is only these spatial locations that are in the receptive field of spatial location (h, w). It is often most useful to consider the receptive field of (h, w) in the input image, $m = 0$, which allows us to view the activations $V_n[h, w, :]$ as "paying attention to", or encoding information about, a rectangular subregion of the input image.

These structured, sparse receptive fields are useful for image processing because in images, spatially local correlations are generally much more interesting than spatially distant ones; the features of interest in images, such as edges, keypoints, and even objects, are mostly spatially local. By restricting the neurons' attention to subregions of the input image, we simplify their task significantly, effectively forcing them to ignore correlations between spatially distant pixels.

Weight Sharing. The other property of the convolutional architecture that makes it well-suited to image processing is weight sharing. We have established that in a given layer, neurons at different spatial locations encode information about different rectangular subregions of the input image. In principle, we could process these different subregions heterogeneously; for example, we could use one set of weights to process a subregion in the top-left corner of the image, and another set of weights to process a subregion in the bottom-right corner. Convolutional neural networks take a different approach, called weight sharing, in which each subregion of the image is processed in exactly the same way, using exactly the same weights. This weight sharing is implemented by the convolution operation, which applies the filters (weights) repeatedly at each possible spatial location.

Weight sharing is appropriate for images because any visual feature could appear at any location in the input image, and so it makes sense to process all spatial subregions in the same way. Moreover, weight sharing significantly reduces the number of parameters in the network, which means training the network will be more sample efficient. One way to think of this is as a sort of multi-task learning, wherein each subregion of the input image is a different task. Without weight sharing, one would be forced to use a separate set of weights for each task, and learn each task separately. In contrast, weight sharing is effectively saying that all of these tasks are in fact identical, and hence we can pool data from all tasks and use a single set of weights for all tasks.

Computational Efficiency. Convolutional networks are a highly computationally efficient way of implementing the combination of spatially local receptive fields and weight sharing. Consider, for instance, one alternative way of achieving the same properties. We could

first divide the input image up into a number of overlapping rectangular subregions (the same rectangular subregions that a CNN would impose with its receptive fields), and process the image by applying a single MLP separately to each region. This has similar properties to a CNN, but would be significantly slower. Since the subregions will overlap, we will end up processing information from any given pixel multiple times, once for each subregion that contains the pixel. In contrast, in CNNs each spatial location in the image is processed locally by the filters, and that processed information is reused in multiple downstream locations, and this kind of reuse happens in every layer.

Image size flexibility. One additional benefit that comes from the combination of local receptive fields and weight sharing is that a convolutional neural network can be trained on images of different sizes, and after training can be used to process images larger than any seen during training. To see why this is the case, note that in performing the convolution operation, we are simply sliding the filters spatially over the input volume, and in principle there is no limit to how far we can slide. If we need to process a new image that is larger than any we have seen before, we will simply have more locations at which to apply the filters, and the spatial size of the output volume will be correspondingly larger. This is reflected by the fact that in Equation (2.2), the spatial size of the output volume of any layer is a simple monotonically increasing function of the spatial size of the input volume. Contrast this with an MLP, wherein the number of units in each layer is fixed, and it is unclear how a trained network would be extended to handle a larger image.

To see how this flexibility can be useful, consider the example of Fully Convolutional Networks (FCN) [112]. This work showed that by exploiting image size flexibility, a convolutional neural network that had been originally trained for the task of image classification could be straightforwardly adapted to perform dense semantic segmentation of spatially larger images. In effect, this amounts to applying the image classification network to different subregions of the larger images that were to be spatially segmented, obtaining a classification output for each subregion. Each classification label could then be

reinterpreted as a segmentation label for the pixel at the center of each of the corresponding subregions.

2.2.3 Spatial Transformers

Spatial Transformers (ST) [83] constitute another important class of differentiable function, providing a parameterized, differentiable spatial transformation of an image or convolutional volume. Importantly, they allow the parameters of the transformation to be conditioned on the data itself, which can be used to implement interesting functionality such as attention. A schematic showing different uses of Spatial Transformers is shown in Figure 2.3.

Suppose we are given an input volume V_{in} with shape $(H_{\text{in}}, W_{\text{in}}, C_{\text{in}})$, and that we want to obtain a spatially transformed version of that volume, denoted as V_{out}, with a new shape $(H_{\text{out}}, W_{\text{out}}, C_{\text{in}})$ (note that the channel dimension C_{in} is unchanged). We begin by laying a 2D coordinate system over the spatial dimensions of the input volume, where coordinate $(-1, -1)$ gives the top left corner of the volume and $(1, 1)$ gives the bottom right corner. We define a function $p(n, N) = \frac{2}{N}(n - 0.5) - 1$ which gives the location of the center of the n-th cell when the interval $(-1, 1)$ is divided up into N cells of equal size. We can use this function to get the coordinates for a given spatial location in our new coordinate system; the coordinates of spatial location (h, w) in V_{in} are $(p(h, H_{\text{in}}), p(w, W_{\text{in}}))$.

We now use this coordinate system to define a differentiable spatial sampling operation over the input volume. Given some sampling point $(y, x) \in [-1, 1]^2$, define the sampling operation as:

$$S((y, x), V_{\text{in}}) = \sum_{h=1}^{H} \sum_{w=1}^{W} V_{\text{in}}[h, w, :] \cdot k(y - p(h, H_{\text{in}})) \cdot k(x - p(w, W_{\text{in}})) \,, \tag{2.3}$$

where k is a sampling kernel. Any differentiable kernel can be used, however it is common to use a bilinear kernel, $k(d) = \max(0, 1 - |d|)$; this choice effectively turns Equation (2.3) into a weighted average of the four feature columns in V_{in} whose coordinates are closest

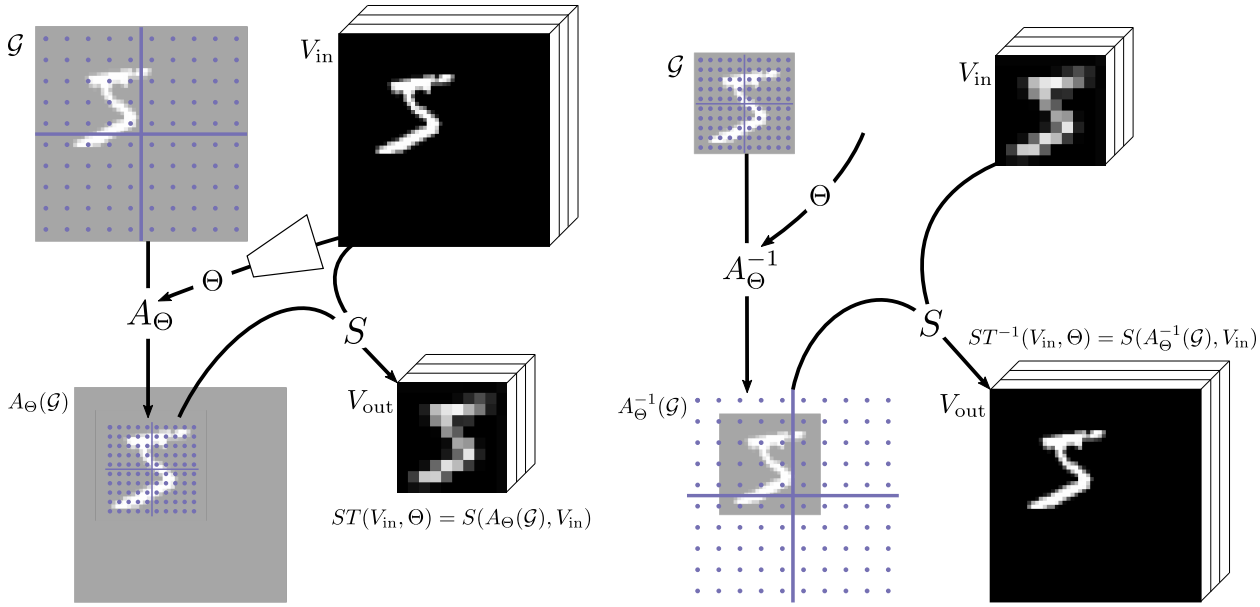

Figure 2.3. Schematic depicting different uses of Spatial Transformers. Each blue dot is a sample point. Left: Using spatial transformers to implement a form of attention. The white trapezoid is a neural network that has learned to inspect the input image V_{in} and yield transformation parameters Θ which center the output image on the object of interest, in this case an MNIST digit. Right: Using inverse spatial transformers for placing a small image (e.g. an object appearance) inside of a larger image, with a position and size specified by transformation parameters Θ. In this case Θ will typically come from some external source rather than being conditioned on the input image.

to (y, x) (since the product of the kernels is 0 for all other terms). Importantly, this in turn allows an efficient implementation, since we can avoid summing over all spatial locations in the input volume. Note that it is possible for the sample point (y, x) to fall outside of the input image. To handle this, we define a default feature column V_{default} and pretend that V_{in} is surrounded on all sides by copies of it; the effect is that sampling with an out-of-bounds point returns V_{default}.

Next, for each spatial location (i, j) in the *output* volume we define a canonical sampling location $g_{ij} = (p(i, H_{\text{out}}), p(j, W_{\text{out}}))$. We group these together in a set to form a canonical grid of sampling location $\mathcal{G} = \{g_{ij} | (i, j) \in \{1, \ldots, H_{\text{out}}\} \times \{1, \ldots, W_{\text{out}}\}\}$. We then define $S(\mathcal{G}, V_{\text{in}})$ as applying the sampling operation in Equation (2.3) separately at each point in $\mathcal{G}$, and arranging the result into an output volume so that $S(\mathcal{G}, V_{\text{in}})[i, j, :] = S(g_{ij}, V_{\text{in}})$.

Note that $S(\mathcal{G}, V_{\text{in}})$ effectively constitutes an image resizing operation if the input and output spatial shapes differ, and otherwise implements the identity function. However, we

can obtain a more general class of operations by introducing a parameterized transformation of the sampling points, A_Θ:$[0,1]^2 \rightarrow [0,1]^2$, and stipulating that $A_\Theta(\mathcal{G}) = \{A_\Theta(g)\}_{g\in\mathcal{G}}$, so that $A_\theta(\mathcal{G})$ is a transformed version of the canonical grid. We then apply this transformation to the sampling grid before performing the sampling procedure S, allowing the sampling operation as a whole to be controlled by varying Θ. It is common to restrict the class of transformations, for example to the set of affine transformations:

$$A_\Theta((y,x)) = \begin{bmatrix} \Theta_h & 0 & \Theta_y \\ 0 & \Theta_w & \Theta_x \end{bmatrix} \begin{bmatrix} y \\ x \\ 1 \end{bmatrix} .$$

The sampled region will have center (Θ_y, Θ_x), height $2\Theta_h$ and width $2\Theta_w$.

Putting all this together, a Spatial Transformer is defined as:

$$ST(V_{\text{in}}, \Theta) = S(A_\Theta(\mathcal{G}), V_{\text{in}}) .$$

Importantly, the output is differentiable with respect to the input volume, and with respect to the transformation parameters Θ as long as the kernel k is differentiable. Throughout this work, we use Spatial Transformers for several different purposes, in all cases using a bilinear kernel and restricting the class of transformations to affine transformations.

Spatial Transformers for Visual Attention

One typical use of spatial transformers is to have a neural network process the input image and yield parameters Θ for the sampling grid transformation [69, 83]; because Spatial Transformers are differentiable, this neural network can be trained by backpropagating through the output volume. This kind of processing can be used to support a form of spatial attention, shown in Figure 2.3, left. The network that predicts Θ is expected to learn to select regions from the input image that contain information relevant to the task of interest. For example, in the case of object classification, it could learn to locate the object to

be classified within the image. Ideally, the trained Θ-network will pick an input-dependent value of Θ which results in a spatial transformation that crops out all but a bounding box centered on the object of interest, thereby significantly simplifying the job of downstream layers. This cropped version of the input image is sometimes called a "glimpse" [120].

Inverse Spatial Transformers for Object Placement

Another common use of spatial transformers is differentiably placing a small image inside of a larger image at a desired position. An example is shown in Figure 2.3, right. To achieve this, we simply interpret the transformation parameters Θ in the opposite way as before, using an inverse transformation of the grid:

$$A_\Theta^{-1}((y,x)) = \begin{bmatrix} 1/\Theta_h & 0 & -\Theta_y/\Theta_h \\ 0 & 1/\Theta_w & -\Theta_x/\Theta_w \end{bmatrix} \begin{bmatrix} y \\ x \\ 1 \end{bmatrix} .$$

This will place the center of the small image at location (Θ_y, Θ_x), and will give it height $2\Theta_h$ and width $2\Theta_w$. We will write inverse spatial transformers as:

$$ST^{-1}(V_{\text{in}}, \Theta) = S(A_\Theta^{-1}(\mathcal{G}), V_{\text{in}}) .$$

In later chapters, we use this technique to implement a differentiable, object-based renderer. In particular, we apply inverse spatial transformers to multiple objects independently, and then combine the resulting images into a single output image through (weighted) summation.

2.3 Variational Autoencoders

In this section we introduce the Variational Autoencoder (VAE) framework [95], which provides the theoretical foundation for our approach to object discovery. VAEs are a class

of latent variable model which lend themselves well to optimization via stochastic gradient descent. In this thesis our main focus is on using VAEs to model images, though they can be used for any kind of data.

2.3.1 Overview

Suppose that we have some data, and we are interested in discovering latent factors underlying that data. For the specific case of images, the latent factors could be the physical objects in the world that give rise to the image. Let x and z denote a data point and its latent factors, respectively. To model this notion of latent factors, we assume the data are sampled from a *generative model*. The latent representation z is first sampled from a prior distribution $P(z)$, and the data point x is then produced according to a generative distribution $P(x|z)$:

$$P(x) = \int P(x|z)P(z)dz \ .$$

The generative distribution is unknown to us, so we approximate it using a function $P_\theta(x|z)$ parameterized by a vector θ. The probability of a data point under our approximate generative distribution is:

$$P_\theta(x) = \int P_\theta(x|z)P(z)dz \ .$$

We are interested in modeling noisy, high-dimensional data such as images, so we consider the case where z is a high-dimensional vector and $P(z|x)$ is an arbitrarily complex conditional distribution. Under these assumptions, it is necessary to model $P_\theta(x|z)$ as a flexible function approximator such as a neural network.

Now suppose we have access to a finite dataset assumed to be sampled from this generative model, $D = \{x_i\}_{i=1}^N, x_i \sim P(x)$. One task of immediate interest is finding

generative parameters θ^* which maximize the probability of the dataset:

$$\begin{aligned}\theta^* &= \arg\max_\theta P_\theta(D) \\ &= \arg\max_\theta \prod_{i=1}^{N} P_\theta(x_i) \ .\end{aligned}$$

In order to avoid working with a many-factor product of probabilities (some of which may be very small) we take the logarithm of our objective function. This is justified (yields the same value of θ^*) since the logarithm is a monotonically increasing function. This gives us:

$$\begin{aligned}\theta^* &= \arg\max_\theta \ \log\left(\prod_{i=1}^{N} P_\theta(x_i)\right) \\ &= \arg\max_\theta \sum_{i=1}^{N} \log P_\theta(x_i) \\ &= \arg\max_\theta \sum_{i=1}^{N} \log \int P_\theta(x_i|z_i)P(z_i)dz_i \ .\end{aligned}$$

Here we would appear to be at an impasse; due to our assumption that z is high-dimensional, the expectation inside the logarithm is intractable and attempts to approximate it (or its gradients) via sampling will yield high-variance estimates.

Variational methods proceed by introducing a conditional distribution $Q(z|x)$, called a variational distribution, mapping a given input data point to a distribution over latent variables. $Q(z|x)$ will turn out to be an approximation of the posterior distribution $P_\theta(z|x) = P_\theta(x|z)P(z)/P_\theta(x)$, a connection which will be made clear further below.

Using $Q(z|x)$, we can start to rewrite $\log P_\theta(x)$ in a more useful form:

$$
\begin{aligned}
&\log P_\theta(x) \\
&= \int Q(z|x) \log P_\theta(x) dz \\
&= \int Q(z|x) \log P_\theta(x) \frac{P_\theta(z|x)}{P_\theta(z|x)} dz \\
&= \int Q(z|x) \log \frac{P_\theta(x|z)P(z)}{P_\theta(z|x)} dz \\
&= \int Q(z|x) \log \frac{P_\theta(x|z)P(z)}{P_\theta(z|x)} \frac{Q(z|x)}{Q(z|x)} dz \\
&= \int Q(z|x) \log P_\theta(x|z) dz + \int Q(z|x) \log \frac{P(z)}{Q(z|x)} dz + \int Q(z|x) \log \frac{Q(z|x)}{P_\theta(z|x)} dz \\
&= \int Q(z|x) \log P_\theta(x|z) dz - D_{KL}(Q(z|x) \parallel P(z)) + D_{KL}(Q(z|x) \parallel P_\theta(z|x)) \,. \qquad (2.4)
\end{aligned}
$$

Here $D_{KL}(\cdot \parallel \cdot)$ is the Kullback-Leibler (KL) Divergence, which is defined as $D_{KL}(a(z) \parallel b(z)) = -\int a(z) \log \left(b(z)/a(z)\right) dz$ for any two distributions $a(z)$ and $b(z)$.

We cannot compute the rightmost term in Equation (2.4) directly as it involves the intractable posterior distribution $P_\theta(z|x)$; however, we at least know that it must be ≥ 0 since KL divergences are always non-negative. Thus we have:

$$
\log P_\theta(x) \geq \int Q(z|x) \log P_\theta(x|z) dz - D_{KL}(Q(z|x) \parallel P(z)) \,. \qquad (2.5)
$$

The right-hand side of Relation (2.5) is called the *evidence lower bound* (ELBO), and we will denote it as $\mathcal{L}(x, \theta, Q)$. Variational methods in general work by maximizing $\sum_{i=1}^{N} \mathcal{L}(x_i, \theta, Q)$ with respect to both θ and Q, as a surrogate for maximizing $\sum_{i=1}^{N} \log P_\theta(x_i)$ directly.

Variational Autoencoders in particular use a differentiable function $Q_\phi(z|x)$ parameterized by a vector ϕ (i.e. a neural network). We can then rewrite the ELBO as a function of ϕ rather than Q:

$$
\mathcal{L}(x, \theta, \phi) := \int Q_\phi(z|x) \log P_\theta(x|z) dz - D_{KL}(Q_\phi(z|x) \parallel P(z)) \,. \qquad (2.6)
$$

Overall our optimization problem is thus:

$$(\theta^*, \phi^*) \approx \arg\max_{(\theta,\phi)} \sum_{i=1}^{N} \mathcal{L}(x, \theta, \phi)$$
$$= \arg\max_{(\theta,\phi)} \sum_{i=1}^{N} \int Q_\phi(z_i|x_i) \log P_\theta(x_i|z_i) dz_i - D_{KL}(Q_\phi(z_i|x_i) \parallel P(z_i)) \,. \quad (2.7)$$

We can perform this optimization in terms of θ and ϕ simultaneously using SGD or any of its variants; however, computing the required gradients requires certain specialized techniques, outlined below in Sections 2.3.3 and 2.3.4.

We can derive additional insight into the role of $Q_\phi(z|x)$ as follows. First note that Equation (2.4) combined with the definition of $\mathcal{L}(x, \theta, \phi)$ gives us:

$$\log P_\theta(x) - \mathcal{L}(x, \theta, \phi) = D_{KL}(Q_\phi(z|x) \parallel P_\theta(z|x)) \,.$$

But also note that $\log P_\theta(x)$ does not change as ϕ changes. Therefore, when we maximize $\mathcal{L}(x, \theta, \phi)$ with respect to ϕ, this is the same as *minimizing* $D_{KL}(Q_\phi(z|x) \parallel P_\theta(z|x))$ with respect to ϕ. In other words, $Q_\phi(z|x)$ is trained to be an approximation of the intractable posterior $P_\theta(z|x)$, where the quality of the approximation is measured by the KL Divergence. $Q_\phi(z|x)$ is often called an approximate posterior or *inference* network, reflecting the fact that its role is to infer a distribution over the values of z which gave rise to an observed x.

Finally, we provide some brief intuition as to the role of the two terms in the ELBO defined in Equation (2.6). The first term can be viewed as forcing $Q_\phi(z|x)$ to extract information from x, storing it in z, and forcing $P_\theta(x|z)$ to use the information stored in z to build an accurate reconstruction of x; for this reason, Q_ϕ is sometimes called an *encoder* and P_ϕ a *decoder*. Meanwhile the KL Divergence term forces $Q_\phi(z|x)$ to only generate values z which are assigned high probability by the prior $P(z)$. This can be interpreted as limiting the amount of information that Q_ϕ encodes in z, acting as a kind of regularizer which forces Q_ϕ to learn an efficient code. After optimization has converged, we expect Q_ϕ to

have learned to use z as an efficient, compressed representation of the input x, and P_θ to be useful for generating data; in particular, the process of sampling $z \sim P(z)$ and then $x \sim P_\theta(x|z)$ should yield samples similar to the training data.

2.3.2 Parameterizing Distributions with Neural Networks

Until now we have been talking about $P_\theta(x|z)$ and $Q_\phi(z|x)$ as both neural networks and as density functions of conditional probability distributions; here we unpack exactly what that means. For simplicity we will discuss $Q_\phi(z|x)$, but the same applies to $P_\theta(x|z)$.

To say that the conditional distribution $Q_\phi(z|x)$ is parameterized by a neural network is to say that it has a particular form. In the most basic case, it is broken up into two parts, one stochastic and independent of neural network parameters ϕ, the other deterministic and dependent on ϕ. The stochastic portion is assumed to be some simple parameterized distribution (e.g. Normal) with probability density $\xi(z;\eta)$ where η are the distribution parameters; ξ should be differentiable with respect to both z and η. The deterministic portion is a neural network Q'_ϕ (e.g. an MLP) which maps x to a value for η, $\eta = Q'_\phi(x)$. Combining the two parts we have $Q_\phi(z|x) = \xi(z; Q'_\phi(x))$.

We can also allow parameterized autoregressive conditioning between the dimensions of z. Partition z into K disjoint sets of latent variables $z_1, \ldots, z_K$, where $z = \{z_k\}_{k=1}^K$. We introduce K conditional distributions $Q_{k,\phi_k}(z_k|z_1, \ldots, z_{k-1}, x) = \xi_k(z_k; Q'_{k,\phi_k}(z_1, \ldots, z_{k-1}, x))$ where $\phi_k, Q_{k,\phi_k}, Q'_{k,\phi_k}, \xi_k$ are each analogous to their counterparts from the simple unconditional case. This gives us:

$$Q_\phi(z|x) = \prod_{k=1}^{K} Q_{k,\phi_k}(z_k|z_1, \ldots, z_{k-1}, x) = \prod_{k=1}^{K} \xi_k(z_k; Q'_{k,\phi_k}(z_1, \ldots, z_{k-1}, x)) \,.$$

The full set of parameters for Q_ϕ is then $\phi = \{\phi_k\}_{k=1}^K$.

2.3.3 The Reparameterization Trick

As mentioned previously, VAEs are trained by performing the optimization in Equation (2.7) using SGD. To achieve that, we need to be able to compute the gradient of the ELBO (Equation (2.6)) with respect to encoder parameters ϕ and decoder parameters θ:

$$\nabla_{\theta,\phi}\mathcal{L}(x,\theta,\phi) = \nabla_{\theta,\phi}\left[\int Q_\phi(z|x)\log P_\theta(x|z)dz - D_{KL}(Q_\phi(z|x) \parallel P(z))\right] .$$

This computation is not completely straightforward. The main issue is that the first term of the ELBO will not, in general, be expressible in closed form. It involves an expectation over $Q_\phi(z|x)$, which suggests that a sampling-based approach may be appropriate; however, this is complicated by the fact that we are optimizing with respect to the parameters ϕ of the would-be sampling distribution. This can be overcome using score function estimators, however these typical yield high-variance gradient estimates [140].

Fortunately, a lower-variance gradient estimate can be obtained using a technique called the reparameterization trick [95]. This works by analyzing $Q_\phi(z|x)$ and identifying a noise distribution $\rho(\epsilon)$, and a transformation $\mathcal{T}(\epsilon,\phi,x)$ (required to be differentiable with respect to ϕ) such that $z = \mathcal{T}(\epsilon,\phi,x)$. In other words, we find a way of sampling from $Q_\phi(z|x)$ that operates by first sampling from a fixed noise distribution, and then applying a transformation controlled by ϕ. For example, consider the case where z is a 1-dimensional latent variable, and the $\xi(\eta)$ for $Q_\phi(z|x)$ (see Section 2.3.2) is a Normal distribution with parameters $\eta = (\mu,\sigma)$. Then we can pick $\epsilon \sim \mathcal{N}(0,1)$ and $\mathcal{T}(\epsilon,\phi,x) = \mu_\phi(x) + \sigma_\phi(x)\epsilon$, where $\mu_\phi(x), \sigma_\phi(x) = Q'_\phi(x)$.

We will use this new form to rewrite the first term in the ELBO, performing a change-of-variables that replaces z with ϵ:

$$\int Q_\phi(z|x)\log P_\theta(x|z)dz = \int \rho(\epsilon)\log P_\theta(x|\mathcal{T}(\epsilon,\phi,x))d\epsilon .$$

We are no longer optimizing with respect to the sampling distribution, so we can straightforwardly estimate gradients by sampling:

$$\begin{aligned}\nabla_{\theta,\phi} \int Q_\phi(z|x) \log P_\theta(x|z) dz &= \nabla_{\theta,\phi} \int \rho(\epsilon) \log P_\theta(x|\mathcal{T}(\epsilon, x, \phi)) d\epsilon \\ &= \int \rho(\epsilon) \nabla_{\theta,\phi} \log P_\theta(x|\mathcal{T}(\epsilon, x, \phi)) d\epsilon \\ &\approx \frac{1}{M} \sum_{j=1}^{M} \nabla_{\theta,\phi} \log P_\theta(x|\mathcal{T}(\epsilon_j, x, \phi)) \qquad \epsilon_j \sim \rho(\epsilon) \; .\end{aligned}$$

We will often estimate this using a single sample from $\rho(\epsilon)$, in which case we have:

$$\nabla_{\theta,\phi} \int Q_\phi(z|x) \log P_\theta(x|z) dz \approx \nabla_{\theta,\phi} \log P_\theta(x|\mathcal{T}(\hat{\epsilon}, x, \phi)) \qquad \hat{\epsilon} \sim \rho(\epsilon) \; .$$

As for differentiating the second term of the ELBO, in many cases this KL Divergence can be computed in closed form, so sampling can be avoided. Failing that, the reparameterization trick applies there as well.

2.3.4 Reparameterizing Discrete Latent Variables

In many cases we will need to use latent variables with discrete values. However, the reparameterization trick cannot be applied in a straightforward manner to such variables, as gradients cannot propagate through discrete functions. Fortunately, we can get around this to an extent by replacing any discrete latent variables with Concrete random variables [84, 114], which are continuous, differentiable relaxations of certain kinds of discrete variables.

Consider a Categorical random variable over n classes. We represent draws from this distribution as one-hot binary vectors $z = [z_1, \ldots, z_n], z_i \in \{0, 1\}$, with $\sum_{i=1}^{n} z_i = 1$, and parameterize the distribution using a vector of unnormalized probabilities $\alpha = [\alpha_1, \ldots, \alpha_n], \alpha_i \in (0, \infty)$. The following procedure can be used to sample from Categorical(α):

1. Sample $u_i \sim$ Uniform$(0, 1)$ for $i = 1, \ldots, n$.
2. Compute $g_i = \log \alpha_i - \log(-\log u_i))$ for $i = 1, \ldots, n$.
3. Yield $z_i = \mathbb{I}(i = \arg\max_j g_j)$ for $i = 1, \ldots, n$.

where $\mathbb{I}(\cdot)$ is the indicator function. This sampling strategy conforms to the noise-then-transform format required by the reparameterization trick, but it cannot be used directly because the argmax and indicator functions do not have useful gradients with respect to distribution parameters α. To get around this, one performs a *relaxation*, replacing the argmax/indicator combination with a softmax. Random variables sampled in this manner are called Concrete random variables, and their support is the simplex Δ^{n-1}. Given a temperature hyperparameter $\lambda > 0$, we replace step 3 of the sampling algorithm with:

$$z_i = \frac{e^{g_i/\lambda}}{\sum_{j=1}^{n} e^{g_j/\lambda}} \,. \tag{2.8}$$

Under this strategy, z is differentiable with respect to α, and thus the reparameterization trick can be used.

The parameter λ in Equation (2.8) controls the sparsity of the sampled vectors; decreasing λ places more weight on points near the corners of the simplex (the corners being the one-hot vectors), resulting in "sparser" sampled vectors. Indeed, the corresponding Categorical (whose sampled one-hot vectors can be thought of as maximally sparse) is a limiting case of the Concrete as $\lambda \to 0$. The tactic of replacing Categorical latent variables with Concrete relaxations does come with a slight cost: the gradient estimates obtained by applying the reparameterization trick to the Concrete will be biased with respect to gradients of the original computation graph. Ultimately λ controls a trade-off

between smoothness of the optimization and bias of the gradients, with larger values of λ corresponding to smooth-but-biased.

We can perform a similar relaxation for Bernoulli random variables. If we parameterize the Bernoulli using α, the *odds* of the positive result (i.e. $\alpha = p/(1-p)$, where p is the probability of the positive result), then one way to sample such a variable is:

1. Sample $u \sim \text{Uniform}(0, 1)$.
2. Compute $g = \log \alpha + \log u - \log(1-u)$.[2]
3. Yield $z = \mathbb{I}(g \geq 0)$.

The relaxation is obtained by replacing the hard, non-differentiable thresholding operation in step 3 with a soft, differentiable sigmoid function:

$$z = \text{sigmoid}(g/\lambda) = \frac{1}{1 + \exp^{-g/\lambda}} \,,$$

where λ is once again a temperature/sparsity parameter. The resulting random variable is called a BinConcrete (for "Binary Concrete").

2.4 Supervised Object Detection with Deep Convolutional Neural Networks

Supervised object detection is a subfield of computer vision in which the goal is to build models capable of taking in an input image and returning a description of the objects therein. Deep neural networks are currently the state of the art for object detection [151], and this success is highly relevant for the project pursued in this thesis, because one crucial requirement of an Object-Oriented World Model (OOWM) is a means of differentiably mapping from an input image to a set of predicted object representations (this corresponds to estimating and/or updating the OOWM's object-oriented internal state based on an

[2]Note that $\log \alpha + \log u - \log(1-u)$ is a Logistic random variable with location parameter $\log \alpha$

input frame). Here we present an overview of a few of the most significant architectures and concepts in deep supervised object detection, several of which feature prominently in later chapters when designing novel OOWMs. Note that in supervised object detection, objects are represented as bounding boxes, which is a large part of the reason that the OOWMs presented in later chapters use bounding boxes as their core object representation.

2.4.1 Problem Definition

In supervised object detection, we want to build systems that can take an image as input and output a set of predicted objects for the image. In this setting, ground-truth objects have two components: a bounding box specifying the spatial extent of the object in the image, and a categorical variable representing the class of the object. The bounding box is represented as a 4-tuple (b^y, b^x, b^h, b^w) where the first two elements give the location of the center of the bounding box, and the last two give the height and width of the box. The object class is a C-dimensional one-hot vector, $c \in \{0,1\}^C, \sum_{i=1}^{C} c[i] = 1$, assuming C possible classes.

Predicted objects are represented slightly differently, with a bounding box $(\hat{b}^y, \hat{b}^x, \hat{b}^h, \hat{b}^w)$, and a length-$(C+1)$ probability vector $\hat{c} \in \mathbb{R}^{C+1}, \hat{c}[i] \geq 0, \sum_{i=0}^{C} \hat{c}[i] = 1$, where class 0 is a background or null class. We define the object's *presence* as $\hat{c}^{\text{pres}} = \sum_{i=1}^{C} \hat{c}[i] = 1 - \hat{c}[0]$ (sometimes also called the confidence or objectness), which can be thought of as the network's confidence in the existence of the object. The use of a null class, and the consequent definition of object presence, helps solve one of the primary issues in using neural networks for object detection, namely dealing with the fact that different images have different numbers of objects. This is solved by giving networks a large, fixed number of output object slots, and then predicting high presence values for only a subset of slots.

The word "supervised" in supervised object detection means that at training time we have access to a large dataset consisting of images paired with ground-truth object annotations. Concretely, the training set is given by $D_{\text{train}} = \{(x_n, \ell_n)\}_{n=1}^{N_{\text{train}}}$, where x_n is an image and ℓ_n is the list of annotations for objects in the image, $\ell_n = \{((b^y_{n,k}, b^x_{n,k}, b^h_{n,k}, b^w_{n,k}), c_{n,k})\}_{k=1}^{K_n}$,

letting K_n denote the number of ground-truth objects in image x_n. Validation and test datasets, D_{val} and D_{test}, are defined similarly.

2.4.2 Measuring Performance: Average Precision

The primary measure of performance used in supervised object detection is a family of metrics collectively called Average Precision (AP) [44]. We call it a *family* of metrics because it takes parameters, and different datasets, competitions and papers make use of different values of these parameters (though some widely used standards have been established).

First we define a measure of the quality of match between a ground-truth bounding box b and a predicted bounding box $\hat{b}$ called the Intersection-over-Union (IOU), also known as the Jaccard similarity or Jaccard index [82]. The IOU is given by the area of the intersection of the two boxes divided by the area of their union:

$$IOU(b, \hat{b}) = \frac{\text{Area}(b \cap \hat{b})}{\text{Area}(b \cup \hat{b})} .$$

Note that $0 \leq IOU(b, \hat{b}) \leq 1$.

In defining AP, we first restrict ourselves to objects of a single class i, meaning that we only consider ground-truth objects whose class is i. We also introduce an IOU threshold τ_{IOU} and a probability threshold τ. Now, considering only predicted objects whose probability for class i, $\hat{c}[i]$, is greater than τ, we pair up ground-truth objects with predicted objects using a matching process. For ground-truth bounding box b and predicted bounding box $\hat{b}$ to be paired up, we require that $IOU(b, \hat{b}) > \tau_{IOU}$.

With pairs in hand, the strategy in AP is to view the problem of detecting objects of class i as a binary classification problem. In the nomenclature of binary classification, we view each pair as a true positive, each unpaired ground-truth object as a false negative, and each unpaired predicted object as a false positive; there is no analog of a true negative. We can then compute standard measures derived from binary classification, namely

precision $= TP/(TP + FP)$ and recall $= TP/(TP + FN)$, where TP, FP and FN are the number of true positives, false positives and false negatives, respectively.

Note that the precision and recall are both functions of the probability threshold τ; specifically, τ controls a trade-off between precision and recall, where increasing τ generally decreases recall while (ideally, though not always) increasing precision. When using a trained classifier in practice, we would choose a value for τ that strikes an appropriate balance between precision and recall for the application of interest (estimated using a validation dataset). However, in evaluating object detectors without a particular application in mind, we would prefer a measure that takes into account performance at all thresholds. To obtain such a measure, we first define the precision-recall curve, which gives precision as a function of recall. Then our metric is simply the area under the precision-recall curve, abbreviated as AUC (Area Under the Curve). In this thesis we approximate AUC by averaging precision over a fixed set of target recall values [44]; this approximation was used in early papers on object detection, though it is also possible to obtain exact values [43].

The final step in computing AP is simply to take an average of the per-class scores. Using a simple average over classes means that AP assigns equal importance to all classes regardless of their frequency in the test set. This is important, as certain classes are often significantly over-represented in the data. For example, in one standard dataset, Microsoft Common Objects in Context (MS-COCO) [107], nearly 1/3 of ground-truth objects in the union of the train and validation sets have the class "person"; thus if AP combined per-class performance by, for instance, weighting by the class frequency, then the metric would largely reflect models' ability to detect humans.

As a final note, it is also common to average over a range of values for τ_{IOU}, which allows taking into account different requirements for the accuracy of bounding box predictions; a higher value of τ_{IOU} translates into a stronger penalty for inaccurate bounding boxes. Again, different downstream applications will place different requirements on the

quality of the bounding boxes, so it is reasonable to average over several values when evaluating without a particular application in mind.

2.4.3 Proposal–based Object Detection

The first wave of neural network–based object detection solutions made use of a relatively complex, 2-stage process [51, 52, 134]. In the first stage, called the *Region Proposal* stage, a number of class-independent candidate bounding boxes (or regions) are proposed; these proposals are typically chosen based on some measure of how likely they are to contain an object. In the second stage, called the *Region Evaluation* stage, these candidate bounding boxes are assigned confidence scores for each class, indicating how likely it is that the given region corresponds to an object of each class. The bounding boxes may also be refined in this stage. An additional post-processing step called Non-Maximum Suppression may also be applied, designed to reduce the number of duplicate detections.

The first approach in this category was R-CNN (**R**egions with **C**onvolutional **N**eural **N**etwork features) [52]. In R-CNN, the Region Proposal stage is achieved using a non-neural algorithm called Selective Search [156], which yields around 2000 proposals per image. In the Region Evaluation stage, for each of the 2000 proposed bounding boxes, the image pixels inside a dilated version of the box are extracted and warped into a square image. Each of these square images is then independently fed into a CNN which outputs a feature vector describing the region. A set of per-class Support Vector Machines consume these feature vectors and output class scores for the proposals, and a separate linear model yields refined bounding boxes.

R-CNN achieves good object detection performance but is quite slow at test time, requiring as much as 47 seconds per image. One reason for this long processing time is that in the Region Evaluation stage, the CNN has to be run independently on each of the 2000 sub-images derived from the proposals. One of R-CNN's successors, Fast R-CNN [51], improves this step by making the observation that since many of the 2000 proposed regions will overlap significantly, parts of the image are being wastefully re-processed

many times over (each pixel is processed once for each region that contains it). In Fast R-CNN, the Region Evaluation stage starts by first running the entire input image through a CNN, yielding an output volume V. Next we extract feature vectors for each of the proposed regions by max pooling features from the volume V over the region. These feature vectors are then mapped to class predictions and bounding box refinements via an MLP; this MLP replaces the classification SVM and linear box refiner from R-CNN, and allows end-to-end training. In terms of speed, the main benefit of this approach is that the relatively expensive CNN has to be run only once per image, rather than once per proposed region.

After Fast R-CNN improved the Region Evaluation stage, the Region Proposal stage became the major bottleneck. To fix this, a model called Faster R-CNN [134] invented a way to speed up the Region Proposal stage by replacing the Selective Search algorithm with a CNN, called a Region Proposal Network (RPN), trained to propose bounding boxes that were likely to contain objects. Crucially, the RPN was made to share many of its layers with the feature extraction CNN from the Region Evaluation stage. This layer sharing improves execution speed, and may also result in a form of multi-task learning [20], wherein the parameters of the shared layers are regularized by having to learn features that are useful for both the proposal and evaluation tasks.

2.4.4 Single Shot Object Detection

An alternate class of object detection architectures, called single-shot detectors, strive for speed and simplicity compared to the relatively slow and complex multi-stage proposal–based detectors [111, 130, 131, 132]. Rather than proposing a set of regions in one step and then classifying those regions in a second step, single-shot detectors map directly from the input image to a set of complete objects (that is, including both bounding box and class predictions), all in one stage and using a single CNN. The CNN at the core of a single shot detector has a structure very similar to the RPN from Faster R-CNN. The primary difference is that the RPN outputs only bounding boxes (and an objectness score), leaving

the classification of those boxes for the Region Evaluation stage, while the CNN in a single shot detector outputs bounding boxes and classification scores all at once. By combining Region Proposal and Region Evaluation stages, single-shot detectors are able to achieve significantly faster inference speeds while retaining object detection performance that is competitive with the multi-stage detectors.

The prototypical single shot detector is YOLO (You Only Look Once) [130]. YOLO uses a CNN backbone, and an object prediction method similar to the RPN from Faster R-CNN, but adds a classification head alongside the bounding box prediction head. YOLO is able to achieve real-time prediction speeds (defined as 30Hz), though its object detection accuracy (measured by AP) was somewhat worse than Faster R-CNN. A subsequent version of YOLO, YOLOv2 [131], added a number of features including a fully convolutional architecture (YOLO used an architecture that was largely convolutional, but included two fully-connected layers near the end), batch normalization, anchor boxes, anchor box clustering, direct location prediction, multi-scale training, and a custom network architecture. Together these improvements yield a network that is both faster and more accurate than Faster R-CNN. Finally, YOLOv3 [132] adds predictions at multiple-scales, multi-class rather than single class prediction, and a deeper convolutional backbone.

Another single shot detector, appropriately named the Single Shot Detector (SSD) [111], uses many of the same concepts as YOLO. However, it pioneered one important feature, namely making object predictions at multiple layers of the convolutional network, each designed to handle objects of different scales. This approach was integrated into YOLOv3.

2.4.5 Architectural Concepts for Object Detection

Here we examine in greater detail several design strategies used by many of the above architectures, many of which will show up again in later chapters.

Receptive Fields, Grid Cells and Predicting Bounding Box Location

Supervised object detection architectures make heavy use of CNNs, allowing them to benefit from attractive CNN properties like weight sharing, local receptive fields, and image size flexibility (discussed previously in Section 2.2.2). Given that we are using a convolutional architecture, an important question is how to map from the CNN output volume to a set of predicted objects. One simple approach would be to first reduce the volume to a fixed length vector (either by flattening it, or by global spatial pooling); said vector could then be mapped to a sequence of objects by a recurrent neural network. This strategy is similar to standard architectures used for image classification, wherein the final convolutional volume is flattened and mapped to classification logits by way of a number of fully-connected layers. However, a significant downside of this approach is that the flattening (or global pooling) operation discards spatial structure that is present in the output volume; in particular, it discards the fact that different spatial locations in the volume store information about specific regions of the input image. Losing this structure is acceptable for image classification, as there it is necessary to integrate information about the entire image in order to make the classification decision. However, when detecting objects in an image, which requires making relatively fine-grained predictions about bounding box sizes and locations, it ends up being highly beneficial to make explicit use of the extra spatial information implicitly stored in the convolutional volume.

In order to exploit the spatial information contained in the convolutional network's output volume, modern object detection architectures employ the strategy of predicting separate objects from each spatial location in the volume. Moreover, they do this in a way that ensures the objects predicted from a given spatial location are (at least partly) inside the spatial location's receptive field. This ensures that each spatial location only predicts objects that it could possibly know about (a given spatial location only has access to information about objects that are inside its receptive field), and generally helps constrain the set of objects that must be predicted at each spatial location, thereby simplifying the learning task.

With appropriately chosen image padding, a given convolutional network can be interpreted as dividing its input image up into a grid of cells, where each cell is at the center of the receptive field of a different spatial location in the output volume. Then when predicting bounding boxes from a specific spatial location, we do so using a transformation which ensures that the center of the predicted bounding boxes are inside the grid cell assigned to that spatial location. We can view an object detection architecture with this kind of design as taking a single object detector that operates on image subregions, and applying it repeatedly at many different subregions of the input image (i.e. the receptive fields of neurons in the output layer). Importantly, the task of predicting objects in a subregion is relatively simple compared to the task of predicting objects in the whole image.

This kind of approach is used in Faster R-CNN's RPN [134], in SSD [111], in YOLOv2 [131] and YOLOv3 [132]. YOLOv1 [130] used two fully connected layers just before the output, resulting in a network that was not fully convolutional and in which the receptive fields of all locations in the output volume were in fact the entire image.

Anchor Boxes

Another feature common to many modern object detection architectures is the use of *anchor boxes* [134], also known as default boxes [111], which are used to improve an architecture's ability to predict object bounding boxes across a wide range of scales. An anchor box is simply a bounding box with a fixed size that is used as a reference; when predicting the size of an object, the prediction is made as a displacement from this reference size. It is common to choose a set of anchor boxes that roughly span the set of box shapes in the training set. For example, YOLOv2 chooses anchor box shapes by running the k-means clustering algorithm on the bounding boxes in the training set [131].

Typically, architectures will make it so that each spatial location has multiple object detection "heads" so that multiple objects can be predicted at each spatial location, and will assign each head a different anchor box. In effect, the different heads become separate

object detectors that each specialize in predicting objects of different shapes. During training, ground-truth objects are assigned to the head whose anchor box most closely matches the ground-truth object's size, which further enforces this specialization. One side benefit of this multi-head approach is that it allows the network to predict multiple bounding boxes per cell, which may be necessary when objects are densely packed and overlapping.

Making Predictions at Multiple Scales

Another strategy for dealing with objects of widely varying scales is to make object predictions at several different layers of the CNN (SSD [111], YOLOv3 [132]). As a rule of thumb, it is easiest to predict objects that are moderately smaller than the size of the receptive field of the neurons in the layer at which objects are being predicted. When objects are much smaller than the receptive field, they may get lost amid a sea of other visual features, and the feature vectors may not contain sufficiently fine-grained information to predict the small bounding boxes accurately. When objects are bigger than the receptive field, the detector simply does not have access to information about the full spatial extent of the object, and thus has no way of making an accurate prediction.

Recall from Section 2.2.2 that as we proceed deeper in the CNN, the neurons' receptive fields (in the input image) become progressively larger. Consequently, it should be significantly easier to predict small objects at earlier layers, and larger objects at later layers. Architectures capitalize on this insight by predicting objects at multiple layers of the CNN, and equipping each layer with anchor boxes appropriate for the receptive fields of neurons in that layer.

Non-Maximum Suppression

When predicting objects at test time, it is common for individual objects to be detected by multiple detection heads. For example, for large objects, there may be multiple grid cells that are near the center of the object, and it may happen that all those grid cells detect the

object (each with slightly different bounding boxes, since most architectures require that bounding boxes have their center inside the grid cell). The network itself is not capable of preventing this, as the prediction heads at different spatial locations typically do not communicate with one another.

These architectures thus need a way of dealing with duplicate object detections. This is achieved using a simple algorithm called Non-Maximum Suppression (NMS) [45]. In this algorithm, for each class we first sort the predicted objects in descending order of probability score for the class. Next, we iterate through the objects; each iteration, we find all other objects whose degree of overlap with the current object is above a fixed threshold. All such overlapping objects have their probabilities for the class set to 0 and are ejected from the list.

2.5 Conclusion

In this chapter we introduced a number of technical concepts that will serve as the building blocks for the work presented in later chapters. We introduced the framework of deep learning and stochastic gradient descent (SGD), which allows differentiable function approximators to be trained to minimize loss functions. We discussed several classes of differentiable functions trainable via SGD including multi-layer perceptrons (MLPs; a kind of general-purpose neural network), convolutional networks (CNNs; neural networks that are useful for processing images and videos), and spatial transformers (a kind of controllable, differentiable image transformation). Next we introduced the framework of Variational Autoencoders (VAEs), a class of latent variable model that can be efficiently trained using SGD, and can be used to learn latent representations of observed data. Finally, we surveyed the literature on supervised object detection, paying particular attention to how that field builds neural networks which can map from images to sets of detected objects in a scalable way. In the next chapter, we begin to put some of these pieces to

work, describing a VAE-based framework for learning object-oriented representations from images and video.

Chapter 3

Object-Oriented World Models

In this chapter we introduce Object-Oriented World Models (OOWM), which will serve as a unifying framework for the models presented in later chapters. We begin by providing motivation for the project of building models of the world, focusing on how agents can use world models to optimize their performance. Next we show how the task of learning world models can be formulated within the framework of Variational Autoencoders. From there we arrive at the concept of *Object-Oriented* World Models by noting that world models would do well to use internal representations which reflect the way that the world is naturally decomposed into objects. We then provide a few general strategies for building OOWMs, and survey past work in the field. We conclude by identifying a number of open questions which will serve as motivation for the work presented in later chapters.

3.1 Optimizing Agent Performance

In artificial intelligence, we are interested in building agents capable of learning to perform well in arbitrary environments. *Model-based Reinforcement Learning* is one general strategy for achieving that goal, and it can be roughly summarized as follows:

1. Learn to keep track of the state of the world as the agent interacts with it.

2. Learn to predict how the state of the world will evolve in the future conditioned on the current state and a sequence of future actions.

3. When operating in the environment, use the system from step 1 to keep track of the world state. Then at each choice point, choose actions by using the system from step 2 to perform Model-Predictive Control [18]: simulate future action sequences, and select the action that maximizes expected future performance.

Empirical evidence suggests that model-based reinforcement learning algorithms are significantly more sample efficient than their model-free counterparts [65, 66, 67, 90]. Model-based algorithms also have an advantage when it comes to learning to perform new tasks in a fixed environment [17, 149]; a learned model will be useful for all subsequent tasks, whereas model-free methods will typically have to completely retrain their policy to adapt to a new task.

3.2 World Models

The systems for tracking and forecasting the environment state together constitute a *world model* [64]. One way of formulating and training world models is by casting them as temporal Variational Autoencoders (VAEs) [15, 26, 34, 48, 93, 116], which can be seen as non-linear generalizations of classic state-space models like Kalman filters [91] and Hidden Markov Models [129]. While it is possible to define world models outside of the framework of VAEs, VAEs have the advantage of providing native support for latent variables. Latent variables in turn are important for handling the stochasticity and partial observability often encountered in complex environments [65, 66, 90].

In formulating world models as temporal VAEs, we closely follow the Recurrent State-Space Model (RSSM) from [66]. Observations take the form of length-T sequences, $\{x_{(t)}\}_{t=0}^{T-1}$, and we instantiate a separate set of latent variables for each timestep $\{z_{(t)}\}_{t=0}^{T-1}$. We also allow the model as a whole to condition on a sequence of actions $\{u_{(t)}\}_{t=0}^{T-2}$. Notationally, we always surround temporal indices with brackets to differentiate them

from other kinds of indices which will become important later on. We use expressions like $x_{(i:j)}$ to refer to a temporal sequence of variables starting from timestep i and ending at timestep j (inclusive on both ends), so $x_{(i:j)} = \{x_{(t)}\}_{t=i}^{j}$.

The full prior distribution over latent variables is $P(z_{(0:T-1)}|u_{(0:T-2)})$; however we will generally work with a causal decomposition of the prior, $P(z_{(t)}|z_{(0:t-1)}, u_{(0:t-1)})$, where we have made the assumption that the latent state at time t is independent of future actions when conditioning on the past. The causal prior is modeled as a neural network $P_\theta(z_{(t)}|z_{(0:t-1)}, u_{(0:t-1)})$. Following past work [26, 66], we use a recurrent neural network (RNN) to summarize information about past actions and latent variables. Thus to obtain the prior at time t, we first update the recurrent state as:

$$r_{(t)} = f_\theta(r_{(t-1)}, z_{(t-1)}, u_{(t-1)}) \,,$$

where $r_{(t)}$ is the hidden state of the RNN at time t. The prior then takes the form $P_\theta(z_{(t)}|r_{(t)}, z_{(t-1)}, u_{(t-1)})$. The RNN can be thought of as providing a deterministic path between the past and the present; without it, the model would only be able to propagate information over time by way of the latent variables, whose stochasticity would make it difficult to preserve information over long stretches [66].

The full generative model is $P(x_{(0:T-1)}|z_{(0:T-1)}, u_{(0:T-2)})$; however, we will assume that $z_{(t)}$ contains all information required to generate $x_{(t)}$, so that we can decompose the generative model as:

$$P(x_{(0:T-1)}|z_{(0:T-1)}, u_{(0:T-2)}) = \prod_{t=0}^{T-1} P(x_{(t)}|z_{(t)}) \,.$$

The per-timestep generative distribution $P(x_{(t)}|z_{(t)})$ is sometimes called an *observation* distribution. The observation distribution is modeled as a neural network $P_\theta(x_{(t)}|z_{(t)})$.

We now turn our attention to the inference network $Q_\phi(z_{(0:T-1)}|x_{(0:T-1)}, u_{(0:T-1)})$, whose purpose is to yield a distribution over latent variables $z_{(0:T-1)}$ conditioned on observed input and action sequences. Technically, when inferring the latent state $z_{(t)}$ for any particu-

lar timestep t we should condition on *all* observations, including those which occur after time t, as any of them may contain crucial information about $z_{(t)}$[1]. This kind of inference is called *smoothing* [91]. However, smoothing is not appropriate for use in an online setting (such as model-based reinforcement learning) in which we need to make inferences about the latent state without access to future observations. Thus we use an inference network which conditions only on past and current observations and actions (a type of inference called *filtering*), written as $Q_\phi(z_{(t)}|x_{(0:t)}, u_{(0:t-1)})$. In performing inference, we can use the same recurrent network used in the prior to maintain a summary of past latent states and actions, so that the inference network takes the form $Q_\phi(z_{(t)}|x_{(t)}, r_{(t)}, z_{(t-1)}, u_{(t-1)})$.

Thus, the components of a world model are:

$$
\begin{aligned}
&r_{(t)} = f_\theta(r_{(t-1)}, z_{(t-1)}, u_{(t-1)})\,, && \text{(recurrent update)}\\
&P_\theta(z_{(t)}|r_{(t)}, z_{(t-1)}, u_{(t-1)})\,, && \text{(prior)}\\
&P_\theta(x_{(t)}|z_{(t)})\,, && \text{(generation)}\\
&Q_\phi(z_{(t)}|x_{(t)}, r_{(t)}, z_{(t-1)}, u_{(t-1)})\,. && \text{(inference)}
\end{aligned}
$$

The structure of generation and inference for a world model are shown in Figure 3.1.

In terms of the algorithm outlined in the previous section, we can keep track of the state of the world (step 1) using inference (f_θ and Q_ϕ), and we can simulate the effects of future sequences of actions (step 2) using the causal prior (f_θ and P_θ).

A world model can be trained by using gradient ascent to maximize a suitably adapted version of the evidence lower bound (ELBO) (see Equation (2.6) in the previous chapter):

[1] As an example of a case in which it is important to take into account information from future observations, imagine a model in which the latent variables store the state of all objects in the environment, and consider a sequence in which one of the objects is completely occluded until time t'. In order to infer the full latent state at any time $t < t'$, it would be informative to take into account observation $x_{(t')}$ as it provides information about the hidden object.

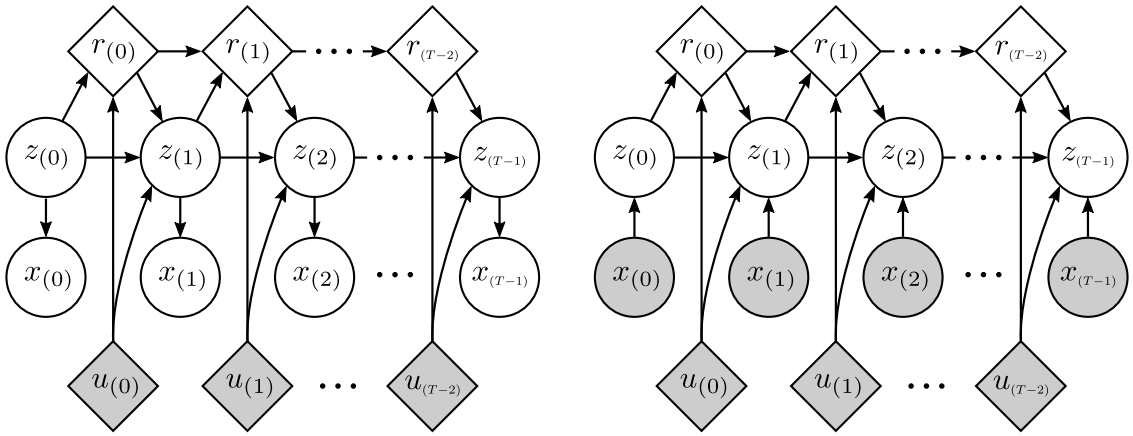

Figure 3.1. Dependency structure of a world model formulated as a temporal VAE. Circles/diamonds are stochastic/deterministic functions of their inputs. Nodes that have observed values are colored grey. Left: Generative model. Right: Filtering-based inference.

$$\begin{aligned}
&\mathcal{L}(x_{(0:T-1)}, u_{(0:T-2)}, \theta, \phi) \\
&= \sum_{t=0}^{T-1} E_{z_{(0:t-1)} \sim Q_\phi} \Bigg[\int Q_\phi(z_{(t)} | x_{(t)}, r_{(t)}, z_{(t-1)}, u_{(t-1)}) \log P(x_{(t)} | z_{(t)}) dz_{(t)} \\
&\qquad + D_{KL}(Q_\phi(z_{(t)} | x_{(t)}, r_{(t)}, z_{(t-1)}, u_{(t-1)}) \parallel P_\theta(z_{(t)} | r_{(t)}, z_{(t-1)}, u_{(t-1)})) \Bigg] .
\end{aligned}$$

Gradients of this expression can be estimated using the reparameterization trick and ancestral sampling from Q_ϕ.

3.3 Object-Oriented World Models

We now turn our attention to the question of building world models for environments which are composed of multiple objects interacting with one another, a class which includes many real-world environments. In such cases, we may hope to build world models whose internal representations are object-oriented, reflecting the natural structure of the environment. Furthermore, we may hope to structure components of the world model (e.g. the causal prior) in ways which exploit properties of those object-oriented representations

in order to realize improved performance along various dimensions. We will broadly refer to such models as Object-Oriented World Models (OOWMs).

In this section we briefly discuss the notion of an object, provide a general characterization of object-oriented representations, and show how such representations can be incorporated into world models. We then discuss some of the benefits that can be expected from the use of object-oriented representations, and finally discuss some general strategies that can be used to learn object-oriented representations which faithfully represent objects in the environment.

3.3.1 Objects

In science and engineering, a powerful metaphor, famously used by Plato (Phaedrus 265e), is that of "carving up the world at its joints": just as a butcher has an easier time carving up an animal carcass by making cuts at the animal's joints, in analyzing any complex system it is often useful to identify and work with natural ways in which the system may be decomposed into parts. Modern engineering, for example, adheres closely to the principle of building complex machines out of modular parts, so that there is a high-degree of within-part functional interdependence but relatively little dependence across parts [142]. Meanwhile, in the philosophy of science it has been suggested that the world is made up of natural kinds (e.g. species of animal), and that the goal of science is to identify those kinds and delineate their boundaries [144].

Any domain can be decomposed in an infinite number of ways, but some decompositions will be more natural than others; it is thus worthwhile to identify criteria for considering a decomposition to be natural. One possible criterion is the degree of *modularity* of the decomposition [61]. A modular decomposition of a domain is one in which:

1. There is a high degree of within-component structure.
2. The components are relatively independent of one another.
3. The components tend to recur and be reusable in different contexts.

One example of a modular decomposition is the decomposition of strings of characters into words in natural language text. Imagine we were presented with an English text from which all spaces and all punctuation had been artificially removed by some adversary. A decomposition in this case would be some algorithm for grouping strings of adjacent characters; each group is one component of the decomposition. Many decompositions are possible; for example, the simplest possible strategy would be to designate each individual character is its own group. Another possible decomposition is one in which each group is a maximally long string of all vowels or all consonants. A third strategy would be one in which the components are English words (i.e. the decomposition imposed by the spaces and punctuation before they were removed by the adversary). This word-based decomposition strategy can be seen to be the most modular of the three possibilities presented. There is a large amount of structure, or predictability, between the characters in the same word, but comparatively little between characters in adjacent words. Moreover, words are relatively independent; for a given sentence we can often swap any word for some other word and still obtain a valid sentence. Finally, individual words are obviously reused often in different contexts (e.g. different sentences). None of these properties are likely to be true of the other decomposition strategies. Words can thus be viewed as a particularly modular, and thus natural, way of decomposing the characters in a sentence into groups.

Roughly speaking, *physical objects* are the components that we obtain when we apply this idea of a modular decomposition to the macro-scale physical world. While not all domains admit a modular decomposition, laws of physics conspire to ensure that the physical world does. Some pairs of atoms have strong physical bonds with one another (e.g. two atoms in a teacup) whereas other pairs have weak bonds (e.g. an atom in a teacup and an atom in the saucer that the teacup is resting on). As a first approximation, we can define physical objects to be maximal clusters of strongly connected atoms.

This definition results in decompositions with a high-degree of modularity. There is a large degree of structure within objects, as it is relatively hard to change the structural

relations between parts that belong to the same object. Objects defined in this way are also relatively independent: because of the weak bonds between atoms in an object and atoms in the rest of the world, objects can often move independently of the scene they are in, and for any scene we can imagine a modified version of it in which some object is moved, turned upside down, swapped out for another object, or removed altogether. Finally, these objects also recur in many different contexts. The stability of the within-object physical bonds ensure that objects tend to persist over time, and thus have the potential to end up in many different scenes. Moreover, due to the laws of physics and other higher-order effects (e.g. evolution, human design), for any given object there tends to be a large number of nearly identical copies thereof (think of all the instances of blades of grass, trees, dogs, people), ensuring that each class of object is likely to recur often and in a variety of different contexts.

Note that the definition of objects given above, based on "clusters of strongly connected atoms", is intentionally loose, and is mainly presented mainly as a concrete (if simplistic) example of a modular decomposition of the physical world. The computer vision literature has no shortage of subtle and sophisticated definitions of objects [32, 97]; however, they are largely beyond the scope of this work. For our purposes, all that is required is that the physical world admits a decomposition into modular components, which we take to be relatively uncontroversial. In the next section, we show how to define models and representations which mirror this modular decomposition of the world, and capitalize on it in various ways.

3.3.2 Object-Oriented Representations and World Models

Evolution has endowed humans with built-in machinery for decomposing visual scenes into the kinds of physical objects described in the previous subsection [19, 145, 146], building what we will call *object-oriented representations*. Such machinery undoubtedly consumes precious metabolic resources, suggesting that these representations must come with significant advantages; we should therefore consider it a priority to equip our artificial

agents with the ability to construct and make use of similar representations. In this section we give a concrete definition of object-oriented representations, show how they can be incorporated into world models, and spell out some of the benefits we expect to gain from using them.

For any dataset, there are an infinite number of different ways to represent it. One of the central goals of deep learning is finding *good* representations of data [55]. An important consideration for any representation is which aspects of the represented data are made explicit; that is, which attributes can be "read off" of the representation with minimal additional computation [118]. It is generally desirable for representations to make explicit the independent factors of variation in a dataset, and such representations are said to be *disentangled* [74]. Disentangled representations have a number of desirable properties; for instance, they ensure that any downstream systems making use of the representations have easy access to the important high-level features of the data.

As an example, consider learning a representation for a dataset of images of human faces; here some of the independent factors of variation would be age, gender, and skin color [74]. Each image constitutes one kind of representation of the face that it depicts; however, it is a highly entangled representation, since one still has to do a great deal of work to compute the values of the important factors of variation. In contrast, a disentangled representation would be one with dimensions corresponding specifically to each of age, gender, and skin color.

In complex physical environments involving multiple objects, the independent factors of variation will be the objects themselves. We thus define an object-oriented representation to be one that is disentangled with respect to the objects in a scene. In particular, we will divide the dimensions of our representation into separate subsets, sometimes called "slots", where each slot represents information about a different object in the scene.

In the context of world models, assuming we want to allow room for $K \in \mathbb{N}$ objects, we can simply replace the flat latent vector $z_{(t)}$ with a set of per-object latent vectors, $\mathcal{Z}_{(t)} = \{z_{(t),k}\}_{k=1}^{K}$. Similarly, we replace the single hidden state $r_{(t)}$ with a set of per-object

hidden states $\mathcal{R}_{(t)} = \{r_{(t),k}\}_{k=1}^{K}$. We then adapt the components of a vanilla world model accordingly to obtain a general template for an Object-Oriented World Model[2]:

$$
\begin{aligned}
&r_{(t),k} = f_\theta(r_{(t-1),k}, z_{(t-1),k}, u_{(t-1)})\,, && k = 1, \ldots, K && \text{(recurrent update)}\\
&P_\theta(z_{(t),k} | \mathcal{R}_{(t)}, z_{(t-1),k}, u_{(t-1)})\,, && k = 1, \ldots, K && \text{(prior)}\\
&P_\theta(x_{(t)} | \mathcal{Z}_{(t)})\,, && && \text{(generation)}\\
&Q_\phi(z_{(t),k} | x_{(t)}, \mathcal{R}_{(t)}, z_{(t-1),k}, u_{(t-1)})\,. && k = 1, \ldots, K && \text{(inference)}
\end{aligned}
$$

We allow the prior and inference networks to condition on the recurrent states for all objects $\mathcal{R}_{(t)}$ in order to account for the effects of interactions between objects. Also, while this template captures many OOWMs, some may introduce additional dependencies; for example, some models impose an ordering on the slots and perform inference sequentially, conditioning their inference for each object on inferences for previous objects from the same timestep [98].

3.3.3 The Advantages of Object-Oriented Representations

Having introduced object-oriented representations and how they can be integrated into world models, we now outline some of the advantages we expect to get in return.

Representing Changes Efficiently

One consequence of the relative independence of objects with respect to the rest of a scene is that the most common changes to a scene will involve changes to one or a small handful of objects, leaving the remainder of the objects constant. For example, during a tea party, most objects will be stationary most of the time; the changes that do happen will often be sparse in terms of the objects affected, as guests pour tea for one another or take sips from their cups at uncorrelated points in time.

[2] Arguably the first OOWM to include all these components was Sequential Attend, Infer, Repeat [98].

One reason that object-oriented representations are useful is that they are able to represent such sparse scene changes efficiently. In particular, when the properties of an object in a scene change (e.g. a teacup is picked up), the change to our representation of the scene is sparse; only those dimensions of the representation devoted to the modified object need to be changed. Contrast this with standard unstructured representations, in which information about the different objects in a scene will be highly entangled: such sparse scene changes may necessitate a change to all dimensions of the representation.

Object-Oriented and Relational Reasoning

Object-oriented representations also support certain patterns of reasoning which can result in powerful generalization abilities. One place where such reasoning can yield immediate benefits is in the design of the prior which predicts the evolution of objects based on their history, though it can also yield benefits for downstream systems making use of the object-oriented representations for other tasks.

Imagine, for instance, modeling an environment consisting of pool balls on a pool table; here an object-oriented representation will maintain a separate state for each ball, and the goal of the prior network is to predict how the balls will evolve at the next timestep given their past trajectories. One naive way that we could structure the prior is as one large Multi-layer Perceptron (MLP; a fully-connected neural network, see Section 2.2.1), which takes the state of all balls as input, and outputs one long vector whose dimensions can be interpreted as predictions about how the different balls will move. However, there are a number of different kinds of structure in this prediction problem that the monolithic MLP fails to exploit. For one, the MLP allows the prediction for every ball to condition on the history of every other ball, despite the fact that the fates of most of the balls are independent from one another; in predicting the one-step evolution of a ball in one corner of the table, we do not need to condition on a ball in the opposite corner, since we know that balls have to touch to affect one another. For another, the MLP uses separate weights

for each ball; however, the balls will all behave nearly identically, and therefore it would be more efficient to use a single set of weights for all balls.

The cited problems with the monolithic MLP approach can be remedied by using graph neural networks, sometimes called relational neural networks [9]. These networks employ a single "kernel" neural network which is applied separately to every ball. When applied to a target ball, the kernel neural network uses the ball's current state, as well as the states of nearby balls, in order to predict the future trajectory of the target ball [22]. This network structure has a number of advantages. First, it ensures that the prediction for each target ball only conditions on other balls which could plausibly affect it (i.e. nearby balls), simplifying the prediction task by forcing the network to ignore irrelevant information. Second, each ball is processed identically, by a single set of weights; this reduces the number of parameters, and also allows a kind of strong generalization, where information that we learn about one ball automatically transfers to all similar balls. Note that this kind of processing, and the benefits that we reap from it, are reminiscent of the local receptive fields and weight sharing found in convolutional neural networks (see Section 2.2.2). This is no accident, as graph neural networks originally arose out of attempts to generalize convolutional neural networks to unstructured (i.e. graph-like) data [13, 14].

Productivity

Productivity refers to the ability of compositional systems to generate arbitrarily complex structures by repeatedly applying compositional rules that combine parts into larger wholes [115]. Physical scenes, viewed as collections of objects, have a kind of productivity in that scenes can contain an arbitrary number of objects, and for any given scene we can obtain a new, more complex scene by adding an object at some location.

In order to be able to represent structures generated by a productive system, our representation should be productive as well. Object-oriented representations meet this criteria; for example, if we know that we are going to be encountering scenes with large numbers of objects (a sporting event, for example), then an object-oriented representation can adapt

by simply adding additional object slots. Contrast this with a system, like a vanilla world model, which uses a flat, unstructured representation: since the dimensionality of the representation is fixed and has no useful structure, there is no systematic way to accommodate larger numbers of objects. Such unstructured systems will often be limited to representing scenes with the numbers of objects that were present in the training data, as their entangled latent representations will be optimized for those cases [42]. In contrast, there are several examples of object-oriented systems being able to handle scenes with more objects than scenes present in the training data [16, 59].

Interpretability

One problem with representations learned by deep models is that they are often not interpretable; the dimensions of the representation will not necessarily correspond to attributes which can be intuitively understood by humans [35, 167]. This can make it hard for humans to debug these models or make downstream use of the learned representations, and can decrease humans' ability to trust the model's predictions, limiting their use in performance-critical applications [21, 38, 73]. Object-oriented representations, in contrast, being heavily inspired by the representations that humans natively use [19], have the potential to be highly interpretable and thus to help alleviate many of these issues.

3.3.4 Strategies for Learning Object-Oriented Representations

Object-oriented representations are most useful when the per-object slots come to represent actual objects in the scene. We would also like the correspondence between slots and in-scene objects to be stable over time, so that if slot k represents a particular blue cube at time t, it also represents that same blue cube at time $t + 1$ and for all timesteps until the cube ceases to exist. However, except in very simple cases [96], simply splitting the state into slots as we have done so far is usually not enough to achieve either of these goals. Here we outline some additional strategies that can be used to learn object-oriented representations that properly correspond to objects in the world.

Explicitly Modeling Object Properties

One common strategy is to take the per-object states $z_{(t),k}$, and further divide them into subsets of variables representing human-interpretable object properties. For example, in an environment where objects are 2D and exist on the 2D image plane, we could stipulate that $z_{(t),k} = (z^y_{(t),k}, z^x_{(t),k}, z^{\text{app}}_{(t),k})$ where $(z^y_{(t),k}, z^x_{(t),k})$ represents the object's 2D location and $z^{\text{app}}_{(t),k}$ represents information about the object's appearance. If our network were able to extract this information from input videos, then we would start to have representations that accurately reflect how the world is decomposed into objects. Of course, this prompts the question: how do we design our world model such that the network is forced to use these object properties in the way that we intend them to be used?

Structured Generative Models

An effective tool for encouraging the network to learn to represent actual properties of objects in the environment is to design a structured generative/observation model $P_\theta(x_{(t)}|\mathcal{Z}_{(t)})$ which produces output images in an *interpretable way*. In particular, we want generative models that use the object properties as if they represented the properties that we expect them to. In the example of the 2D world used previously, in rendering the output image the generative model should construct the appearance of the k-th object from $z^{\text{app}}_{(t),k}$, and should place that appearance at location $(z^y_{(t),k}, z^x_{(t),k})$ in the output image.

To see why structured generative models are helpful, consider what would happen if we used an unstructured network for the generative model, such as a simple MLP. There would then be nothing forcing the network to use variables $(z^y_{(t),k}, z^x_{(t),k})$ as the position of an object; it could just as easily use those latent dimensions to encode object color or size. Building structured generative models can thus be seen as a way of imposing *meaning* onto certain dimensions of the latent space.

Structured generative models of this kind bear a distinct similarity to graphical rendering engines [80]. Such rendering engines, like our structured generative models, take high-level properties of objects and the environment as input, and produce images as

output. Graphical rendering engines can thus act as a significant source of inspiration in designing our generative models, and we might even hope to use a rendering engine directly as a generative model. However, using off-the-shelf rendering engines can be problematic, as they are generally not differentiable and thus not immediately compatible with the deep learning framework we operate in. This problem can be mitigated to an extent by differentiating through the rendering engine using numerical techniques [42], though the scalability of this approach is unproven. Fortunately, the field of differentiable rendering has seen a great deal of progress in recent years [153]. Thus many OOWMs, including all those presented in this work, construct generative models which are effectively differentiable graphical rendering engines.

Structured Inference and Prediction Networks

Another tool we have for encouraging the network to learn to represent object properties is to build structure into the inference network $Q_\phi(z_{(t),k}|x_{(t)}, \mathcal{R}_{(t)}, z_{(t-1),k}, u_{(t-1)})$ and the prior network $P_\theta(z_{(t),k}|\mathcal{R}_{(t)}, z_{(t-1),k}, u_{(t-1)})$. Both of these provide additional opportunities for the model designer to force the network to treat certain dimensions of the per-object latent representations as if they represented particular properties of objects in the environment We provide two examples as illustration.

We first look at the inference network $Q_\phi(z_{(t),k}|x_{(t)}, \mathcal{R}_{(t)}, z_{(t-1),k}, u_{(t-1)})$. For a particular object index k, we intend for $z_{(t),k}$ and $z_{(t-1),k}$ to represent a single object in the world at subsequent timesteps (i.e. we want the object indices to correspond to true object identity); but how do we ensure this? That is, how do we ensure that it does not happen that $z_{(t-1),k}$ represents, say, a red square, while $z_{(t),k}$ switches to some other object, say a blue circle? One possible approach, which assumes an explicit representation of object position $(z^y_{(t),k}, z^x_{(t),k})$, runs as follows: when Q_ϕ infers $z_{(t),k}$ based on $x_{(t)}$, condition the prediction on only a small subregion of $x_{(t)}$ centered at the object position on the previous timestep $(z^y_{(t-1),k}, z^x_{(t-1),k})$. This limits the information available to Q_ϕ in predicting $z_{(t),k}$, and, to an

extent, will force $z_{(t),k}$ to represent the same object as $z_{(t-1),k}$. This strategy was introduced in Sequential Attend, Infer, Repeat (SQAIR) [98].

For the second example we look to the learned prior network $P_\theta(z_{(t),k}|\mathcal{R}_{(t)}, z_{(t-1),k}, u_{(t-1)})$; once again we assume an explicit representation of object position. In predicting $z_{(t),k}$, we obviously need to take into account properties of the object from the previous timestep $z_{(t-1),k}$, and, since objects can interact, we likely also need to take into account properties of other objects $\{z_{(t-1),j}\}_{j\neq k}$. However, considering *all* other objects is probably unnecessary, at least for simple physical objects, since most interactions will be between nearby objects. We can thus pick a maximum radius r, and only consider objects whose distance from object k is less than r. This strategy was introduced in the Neural Physics Engine [22], and is related to relational reasoning and graph neural networks discussed in Section 3.3.2.

Prior Distributions

The final tool at our disposal for ensuring the network learns properties of objects in the world is introducing a fixed (i.e. non-learned) prior distribution $P(\mathcal{Z}_{(t)})$, used alongside the learned prior $P_\theta(\mathcal{Z}_{(t)}|\mathcal{Z}_{(t-1)}, u_{(t)})$. As we train the network, maximizing the ELBO, we minimize the Kullback-Leibler (KL) Divergence between the posterior distributions yielded by Q_ϕ and the fixed prior P. Thus, by selecting the parameters of this fixed prior judiciously, we can influence the distributions yielded by Q_ϕ. Again we illustrate this idea with an example.

Beyond position and appearance, two important properties of objects are their size $(z^h_{(t),k}, z^w_{(t),k})$ and the network's degree of belief in their presence/existence $z^{\text{pres}}_{(t),k}$, modeled as a value between 0 and 1. In both cases, we have certain prior knowledge about the space of values these properties should be taking on when the model is faithfully representing objects in the world. For one, since objects are generally confined to local regions of the image, the object size should be relatively small in general. Small object sizes can be encouraged by using a prior distribution on $(z^h_{(t),k}, z^w_{(t),k})$ that encourages small values. Similarly, in the typical case where the number of object slots K is large compared to the

number of objects in a typical scene, we want $z^{\text{pres}}_{(t),k} \approx 0$ for most objects. This state of affairs can be encouraged by choosing a prior distribution which pushes the set of presence attributes to be sparse. Without these priors, the network is free to make its internal object representations arbitrarily large and make all objects present, and the model's internal object representations would then likely fail to correspond to objects in the world.

3.4 Literature Review

In the previous section we formally introduced the idea of object-oriented representations and Object-Oriented World Models, discussed their utility, and identified a number of different strategies that can be used to design OOWMs that learn to pick out objects in the environment. In this section we review past work on designing OOWMs.

One way in which past models differ is in their completeness as OOWMs. Different models contain different subsets of the components described in the OOWM template given in Section 3.3.2. In particular, many of the models that we will discuss apply only to images rather than videos, and therefore lack components for modeling object dynamics (i.e. they lack a learned causal prior).

As noted in [109], the space of OOWMs can to a large extent be divided into two camps: Scene Mixture Models [16, 40, 59, 60, 158, 160] and Spatial Attention Models [42, 70, 98, 162]. These two model classes differ primarily in the kinds of object properties that are explicitly modeled, and in the structure of their generation and inference networks. Most saliently, Scene Mixture Models generate scenes as per-pixel mixture models (e.g. Gaussian Mixture Models [12]) with the different mixture components corresponding to different object slots, and attribute to each object a soft image-sized pixel ownership mask as well as an image-size appearance map. In contrast, Spatial Attention Models explicitly represent certain interpretable object properties such as presence, position and size, and make use of these properties to perform spatial attention during inference.

3.4.1 Scene Mixture Models

In describing Scene Mixture Models, we restrict ourselves to the case where observations are images $x \in \mathbb{R}^{H_{\text{img}} \times W_{\text{img}} \times 3}$ rather than videos. Let $N_{\text{img}} = H_{\text{img}} \cdot W_{\text{img}}$ be the number of pixels in the image. The main property that Scene Mixture Models have in common is that their generative model $P_\theta(x|\mathcal{Z})$ is structured as a set of coupled per-pixel mixture models. For a model with $K \in \mathbb{N}$ object slots, we first instantiate a set of discrete per-pixel random variables $\{O_i\}_{i=1}^{N_{\text{img}}}$, where O_i takes on values in $\{1, \ldots, K\}$ and $O_i = k$ indicates that pixel i belongs to, or is generated by, object k. We also instantiate a set of per-pixel mixture weights $\gamma \in \mathbb{R}^{K \times N_{\text{img}}}$, where $\gamma_{k,i} = P(O_i = k)$, and a set of per-pixel component distribution parameters $\eta \in \mathbb{R}^{K \times N_{\text{img}} \times F}$, where $\eta_{k,i}$ parameterizes a probability distribution over values for pixel i conditioned on it being generated by object k. The per-object, per-pixel distribution $P(x_i|\eta_{k,i})$ is a simple distribution such as a Normal. Then the probability of an image x is given as a product of the per-pixel mixtures:

$$\begin{aligned} P(x|\gamma, \eta) &= \prod_{i=1}^{N_{\text{img}}} P(x_i|\gamma_{:,i}, \eta_{:,i}) \\ &= \prod_{i=1}^{N_{\text{img}}} \sum_{k=1}^{K} P(O_i = k) P(x_i|\eta_{k,i}) \\ &= \prod_{i=1}^{N_{\text{img}}} \sum_{k=1}^{K} \gamma_{k,i} P(x_i|\eta_{k,i}) \ . \end{aligned}$$

$\eta_{k,:}$ is usually predicted by a neural network that conditions on the k-th latent vector, $\eta_{k,:} = f_\theta(z_k)$; this acts to couple the per-pixel mixtures. As for the mixture weights γ, some models [60, 158] treat O_i as latent variables, so that $\gamma_{:,i}$ act as the parameters of an approximate posterior over O_i. Other models treat $\gamma_{k,:}$ merely as quantities predicted as part of either inference [16] or generation [40, 59]. In general, all these models can be viewed as attributing to the k-th object a soft image-sized ownership mask $\gamma_{k,:}$ and an image-sized appearance map $\eta_{k,:}$.

Having outlined an overarching framework for Scene Mixture Models, we now provide additional details for a few of the most important models. We group these models based on the kind of inference they use: some models perform standard VAE inference using an inference network Q_ϕ, while others use a more complex, multi-step inference scheme which permits them to obtain better estimates of the posterior distribution and, consequently, tighter variational lower bounds [116, 117].

Standard Inference

One standard approach is MONet [16]; it features an inference network which iterates recurrently over objects, predicting a soft ownership mask $\gamma_{k,:}$ on iteration k. Each iterative step conditions on an aggregate mask which indicates to what extent each of the pixels has been claimed thus far. The masks are then passed independently into an object-wise VAE which yields the appearance maps $\eta_{k,:}$.

One downside of MONet is that it is not a well-formed probabilistic model. While it does feature a per-object VAE, the recurrent mask prediction step occurs "outside" of this VAE; there is no probabilistic model for the image as a whole, only for individual components. This means that it is not possible to sample from the model. Another downside is that the sequential mask prediction step is quite slow, as it requires running a convolutional encoder-decoder network on each iteration. A subsequent model GENESIS [40] fixes these issues, creating a well-defined probabilistic model, and employing recurrent latent variable inference which does not need to run a convolutional network at each recurrent step, alleviating the scaling issues.

As for using scene mixture models in support of model-based reinforcement learning, some progress was made in a recent algorithm called COBRA [163]. Operating in a simple 2D sprites-based environment, COBRA uses MONet to learn to extract object-oriented representations from images, and then trains a forward prediction model to predict future object states. This is used in combination with a simple model-based reinforcement learning algorithm, and is shown to outperform model-free approaches. However, both

the environment and the reinforcement learning algorithm used here are fairly simple, and so more work will need to be done scaling up this approach in order for it to be generally useful.

Iterative Inference.

Many Scene Mixture Models make use of Iterative Variational Inference [116, 117] for inferring posterior distributions over their latent variables. These methods replace the inference network Q_ϕ with a trainable, multi-step optimization procedure. Let λ be a set of variables which parameterize an approximate posterior over $\mathcal{Z}$. In the normal VAE framework, we are effectively training a network to map from an input directly to a value λ (see Section 2.3). However, such a simple, direct mapping may not yield optimal values for λ, especially since we are asking the network to achieve this for a large range of inputs x. In Iterative Variational Inference, the direct mapping from input to posterior parameters is replaced with a trainable optimization procedure. Letting λ^0 be an initial guess for λ, the optimization takes the form:

$$\lambda^i = f_\phi(\lambda^{i-1}, \nabla_\lambda \mathcal{L}(x, \phi, \lambda)) \; .$$

Here f_ϕ is a neural network which learns to update parameters of the approximate posterior based on the gradients of the ELBO. This iterative procedure can often obtain better posterior estimates than the direct mapping strategy used in VAEs. It has been hypothesized to be especially important in the case of Scene Mixture Models, as (among other reasons) it helps deal with the multi-modality introduced by the permutation invariance of the object decomposition [59].

One of the first models to make use of iterative inference, Neural Expectation Maximization (NEM) [60], treats the pixel ownership variables O_i as latent variables and the per-object variables z_k as "parameters" or point estimates; its iterative inference procedure thus resembles Expectation Maximization [12]. Relational Neural Expectation Maximiza-

tion (R-NEM) [158] improves on NEM by structuring the parameter update function f_ϕ as a graph neural network, allowing it to better model interactions between objects [9]. However, both NEM and R-NEM are only able to deal with black and white scenes. A subsequent model called IODINE [59] takes a similar approach but properly models the per-object variables z_k as latents, and is able to deal with complex, colored scenes.

While these iterative inference models were primarily designed for images, they can handle videos in an ad-hoc way by feeding in different frames of the video on successive steps of the iterative inference procedure. A successor to IODINE, OP3 [160], accounts for time more explicitly, performing multiple optimization steps for each timestep of a given input video and instantiating a graph neural network to explicitly model object dynamics. The resulting network was shown to be useful for object-oriented planning in interactive environments.

3.4.2 Spatial Attention Models.

Unlike Scene Mixture Models, Spatial Attention Models explicitly represent a number of object properties, including object appearance z_k^{what} and pose z_k^{where}; the pose is typically specified as a bounding box, $z_k^{\text{where}} = (z^y, z^x, z^h, z^w)$. These models get their name from a particular operation that is used in their inference network. For each object slot k, the inference network first predicts and samples from a distribution over object pose, yielding sampled pose z_k^{where}. We then need to predict a distribution over z_k^{what}; this is done by first using differentiable spatial attention (based on Spatial Transformers [83]; see Section 2.2.3) to "inspect" the contents of the image within the bounding box specified by z_k^{where}, allowing the appearance prediction to condition exclusively on information within the predicted spatial extent of the object. Also, the generative networks for these models are structured and can be thought of as differentiable object renderers: they predict an appearance for each object from z_k^{what} and use inverse Spatial Transformers to place that appearance at location z_k^{where} in the output image.

The earliest member of this class, Composited Spatial Transformed VAE (CST-VAE) [79], was relatively bare bones, including only those components described in the previous paragraph. In particular, it lacked any means of modeling the *number* of objects in the image; the same number of object slots was used for every image, regardless of the number of objects it actually contained. A subsequent model, Attend, Infer, Repeat (AIR) [42], remedied this by adding additional binary *presence* variables, $z_k^{\text{pres}} \in \{0, 1\}$. The inference network Q_ϕ took the form of a recurrent neural network which predicted variables for a different object slot on each iteration. Iteration was halted once an object with $z_k^{\text{pres}} = 0$ was predicted. The discrete presence variables introduced non-differentiability, but this was fixed by resorting to stochastic gradient estimation techniques based on reinforcement learning. Through training, the model was able to learn when to stop iteration for a particular image, equivalent to learning how many object were in the given image.

AIR spawned a number of follow-up papers. [165] introduced a novel means of regularizing the posterior distribution via structural constraints, though was only successful in relatively limited setting in which objects do not overlap. Sum-Product Attend, Infer, Repeat (SuPAIR) [147] used Sum-Product Networks to model object appearances, which allowed latent appearance variables to be marginalized out during optimization and resulted in significantly faster learning. Discrete AIR [162] introduced discrete latent variables for modeling object appearances, allowing the network to learn to cluster object appearances, and also devised a formulation of Spatial Transformers which allowed object rotation to be modeled in addition to size and location.

All of the Spatial Attention Models mentioned so far operate on images, rather than video. Sequential Attend, Infer, Repeat (SQAIR) [98] was the first to tackle videos, extending AIR and turning it into a full-fledged OOWM. A similar model, DDPAE, was proposed concurrently with SQAIR [78]. A subsequent model, STOVE [99], addresses a similar problem as SQAIR and DDPAE, but makes a number of design choices (such using a graph neural network for future state prediction, and modeling object velocity explicitly) which result in an OOWM that is significantly better at predicting trajectories far into the

future (though this comes at the cost of no longer having the ability to model videos with varying numbers of objects). This improved long-term prediction accuracy allows STOVE to be used effectively as part of a model-based reinforcement learning algorithm (recall that this was our original motivation for building world models), albeit in relatively simple environments. Finally, Tracking by Animation (TbA) [70] is a related non-probabilistic model which is able to learn to track objects over time, using many of the concepts present in SQAIR. However, it lacks a causal prior, and so cannot predict object trajectories into the future. Also, since it is not probabilistic, it cannot be used to generate scenes.

3.4.3 Other Approaches

A number of other approaches to building OOWMs have been proposed which do not fall within either of these two main paradigms.

The majority of OOWMs described so far rely on an image reconstruction loss for learning object extraction. While this is generally effective, models trained in this way often miss small objects, since such objects receive comparatively little weight in the reconstruction loss. This is problematic, as small objects can be every bit as important as large objects for performing downstream tasks (think of the ball in the Pong or Breakout Atari games). Consequently, Contrastive Structured World Models (C-SWM) [96] formulate an alternative objective for learning to extract objects, based purely on predicting the next object state based on the current state. The resulting model is simple, does not require a generative network, and should treat large and small objects equally. However, it also has a number of downsides, including requiring that all objects are present in every scene, an inability to model multiple instance of the same object class, and inability to handle objects which move stochastically. Thus, more work is needed on this promising approach.

Another possible signal for training OOWMs without supervision is *motion*: pixels which move together tend to belong to the same object. This idea is closely to observations from developmental psychology indicating that, from a very young age, infants expect objects to move through space as cohesive and bounded wholes [146]. Capitalizing on

this idea, models CDNA and STP [46] find out about objects by learning to predict video optical flow in an object-wise manner. Similarly, SfM-Net [161] takes pairs of subsequent video frames as input, and from them predicts an optical flow map that is segmented into objects. SfM-Nets have been adapted for use in interpretable, object-oriented reinforcement learning [54]. One obvious downside of motion-based approaches is that they will fail to capture objects which do not move or move only rarely. A fruitful line of work in the future may involve combining ideas about motion with reconstruction-based approach, aiming to obtain the best of both worlds.

3.5 Outstanding Issues

Learning quality object-oriented representations without supervision is a challenging task, and the foregoing list of models represents the significant progress the field has made in recent years. Nevertheless, there is clearly plenty of ground left to cover between the current state-of-the-art and what might be seen as the field's ultimate goal, namely building OOWMs with the ability to reliably discover and model objects in arbitrary novel environments without supervision. Here we touch on a few particular directions in which improvements are possible; these will serve as motivation for the work presented in the next 3 chapters.

3.5.1 Scaling to Large, Many-Object Scenes

One aspect that is clearly lacking in existing models is an emphasis on *scalability*, particularly scaling to large images and large numbers of objects. The largest number of objects modeled in any of the architectures discussed in this chapter is 11 by MONet [16] and IODINE [59] on the CLEVR dataset [86], and the majority of papers allow for significantly fewer objects than that, often no more than 3.

It has been argued that 11 objects should be considered more than sufficient [61], citing evidence that human abilities fall significantly short of 11 objects. For example,

in humans both the parallel individuation system for multiple-object tracking [19] and the visual working memory system [49] are limited to representing at most 4 objects at a time. However, both of these systems are likely only used for certain kinds of (albeit sophisticated) processing. A separate system, the *analog magnitude* system for approximately representing the size of sets of entities, is able to represent magnitudes of much larger sets of objects and presumably has to represent the objects to be enumerated as an intermediate step [19].

One may also question the relevance of human limitations in building AI systems. Artificial systems operate under markedly different constraints, and serve markedly different purposes, than humans. Indeed, *supervised* object detection systems are able to detect very large numbers of objects. In chapters 4 and 5 we show how to use insights about spatial invariance, taken from the supervised object detection literature, to build OOWMs capable of representing and tracking many 10s of objects simultaneously. This focus on scalability will also turn out to have significant side benefits, such as an ability to seamlessly generalize to scenes both larger and/or containing different numbers of objects than seen during training.

3.5.2 3D Object-Oriented World Models

Another feature common to many of the models listed above is that they are 2D models, even when applied to 3D scenes such as CLEVR. That is, they model objects as 2D entities existing on the image plane, rather than 3D entities existing in a 3D environment.

In Scene Mixture Models, for example, each object is modeled primarily as a soft, image-sized ownership mask; in other words, an object is identified with the set of pixels that it occupies in an image or video. Similarly, in Spatial Attention Models objects are identified with their 2D bounding boxes in the image plane. In both cases, the notion of an object is fundamentally 2D, and significantly impoverished compared to humans' internal object representations. In humans, there is evidence that objects are represented as 3D models. For example, in the Shepherd-Metzler mental rotation experiments, subjects were

shown images of two similar-looking 3D objects, and their task was to determine whether one object was a rotated version of the other. It was found that subjects' response time was proportional to the degree of rotation between the objects, suggesting that subjects were performing mental rotations on an internal 3D representation of the depicted object [141].

Humans have a number of other valuable object-oriented abilities which can much more easily be accounted for by 3D object representations than by the 2D image-bound object representations used in previous OOWMs. One example is object permanence, the understanding that objects do not suddenly cease to exist when they pass out of sight [7]. When we explicitly represent the positions of objects in 3D space, we can support object permanence by simply maintaining those object positions after objects become occluded or pass out of our field of view. It is much harder to imagine how to do this with the 2D object representations used in most OOWMs, where objects are closely tied to the pixels that depict them in an image. Similar arguments can be made about the task of keeping track of (possibly unseen) objects as the agent itself moves throughout the world; this is trivial with a 3D representation of object position, and very difficult to account for with a 2D representation.

Another example of an ability that can be uniquely accounted for by 3D representations is intuitive physics, the ability to predict how an arrangement of objects will interact as time unfolds [6]. There are many situations whose outcome is difficult to predict without a proper 3D representation. Consider a very simple example in which two objects are moving apparently toward one another, parallel to the camera plane, and imagine that our task is to guess whether the objects will collide. This cannot be done without knowledge of the relative depths of the objects and their 3D shapes, information which is beyond the scope of models that are limited to 2D representations.

In summary, there are significant potential benefits if we are able to design OOWMs which properly model objects as 3D entities. In Chapter 6, we make progress on this challenge, proposing a novel OOWM which explicitly models the positions of objects in 3D space.

3.6 Conclusion

In this chapter we motivated and introduced the concept of an Object-Oriented World Model, surveyed past attempts at building OOWMs, and concluded by identifying two significant research questions left unaddressed by past work. In the next 3 chapters, we present new research directly addressing these questions.

Chapter 4

Scaling Unsupervised Object Detection Through Spatial Invariance

4.1 Introduction

As discussed in the previous chapter, object-oriented representations of the world are extremely useful: they can be used as input to graph neural networks which can exploit the object-oriented structure to achieve powerful forms of generalization [9], and, being symbol-like representations, they have the potential to be used as input to downstream human-designed symbolic algorithms (i.e. conventional computer programs) [50]. However, obtaining the object-oriented representations in the first place can be difficult; for systems that take raw perceptual information as input, reasoning and learning in terms of objects requires an initial object detection step wherein the pixel-level information is transduced into high-level object representations.

Past machine learning methods that reason in terms of objects have obtained objects either using ad hoc, task/environment specific object detection methods [33, 50], or, in settings where it is possible, using object representations provided directly by the environment [92]. Alternatively, if one has access to a large dataset of images annotated with ground-truth bounding boxes for the objects therein, one could use it to train any of a large

number of sophisticated supervised object detection architectures [130, 134]. However, obtaining such an annotated dataset is expensive in terms of human labor. Thus, in the current chapter, we are interested in devising an object detection method that does not require bounding box annotations at training time. We call this task, which involves both discovering which objects are common in a dataset and learning to detect those objects, *unsupervised object detection*.

One way to address unsupervised object detection is by building Variational Autoencoders (VAEs) [95] with object-oriented latent representations, allocating a separate latent variable for each object (in other words, an image-only Object-Oriented World Model). As we train the VAE to reconstruct input images, the object-oriented structure of the latent representation, combined with an interpretable generative model and well-chosen prior distributions, force the inference network to learn how to decompose images into objects. The inference network essentially becomes an object detector, and no supervision is required.

Attend, Infer, Repeat (AIR) [42] is one model that follows this strategy, but, as we show empirically, has difficulty scaling up to handle large images containing many objects. In this chapter we propose an alternative architecture that avoids these scaling issues by taking inspiration from recent work on *supervised* object detection, which has relied heavily on spatially invariant computations, particularly convolutions [103, 112]. Faster R-CNN, for example, uses convolutional networks to propose object bounding boxes and to extract features from the input image [134], while YOLO uses a convolutional neural network to map directly from an input image to a grid of detected objects [130].

In the current chapter, we propose **Sp**atially Invariant **A**ttend, **I**nfer, **R**epeat (SPAIR), an architecture for unsupervised object detection that is able to handle large, many-object scenes. Overall, SPAIR may be regarded as a VAE with a highly structured, object-like latent representation and a spatially invariant convolutional inference network (inspired by YOLO and Faster R-CNN) that effectively exploits the structure of objects in images in order to achieve scalability.

Through a number of experiments, we demonstrate empirically that whereas competing approaches struggle at unsupervised object detection when images contain more than a few objects, SPAIR scales up well to as many objects as we tested it with. Moreover, we show that SPAIR's ability to exploit spatial invariance allows it to generalize well to images that are larger and/or contain more objects than the images it was trained on. Finally, we show that our network is able to discover and detect objects with enough reliability to facilitate non-trivial downstream tasks.

4.2 Unsupervised Object Detection

We formalize the task of learning to discover objects in images as *unsupervised object detection*. In particular, we assume that the training set is simply a collection of unlabeled images $D_{\text{train}} = \{x_n\}_{n=1}^{|D_{\text{train}}|}$. In contrast, the test set consists of images paired with bounding box annotations indicating the locations of objects, i.e. $D_{\text{test}} = \{(x_n, \ell_n)\}_{n=1}^{|D_{\text{test}}|}$, where $\ell_n = \{(b^y_{n,k}, b^x_{n,k}, b^h_{n,k}, b^w_{n,k})\}_{k=1}^{K_n}$ is the set of bounding box annotations for image x_n, with each box specified by the location of its center $(b^y_{n,k}, b^x_{n,k})$ and its height and width $(b^h_{n,k}, b^w_{n,k})$. We also assume a small annotated validation set D_{val} for hyperparameter selection.

Overall this formulation is similar to supervised object detection (see Section 2.4.1), except that in our case the learner does not have access to an annotated training set. The system must therefore *discover* how to decompose scenes into objects. This models the challenges confronted by an agent attempting to make sense of a novel visual environment with minimal human oversight. Formulating object discovery in a manner similar to supervised object detection allows us to make use of the many insights, tricks and utilities (such as evaluation metrics) that have been invented for that extremely well-studied task. As a final note, in this unsupervised setting we will not concern ourselves with learning the *classes* of objects (an important part of supervised object detection), since class labels will not be available at training time.

4.3 Related Work: Attend, Infer, Repeat

One promising approach to unsupervised object detection is known as Attend, Infer, Repeat (AIR) [42]. AIR is formulated as a VAE [95] with a recurrent inference network, an object-like latent representation, and a generative network that implements a differentiable, object-oriented rendering algorithm. In AIR, the i-th latent object is represented with the latent variables:

$$z_i^{\text{where}} \in \mathbb{R}^4 , \qquad z_i^{\text{what}} \in \mathbb{R}^{N_{\text{what}}} , \qquad z_i^{\text{pres}} \in \{0, 1\} .$$

z_i^{where} specifies the location of the object as a bounding box, z_i^{what} is a feature vector which encodes the appearance of the object, and z_i^{pres} is a binary variable indicating whether the object is present. The set of all object variables for an image is given by $\{(z_i^{\text{where}}, z_i^{\text{what}}, z_i^{\text{pres}})\}_{i=1}^{K_{\text{max}}}$ where K_{max} is a fixed maximum number of objects. At test time the objects represented by these variables can be compared with ground-truth object annotations using standard object detection metrics.

Given an input image x, AIR proceeds by first processing the image using a Multi-layer Perceptron (MLP; a fully-connected neural network, see Section 2.2.1) called q_ϕ^{encode}, yielding an encoding of the input image denoted h. Next it enters a loop; each iteration i, a recurrent network $q_\phi^{\text{recurrent}}$ takes as input the image encoding h and the recurrent hidden state s_{i-1}, and predicts a value for the object bounding box for the i-th object in the scene, z_i^{where}, as well as a new recurrent hidden state s_i. The object bounding box z_i^{where} is used to parameterize a Spatial Transformer, which differentiably extracts a "glimpse" (a rectangular patch) from the image at the specified location (note that this can be viewed as a kind of spatial attention). An additional network q_ϕ^{predict} then processes information extracted from the glimpse and predicts the variables that encode object appearance z_i^{what} and presence z_i^{pres}. Iteration halts as soon as an object with $z_i^{\text{pres}} = 0$ is predicted. As a final step, all predicted objects are fed into the generative network r_θ, which produces an

output image $\hat{x}$ via a differentiable object rendering process. Pseudocode for AIR is given in Algorithm 2.

The total set of latent variables for the network is $\bigcup_{i=1}^{K}\{z_i^{\text{where}}, z_i^{\text{what}}, z_i^{\text{pres}}\}$, where K is the number of iterations that were performed before predicting an object with $z_i^{\text{pres}} = 0$. VAE inference $Q_\phi(z|x)$ is composed of the networks q_ϕ^{encode}, $q_\phi^{\text{recurrent}}$ and q_ϕ^{predict}, while the differentiable renderer r_θ constitutes the VAE generative network $P_\theta(x|z)$. The inference and generative network are both trained by maximizing the evidence lower bound, as described in Section 2.3, using REINFORCE-style gradient estimation [164] to handle the discrete z^{pres} variables. Through this training, it is expected that the inference network will learn to decompose input images in terms of objects, in effect becoming a recurrent object detector.

Algorithm 2 Attend, Infer, Repeat Forward Pass

function AIR(x)
  $h \leftarrow q_\phi^{\text{encode}}(x)$ ▷ Initial encoding of input image
  $z_0, s_0 \leftarrow 0, 0$
  for $i = 1, \ldots, K_{\text{max}}$ **do**
    $z_i^{\text{where}}, s_i \leftarrow q_\phi^{\text{recurrent}}(z_{i-1}, s_{i-1}, h)$ ▷ Predict object location
    $g_i \leftarrow ST(x, z_i^{\text{where}})$ ▷ Extract glimpse using Spatial Transformer
    $z_i^{\text{what}}, z_i^{\text{pres}} \leftarrow q_\phi^{\text{predict}}(g_i, s_i)$ ▷ Predict object appearance and presence
    $z_i = (z_i^{\text{where}}, z_i^{\text{what}}, z_i^{\text{pres}})$
    if $z_i^{\text{pres}} == 0$ **then**
      $K \leftarrow i - 1$
      break
  $\hat{x} \leftarrow d_\theta(z_1, \ldots, z_K)$ ▷ Render objects into output image
  return $\hat{x}, \{z_i\}_{i=1}^{K}$

One question one may ask of AIR is whether it can scale up to large images containing many objects; the paper that introduced AIR [42] restricted itself to images containing at most 3 objects. In the experimental section of this chapter (Section 4.6), we show empirically that AIR in fact struggles in the large-image, many-object setting. This poor scaling can be attributed to at least two aspects of AIR's formulation. First, AIR initially processes the input image with an MLP; this "holistic" style of processing collapses all information in the image into a single flat vector and effectively discards all spatial structure. Second,

the recurrent, sequential nature of AIR means that it has to learn to account for a different object on each recurrent step, but when there are many objects, discovering how to do this becomes a difficult exploration problem.

4.4 Spatially Invariant Attend, Infer, Repeat

Our proposed model, SPAIR, uses the same high-level architecture as AIR, namely a VAE with an object-like latent representation, but reformulates it in a way that takes advantage of spatially invariant computation in order to enable scaling to large images with many objects. We also make a number of improvements not directly related to scaling, including explicitly modeling the relative depth of objects, and making the model fully-differentiable, thereby alleviating the need for REINFORCE-based training. Overall SPAIR is a VAE with a highly structured, object-like latent representation z, a convolutional object-detecting inference network $Q_\phi(z|x)$, and a generative network $P_\theta(x|z)$ that "renders" the detected objects into a reconstructed image in a scalable way. We now describe each of these components in detail.

4.4.1 Object Representation

In SPAIR, an individual object, o, is represented by a collection of named variables, which we will refer to as *attributes*:

$$o^{\text{where}} \in \mathbb{R}^4\,, \qquad o^{\text{what}} \in \mathbb{R}^{N_{\text{what}}}\,, \qquad o^{\text{depth}} \in [0,1]\,, \qquad o^{\text{pres}} \in [0,1]\,.$$

o^{where} decomposes as $o^{\text{where}} = (o^y, o^x, o^h, o^w)$. o^y and o^x specify the location of the object's center, while o^h and o^w specify the object's size. o^{what} acts as a catch-all, storing information about the object that is not captured by other attributes (e.g. appearance). o^{depth} specifies the relative depth of the object; in the output image, objects with higher values for this

attribute appear on top of objects with lower values. o^{pres} specifies the extent to which the object exists; objects with $o^{\text{pres}} = 0$ do not appear in the output image.

One option would be to designate these object variables as our latent variables; however, going forward we will find it useful to be able to define latent variables in a space other than the object space. Consequently, for each object we allocate a separate set of latent variables underlying the object variables:

$$z^{\text{where}} \in \mathbb{R}^4\,, \qquad z^{\text{what}} \in \mathbb{R}^{N_{\text{what}}}\,, \qquad z^{\text{depth}} \in \mathbb{R}\,, \qquad z^{\text{pres}} \in \mathbb{R}\,.$$

Each object variable is computed as a deterministic function of its corresponding latent variable, described in detail in the following sections.

4.4.2 Convolutional, Object-Detecting Inference Network

In this section we describe our inference network which maps from an input image to a set of objects. This network has been carefully designed to scale well to scenes with large numbers of objects. We first provide some intuition for our approach, before delving into details.

One way to achieve scalable object detection would be to first divide the input image into (possibly overlapping) patches, apply a local object detector to each patch, and then pool the detected objects[1]. This strategy, which we will call *Spatial Divide-and-Conquer* (SDC), has a number of significant advantages as compared to an approach that processes input images holistically. For one, an image patch will have significantly less complexity than the image as a whole, which makes life comparatively easy for the local detector; in fact we can make the ratio between image complexity and patch complexity arbitrarily large by considering arbitrarily large images (e.g. satellite or astronomical imagery) while keeping the patch size fixed. For another, we can easily process images of any size, since patches of large images should have the same statistics as patches of small images. Finally,

[1]For now we ignore difficulties related to objects being detected multiple times in different patches.

we can expect improved sample efficiency, since each patch can, to a degree, be interpreted as a separate data point. The SDC approach is justified to the extent that objects are local features of images. Roughly speaking, the patch size must be chosen such that for every object, there is some patch that fully contains it; objects that do not fit inside any patch will be difficult to detect. Note that this strategy is central to most modern supervised object detection architectures [111, 131, 134].

To implement SDC, SPAIR makes use of convolutional neural networks (CNNs); note the similarity of SDC to the principles of weight sharing, local receptive fields, and image size flexibility (discussed in Section 2.2.2) which make CNNs well-suited to image processing. SPAIR's inference network first processes the input image x with a convolutional backbone neural network q_ϕ^{backbone}, yielding a feature volume v:

$$v = q_\phi^{\text{backbone}}(x) \ .$$

For each spatial location (i, j) in v, we will predict a separate object o_{ij}, in the format described in Section 4.4.1. Importantly, predictions for the attributes of o_{ij} will condition on the feature column $v[i, j, :]$, which can be interpreted as encoding information about the (local) receptive field of spatial location (i, j). This scheme can then be interpreted as running a grid of identical local object detectors in parallel, and thus implements SDC, where the receptive fields of the spatial locations in v play the role of the patches from our description of SDC. Moreover, the use of a convolutional network to do the bulk of the processing is much more computationally efficient than a naive implementation of SDC would be (i.e. processing patches fully independently), for reasons discussed in Section 2.2.2. We call each local object detector a *discovery unit*, and the structure of a discovery unit is shown in Figure 4.1.

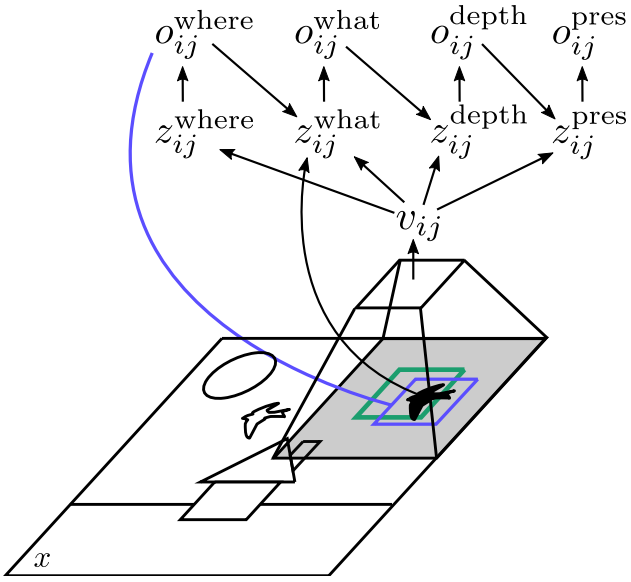

Figure 4.1. Schematic depicting the structure of a discovery unit with indices ij, discovering the black bird. Local information from the frame is first processed by a convolutional filter (trapezoid), which has a receptive field (grey base of the trapezoid) centered on the discovery unit's grid cell (green rectangle). We then predict o_{ij}^{where}, specified with respect to the grid cell (as depicted in Figure 4.3). The object center is required to be inside the grid cell, which ensures that each discovery unit only attempts to account for objects that are within its receptive field. Next, a Spatial Transformer is used to extract information from the frame at o_{ij}^{where} (blue rectangle). Finally, conditioning on this extracted information, we autoregressively predict remaining object attributes o_{ij}^{what}, o_{ij}^{depth} and o_{ij}^{pres}.

Convolutional Receptive Fields, Grid Cells and Image Padding

The information from the input image that each discovery unit has access to is, by definition, its receptive field. It is therefore useful to require the object predicted by a discovery unit to be inside the discovery unit's receptive field. In fact we go one step further, and restrict the object predicted by each discovery unit to have its center inside a small box at the center of the receptive field (a strategy adopted from YOLO [131]).

Let s_ℓ^h and s_ℓ^w be the vertical and horizontal strides, respectively, of the ℓ-th layer in q_ϕ^{backbone}, let L be the number of layers, and assume for simplicity that no pooling is used. Then it can be shown that the distance, in pixels, between the centers of the receptive fields of adjacent spatial locations in the output volume v is $c_h = \prod_{\ell=1}^{L} s_\ell^h$ vertically and $c_w = \prod_{\ell=1}^{L} s_\ell^w$ horizontally [31]. We define a spatial grid over the input image, with the top-left corner of the top-left grid cell aligned with the top-left of the input image,

and we stipulate that the height and width of each cell be (c_h, c_w). For an input frame with dimensions $(H_{\text{inp}}, W_{\text{inp}}, 3)$, the grid has dimensions (H, W), where $H = \lceil H_{\text{inp}}/c_h \rceil$, $W = \lceil W_{\text{inp}}/c_w \rceil$.

If the input image is padded appropriately with zeros before being passed into q_ϕ^{backbone}, then (i, j)-th grid cell will be at the center of the receptive field of the (i, j)-th discovery unit. Thus we will force the object predicted by a discovery unit to have its center inside the discovery unit's grid cell. This has a number of advantages, including ensuring that each discovery unit only predicts objects that are completely contained in its receptive field (assuming that objects are not larger than the receptive field size), and reducing the tendency for neighboring grid cells to predict the same objects. The relationship between receptive fields, grid cells and padding is depicted in Figure 4.2.

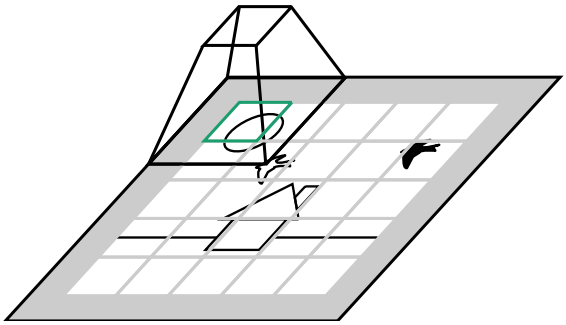

Figure 4.2. Schematic depicting receptive fields, grid cells and padding, as described in Section 4.4.2. The grid of cells implicitly defined by the structure of the convolutional backbone q_ϕ^{backbone} is depicted in gray. The top of the trapezoid is a spatial location in the output of q_ϕ^{backbone}, associated with the grid cell highlighted in green. The bottom of the trapezoid is the receptive field of that spatial location; the frame is padded (grey border area) before being passed into q_ϕ^{backbone} to ensure that all receptive fields are centered on their corresponding grid cells.

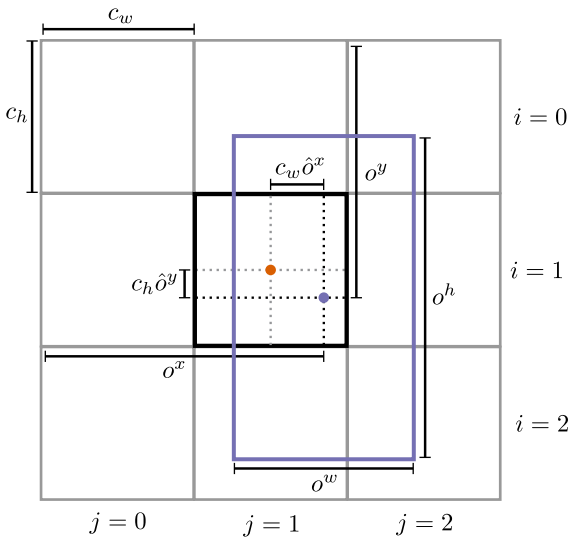

Figure 4.3. Diagram demonstrating the parameterization of object bounding boxes in SPAIR. We focus on a single cell, outlined in black, with indices $(i, j) = (1, 1)$. The blue box represents the bounding box for an object, with the blue dot as its center. The orange dot is the center of the grid cell. Relationships between the quantities in this diagram are given by Equations (4.4.2–4.4.2). To reduce clutter we have omitted the ij subscripts on the variables.

Predicting Object Location

We now begin to describe how the object attributes are predicted; for simplicity we restrict ourselves to a single discovery unit with spatial indices (i, j). The first step is to have an MLP q_ϕ^{where} predict parameters for a posterior distribution over z_{ij}^{where}, and then sample from that distribution:

$$\mu_{ij}^{\text{where}}, \sigma_{ij}^{\text{where}} = q_\phi^{\text{where}}(v[i, j, :]) \,,$$
$$z_{ij}^{\text{where}} \sim N(\mu_{ij}^{\text{where}}, \sigma_{ij}^{\text{where}}) \,.$$

We then deterministically map the latent variables z_{ij}^{where} to the object variables o_{ij}^{where}. Variables z_{ij}^y and z_{ij}^x parameterize object position according to:

$$\hat{o}_{ij}^y = \kappa(\text{sigmoid}(z_{ij}^y) - 0.5) \,,$$
$$o_{ij}^y = (i + 0.5 + \hat{o}_{ij}^y)c_h \,,$$
$$\hat{o}_{ij}^x = \kappa(\text{sigmoid}(z_{ij}^x) - 0.5) \,,$$
$$o_{ij}^x = (j + 0.5 + \hat{o}_{ij}^x)c_w \,,$$

where κ is a positive real number with a default value of $\kappa = 1$; setting $\kappa > 1$ allows the object to have a center slightly outside of the grid cell. Note that the grid cell center has coordinates $(c_h(i + 0.5), c_w(j + 0.5))$, so $(c_h\hat{o}_{ij}^y, c_w\hat{o}_{ij}^x)$ is the displacement of the object from the cell center. See Figure 4.3 for a depiction of the relationships between these variables.

Scale variables z_h^{ij} and z_w^{ij} parameterize the size of the object as:

$$o_{ij}^h = \text{sigmoid}(z_{ij}^h)A_h \,, \qquad o_{ij}^w = \text{sigmoid}(z_{ij}^w)A_w \,,$$

for fixed real numbers A_h and A_w. Specifying object size with respect to A_h and A_w (as opposed to the size of the input image) ensures that z_{ij}^h and z_{ij}^w are meaningful regardless

of the size of the image the network is applied to. Note that (A_h, A_w) can be interpreted as the dimensions of an *anchor box* as used in supervised object detection [134].

Finally, the bounding box for the object is $o^{\text{where}} = (o^y, o^x, o^h, o^w)$.

Extracting a Glimpse

At this point we have predicted an object location; from here, one option would be to predict the rest of the object attributes immediately (as would be done by a single shot object detector like YOLO). Instead, following AIR, we first extract a "glimpse", or sampled image patch, from the input frame x at location o_{ij}^{where}. This will provide the network with relatively high-resolution, fine-grained information about the object's visual appearance which will be useful in predicting the remaining object attributes.

This pattern of predicting object attributes based on an object glimpse mirrors the operation of proposal-based supervised object detectors such as the R-CNN family [69, 134]. However, unlike proposal-based supervised object detectors, for our purposes this glimpse must be extracted *differentiably*, because we do not have ground-truth object annotations with which to train the proposal predictor. To do this differentiable glimpse extraction, we make use of Spatial Transformers [83], parameterized by o_{ij}^{where}, to extract the per-object glimpse g_{ij}:

$$g_{ij} = ST(x, o_{ij}^{\text{where}}) \; .$$

Note that this can be viewed as a form of spatial attention (see Section 2.2.3). This glimpse is then processed by an *object encoder network* q_ϕ^{obj} to yield an object encoding:

$$u_{ij}^{\text{obj}} = q_\phi^{\text{obj}}(g_{ij}) \; .$$

Predicting Object Attributes

To complete object prediction, we need to predict values for the remaining object attributes o_{ij}^{what}, o_{ij}^{depth} and o_{ij}^{pres}. We perform this in an autoregressive, attribute-by-attribute fashion (in order [*what*, *depth*, *pres*]). For each attribute, we use an MLP to predict parameters for a distribution over the latent variable, sample a value from the resulting distribution, and then deterministically map to the object variable. Starting with *what*:

$$\begin{aligned}
\mu_{ij}^{\text{what}}, \sigma_{ij}^{\text{what}} &= q_\phi^{\text{what}}(u_{ij}^{\text{obj}}) \,, \\
z_{ij}^{\text{what}} &\sim N(\mu_{ij}^{\text{what}}, \sigma_{ij}^{\text{what}}) \,, \\
o_{ij}^{\text{what}} &= z_{ij}^{\text{what}} \,.
\end{aligned}$$

In this case the object variable o_{ij}^{what} is *equal to* the latent variable z_{ij}^{what}, since o_{ij}^{what} is a simple feature vector without special structure. Continuing on to *depth* and *pres*:

$$\begin{aligned}
\mu_{ij}^{\text{depth}}, \sigma_{ij}^{\text{depth}} &= q_\phi^{\text{depth}}(u^{\text{obj}}, z_{ij}^{\text{what}}, v[i,j,:]) \,, \\
z_{ij}^{\text{depth}} &\sim N(\mu_{ij}^{\text{depth}}, \sigma_{ij}^{\text{depth}}) \,, \\
o_{ij}^{\text{depth}} &= \text{sigmoid}(z_{ij}^{\text{depth}}) \,,
\end{aligned}$$

$$\begin{aligned}
\omega_{ij}^{\text{pres}} &= q_\phi^{\text{pres}}(u^{\text{obj}}, z_{ij}^{\text{what}}, z_{ij}^{\text{depth}}, v[i,j,:]) \,, \\
z_{ij}^{\text{pres}} &\sim \text{Logistic}(\omega_{ij}^{\text{pres}}) \,, \\
o_{ij}^{\text{pres}} &= \text{sigmoid}(z_{ij}^{\text{pres}}) \,.
\end{aligned}$$

o_{ij}^{pres} can be regarded as an instance of a BinConcrete random variable [114]. Recall from Section 2.3.4 that a BinConcrete is a continuous relaxation of a Bernoulli, to which the reparameterization trick can be straightforwardly applied. Note that AIR used regular, discrete Bernoullis for its version of the *pres* attribute, a choice which necessitated the

use of reinforcement learning. In contrast, SPAIR's use of BinConcretes makes it fully differentiable, alleviating the need for reinforcement learning.

Conditioning Between Discovery Units

One concern with the Spatial Divide-and-Conquer strategy employed by SPAIR is that it is possible for a single object in the image to be detected by multiple discovery units. This may happen, for example, when the center of the object is near the border between two adjacent grid cells; the discovery units corresponding to both of these grid cells may feel that they have equal claim to the object.

One way to address this is by imposing an ordering on the discovery units (e.g. row major ordering according to the grid), and then processing units according to that ordering. The networks that predict object attributes, q_ϕ^{where}, q_ϕ^{depth} and q_ϕ^{pres}, can then be modified to condition on (take as input) nearby objects that occur earlier in the ordering. This would allow the discovery units to coordinate and avoid explaining the same object twice; later units can set their presence attributes to 0 if they see that an object has already been accounted for by an earlier unit. This scheme has similarities to the recurrent, sequential prediction of objects in AIR, though it takes place on a more spatially local scale since we only condition on *nearby* objects. This is the approach taken for all experiments shown in the current chapter (see Section 4.6).

One downside to this style of processing, however, is that it is relatively computationally expensive, as the discovery units cannot be processed in parallel. This is especially problematic for large images, which require large numbers of discovery units and thus have long chains of sequential processing. In subsequent versions of SPAIR, and in later chapters of this thesis in which we make use of encoding modules similar to SPAIR's (see Chapters 5 and 6), we omit the inter-object conditioning, and simply process all discovery units in parallel. We have found that this often achieves quite good performance, while yielding large improvements in runtime. One possible explanation for this reasonable performance is that, as described further below, SPAIR uses a prior distribution on the

z^{pres} variables which encourages the network to use as few objects as possible to explain the input image. Through enough training, this pressure may force the network to learn exactly which objects should be accounted for by which discovery units, thereby avoiding duplicate detections.

4.4.3 Object-Rendering Generative Network

The VAE generative network is responsible for taking in the predicted object representations $\{(o_{ij}^{\text{where}}, o_{ij}^{\text{what}}, o_{ij}^{\text{depth}}, o_{ij}^{\text{pres}})\}_{(i,j)=(1,1)}^{(H,W)}$ and mapping them to an output image $\hat{x}$. One possibility would be to use a relatively unstructured architecture for this stage, such as an MLP or Transpose Convolutional network, mapping directly from the set of objects to the output image. However, doing so would likely limit the quality of the learned object-like representations; the network as a whole could learn to use the object attributes to encode image properties in a way that does not reflect our intention that the latent variables represent objects. Thus, following AIR, SPAIR uses a highly structured generative network which makes use of the object attributes in an interpretable way, thereby endowing them with meaning and forcing them to represent the object properties that we intend them to.

The generative network may be regarded as a differentiable 2D object rendering engine, with limited support for 2.5D in the form of relative object ordering. Object appearance and transparency are predicted from o_{ij}^{what}, the object is placed at location o_{ij}^{where} via inverse Spatial Transformers, o_{ij}^{depth} parameterizes a differentiable approximation of relative depth, and object transparency is multiplied by o_{ij}^{pres} so that objects are only rendered to the extent that they exist.

Objects are rendered on top of a background image x^{bg}. One option would be to augment SPAIR with a separate pathway for inferring the background image from the input image. In the current chapter we take a simpler approach, and assume images have monochrome backgrounds. We either use a single color, obtained by taking the statistical mode of a small sampling of training images, or we train a Multi-Layer Perceptron to map from each image to a background color.

Concretely, the first step of rendering is to predict appearance and partial transparency maps for each object, based on o_{ij}^{what}, using an *object decoder network* r_θ^{obj}:

$$\begin{aligned}\beta_{ij}^{\text{logit}}, \xi_{ij}^{\text{logit}} &= r_\theta^{\text{obj}}(o_{ij}^{\text{what}}) \,, \\ \beta_{ij} &= \text{sigmoid}(\mu^\beta + \sigma^\beta \beta_{ij}^{\text{logit}}) \,, \\ \xi_{ij} &= \text{sigmoid}(\mu^\xi + \sigma^\xi \xi_{ij}^{\text{logit}}) \,.\end{aligned}$$

The appearance map β_{ij} has shape $(H_{\text{obj}}, W_{\text{obj}}, 3)$, while the partial transparency map ξ_{ij} has shape $(H_{\text{obj}}, W_{\text{obj}}, 1)$, for integers H_{obj}, W_{obj}. The variables $\mu^\beta, \sigma^\beta, \mu^\xi$ and σ^ξ are scalar hyperparameters that can be used to control the relative speed with which appearance and transparency are trained.

ξ_{ij} is multiplied by o_{ij}^{pres} to ensure that objects are only rendered to the image to the extent that they are present, yielding a final transparency map:

$$\alpha_{ij} = \xi_{ij} \cdot o_{ij}^{\text{pres}} \,.$$

Next we combine α_{ij} with o_{ij}^{depth} to get an *importance* map, which will be used to implement relative object depth:

$$\gamma_{ij} = \alpha_{ij} \cdot (1 - o_{ij}^{\text{depth}}) \,.$$

Objects with smaller depth will have larger importance values, all else being equal, and will therefore receive greater weight at pixels where multiple objects overlap.

For each object we now have 3 maps, each with spatial shape $(H_{\text{obj}}, W_{\text{obj}})$: appearance β_{ij}, transparency α_{ij} and importance γ_{ij}. To this we add an additional *indicator map*, $\mathbb{1}_{ij}$, consisting of all ones; this will be used to determine which pixels are within the bounding box of each object. Next, we use *inverse* Spatial Transformers (see Section 2.2.3) to create

image-sized versions of these maps, with the input maps placed at location o_{ij}^{where}:

$$\alpha'_{ij}, \beta'_{ij}, \gamma'_{ij}, \mathbb{1}'_{ij} = ST^{-1}([\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \mathbb{1}_{ij}], o_{ij}^{\text{where}}) \; .$$

For each pixel of these image-sized output maps, the value is obtained by sampling from the input map at a location determined by o_{ij}^{where} (for details see Section 2.2.3 on Spatial Transformers). However, some of these sampling locations will lie "outside" of the input map, and for these we use a default value of 0.

Finally, we can perform the final step of building the rendered image from the image-sized per-object maps. We construct a final aggregated indicator map $\mathbb{1}' = (\sum_{k=1}^{HW} \mathbb{1}'_k) > 0$, which has value 1 for every pixel that is within the bounding of some object, and 0 elsewhere. Then the rendered image is computed as:

$$\hat{x} = \mathbb{1}' \left(\frac{\sum_{k=1}^{HW} (\alpha'_k \beta'_k + (1 - \alpha'_k) x^{\text{bg}}) \gamma'_k}{\sum_{\ell=1}^{HW} \gamma'_\ell} \right) + (1 - \mathbb{1}') x^{\text{bg}} \; . \tag{4.1}$$

Here we have indexed objects using a single indices k and ℓ rather than the usual spatial indices (i, j).

Equation (4.1) may be interpreted as follows. For every pixel that is not in the bounding box of any object, we directly take the value of x^{bg}. At all other pixels, we use a convex combination of per-object appearance values, where the mixing weights are given by $\gamma'_k / \sum_{\ell=1}^{HW} \gamma_\ell$. Recall that $\gamma_k = \alpha_k \cdot (1 - o_k^{\text{depth}})$, so in order to receive large weight in the mixture, an object needs to have a high value for α and a small value for o_{ij}^{depth}. Note that for pixels that are only affected by one object, the weight will be 1 for that object, while the default value of 0 used by the inverse Spatial Transformer ensures that $\gamma'_k = 0$ for all other objects. Thus, for such pixels, the output will be given by $\hat{x} = \alpha'_{k^*} \beta'_{k^*} + (1 - \alpha'_{k^*}) x^{\text{bg}}$, where k^* is the index of the single object affecting the pixel.

The output of rendering is a reconstructed image $\hat{x}$ with dimensions $(H_{\text{inp}}, W_{\text{inp}}, 3)$, where each entry is between 0 and 1. To obtain $P_\theta(x|z)$, we use this image to parameterize

a set of (conditionally) independent Bernoulli random variables, one for each pixel and color channel.

Note that all computations in this section can be parallelized to a high degree, across both pixels and objects (except for the step which requires summation over objects). However, for large frames this scheme can still be expensive in terms of both memory and computation. Thus in practice we use an equivalent (but more complex) implementation that avoids explicitly constructing image-sized maps for each object.

4.4.4 Prior Distribution

An important component of a VAE is the prior distribution $P(z)$ over the latent variables. For all real-valued latent variables, we assume that the priors are independent Normal distributions:

$$
\begin{aligned}
z_{ij}^{\text{where}} &\sim N(\mu^{\text{where}}, \sigma^{\text{where}}) \ , \\
z_{ij}^{\text{what}} &\sim N(\mu^{\text{what}}, \sigma^{\text{what}}) \ , \\
z_{ij}^{\text{depth}} &\sim N(\mu^{\text{depth}}, \sigma^{\text{depth}}) \ .
\end{aligned}
$$

Here $(\mu^{\text{where}}, \sigma^{\text{where}}, \mu^{\text{what}}, \sigma^{\text{what}}, \mu^{\text{depth}}, \sigma^{\text{depth}})$ are hyperparameters, which can be chosen to impose prior knowledge about the statistics of objects. For most of these a default of $\mu = 0$ and $\sigma = 1$ is sufficient. The lone exception is the prior over the object size latent variables z_{ij}^{h} and z_{ij}^{w} (recall that these are contained in z_{ij}^{where}); for these using a mean well below 0 was found to be important, as it forces the network to use small, self-contained objects which generally correspond to the kinds of objects we want the network to discover. For example, this can discourage the network from trying to account for two objects in the image using a single object slot.

For the presence latent variables z_{ij}^{pres}, we design a prior that puts pressure on the network to reconstruct the image using as few objects as possible (i.e. as few $o_{ij}^{\text{pres}} \approx 1$ as possible), similar to the prior used by AIR. This pressure is necessary for the network to

extract quality object-like representations; without it, the network is free to set all $o_{ij}^{\text{pres}} = 1$, and to use multiple latent objects to explain each object in the image. For details on this prior, see Appendix A.1.

4.4.5 Training

SPAIR is trained in the usual VAE fashion by maximizing the variational evidence lower bound (ELBO):

$$\mathcal{L}(x, \theta, \phi) = \int Q_\phi(z|x) \log P_\theta(x|z) dz - D_{KL}(Q_\phi(z|x) \parallel P(z)) \, .$$

The optimization is performed using the ADAM variant of Stochastic Gradient Ascent [94]. The required gradients can be computed using standard automatic differentiation software [1, 124]. The expectations are estimated using a single sample from Q_ϕ, and the reparameterization trick is used to backpropagate through this sampling process [95]. For the Normally distributed random variables z^{where}, z^{what} and z^{depth} the reparameterization trick is straightforward, and it can also be done for z^{pres} since we have chosen to model them as BinConcrete random variables in place of the discrete Bernoullis used in AIR.

4.5 Scaling Properties of SPAIR

Having presented SPAIR in detail, we now discuss how different aspects of its architecture contribute to its ability to scale to the large-image, many-object setting.

4.5.1 Comparison with AIR

Earlier we cited two key aspects of AIR [42] that lead to scaling difficulties:

1. An initial encoding step that reduces the whole image to a single unstructured vector.

2. A recurrent, sequential object prediction process.

In this section, we discuss how SPAIR's architecture helps fix these issues. To drive home why these are problematic, it is helpful to consider the problem of discovering objects in satellite or astronomical imagery, wherein images may in principle be arbitrarily large and contain arbitrarily large numbers of objects.

Recall that the first step of AIR's inference network is to encode the input using an MLP, yielding a single, unstructured vector that must encode the entire image. This is problematic for a number of reasons. For one, it may be difficult to compress information about all the objects in an arbitrarily large image into a single flat vector. For another, the number of parameters in an MLP scales poorly as the size and complexity of the input image grows.

One way to fix these issues, without going the whole way to what we propose in SPAIR, would be to simply replace AIR's initial MLP with a convolutional network, yielding an encoding of the image in the form of a feature volume v rather than a flat vector h. This would both preserve spatial structure of the input image *and* greatly reduce the number of parameters, thanks to the local receptive fields and weight sharing employed by the CNN.

However, this would not solve all of AIR's scaling issues, as the recurrent object prediction process presents its own problems. One useful way to view the situation is to think of AIR as having K_{max}-many object prediction "heads" (recall that K_{max} is the fixed maximum number of objects that AIR can predict). The k-th prediction head predicts the object on the k-th iteration of AIR's recurrent loop, taking the image encoding as input along with the recurrent state. Importantly, each head is unconstrained, and is permitted to predict objects anywhere in the image. Contrast this with SPAIR, where each object prediction head (i.e. each discovery unit) is only permitted to predict objects within a small area of the image (the discovery unit's grid cell). Moreover, each of SPAIR's object prediction heads takes as input only those features in the volume v which are directly relevant for predicting objects in its little part of the image (the discovery unit's receptive field), and is architecturally forced to ignore everything else.

4.5.2 Out-of-Distribution Generalization

An important measure of the quality of a trained neural network is the extent to which it can generalize to data that is significantly different from data that was present in the training set. The architecture of SPAIR gives it a built-in ability to generalize to two kinds of out-of-distribution data: images containing different numbers of objects (including *larger* numbers of objects) than seen during training, and images that are spatially larger than those seen during training. We empirically demonstrate SPAIR's ability to generalize along both these dimensions in Section 4.6.3.

The ability to generalize to images containing different numbers of objects than seen during training comes from the Spatial Divide-and-Conquer strategy that SPAIR relies on. To see this, first consider how AIR would view such images. Because AIR processes images holistically (it first processes images using an MLP which ignores spatial structure in the image), images that contain significantly more objects than seen during training may look, from AIR's perspective, significantly different from the images seen in the training data. An additional problem with AIR's use of an MLP to process the image is that each layer uses a fixed number of units, and will have to learn, in each layer, distributed representations of the input images. If the network is only ever trained on images containing, say, 3 objects, then the distributed code that it learns through training will be optimized for representing 3 objects. When it is later faced with an image containing 5 objects, there is no guarantee that the learned code will succeed, as the network may not have learned how to represent 5 objects in its latent code.

In contrast, SPAIR spatially divides the input image into patches (i.e. those patches being the receptive fields of the discovery units), and detects objects in each patch. Handling images with more objects than seen during training should then be straightforward, since the individual patches of the new many-object images will not look different than the patches in the familiar few-object images. The only difference between the many-object images and the few-object images, at a patch level, is that the former will have a greater fraction of patches that contain objects.

SPAIR's ability to generalize to images that are larger than those seen during training has a similar explanation. Since SPAIR effectively sees an image as a collection of patches, the only difference, from SPAIR's point of view, between a normal sized image and a larger image is that the latter contains more patches. Thus, as long those patches still look like patches that were seen in the training set, SPAIR can handle them gracefully. This is a manifestation of the image size flexibility of convolutional neural networks, discussed in Section 2.2.2.

4.5.3 Spatial Invariance vs Spatial Equivariance

Technically, a spatially invariant function is a function f such that $f(T(x)) = f(x)$, where T is any spatial translation operation; as the input to the function is translated spatially, the output does not change. In contrast, a spatially *equivariant* function is a function g such that $g(T(x)) = t_T(g(x))$, where t_T is a function that depends on T; as the input to the function is translated spatially, the output changes in a corresponding manner [55].

A network designed for the task of object *recognition* should be spatially invariant in the above sense; the predicted class should not change as the object is shifted around the image. In contrast, a network designed for the task of object *detection* cannot be spatially invariant in this sense, because its prediction of the object location must change as the object is shifted around the image; it is thus more correct to say that such networks are spatially equivariant. In this formal sense, our use of the term "spatially invariant" as applied to SPAIR is perhaps a misnomer.

However, we intend the term "spatially invariant" to be understood in a looser sense, specifically as it applies to the architecture of the inference network. Our inference network can be understood as a single local object detector being applied repeatedly to a grid of different but overlapping patches from the input image. Thus each of these patches, or chunks of space, is being processed in precisely the same way; so *the computation that is being applied* is invariant with respect to space. It is in this sense that we think of SPAIR as being spatially invariant.

4.6 Experiments

In this section we empirically demonstrate the advantages of SPAIR on a number of different tasks. Source code for these experiments can be found online[2]. Experiment details, including neural architectures, training schedule, and procedures for hyperparameter selection, can be found in Appendix B.

4.6.1 Baseline Algorithm: ConnComp

We devised a simple baseline method which we call ConnComp which detects objects by finding connected components in an image. For each pixel, ConnComp computes the absolute difference between the color at that pixel and the background color (which is given to it). All pixels for which this difference is greater than a threshold τ (a model parameter) are assigned a label of 1, the rest are assigned 0. Each connected cluster of pixels that have label 1 is taken to be an object. We select τ by performing a grid search over its possible values, using the value that maximizes performance on the validation set. Success of ConnComp can be used as a measure of the difficulty of the dataset: it will be successful to the degree that objects do not overlap and are easy to segment.

4.6.2 Comparison with AIR

One of the main benefits that we expect to gain from SPAIR's spatial invariance is a significantly improved ability to discover and detect objects in many-object scenes. To test this, we trained both AIR and SPAIR on 48×48 images, each containing scattered MNIST digits (resized from their usual 28×28 to 14×14) rendered in white on a black background. The goal is to have the models learn to output accurate bounding boxes for the digits in each image, without access to ground-truth bounding boxes for the training set. In order to probe the effect of the number of objects per image on model performance, we used 5 different training conditions; in each condition, the images contain a different

[2] `https://github.com/e2crawfo/auto_yolo`

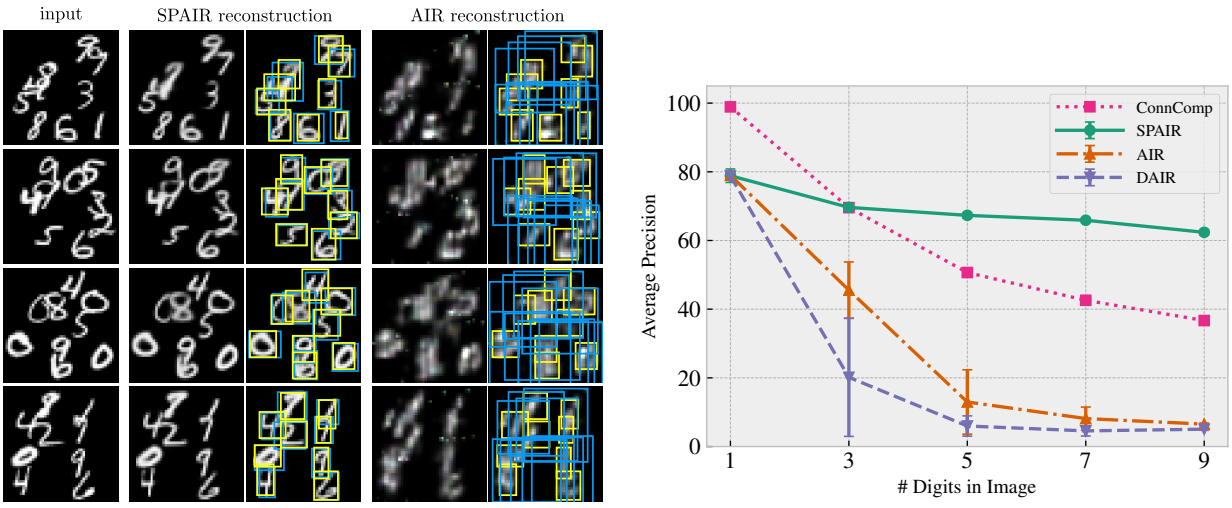

Figure 4.4. Left: Example input images and reconstructions by SPAIR and AIR in the hardest training condition of 9 MNIST digits per image. Predicted bounding boxes are shown in blue, ground-truth boxes in yellow. Right: Average Precision achieved by different algorithms on a scattered MNIST dataset as the number of digits per image varies. For all models except ConnComp (which is fully deterministic), each point is the average over 6 random seeds, and error bars are 95% confidence intervals for the mean.

number of digits, ranging from 1 to 9. Digit instances (i.e. particular MNIST images) are never shared between training, validation and testing datasets, so at test time the networks are seeing digit instances that they have not encountered previously. For each training condition, the training/validation/test datasets have size 64000/1000/1000, respectively. Rejection sampling was used to ensure that in each image, each digit has at most half of its area overlapping with other digits; this restriction ensures that digits do not appear completely on top of one another, while still allowing significant amounts of overlap and ensuring that segmenting the digits from one another is non-trivial.

As a performance measure we use an adapted version of the Average Precision (AP) between the bounding boxes predicted by the model and the ground-truth bounding boxes, a performance measure that is used throughout the supervised object detection literature [44]. In particular, we use AP@IOU=0.1:0.1:0.9; see Section 2.4.2 and `cocodataset.org/#detection-eval` for details on Average Precision.

To simplify the comparison with AIR, we fixed the number of steps executed by AIR's recurrent network to the true number of objects in the image, effectively "telling" AIR how

many objects are present; this allowed us to avoid tuning hyperparameters in AIR that control how the network learns to predict the number of objects in an image. A variant of AIR called Difference Attend, Infer, Repeat (DAIR) [42] was also tested and provided with this same information.

Results are shown in Figure 4.4, and clearly demonstrate that on this task, SPAIR significantly outperforms all tested algorithms when the images contain many objects.

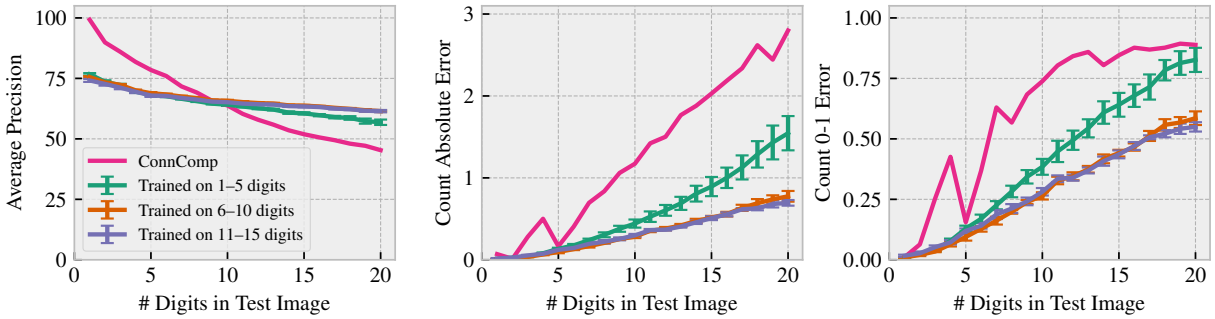

Figure 4.5. Assessing SPAIR's ability to generalize to images that are both larger and contain more objects than images seen during training. Models were trained on small random crops (size 48×48) of large images (size 84×84). The large images contained either 1–5 digits, 6–10 digits or 11–15 digits. Models were then tested on large images containing between 1 and 20 digits. A simple baseline algorithm called ConnComp (Section 4.6.2) was also tested. For all models except ConnComp (which is fully deterministic), each point is the average over 8 random seeds, and error bars are 95% confidence intervals for the mean. Left: Average Precision. The 6–10 and 11–15 lines overlap almost completely. Middle: Absolute difference between the number of objects predicted by the models and the true number of digits. Right: Mean 0-1 error between the number of objects predicted by models and the true number of digits.

4.6.3 Generalization

Another hypothesized advantage of SPAIR's spatial invariance is a capacity for generalizing to images that are larger and/or contain different numbers of objects than images encountered during training. Here we test this hypothesis. We created three different training conditions, each consisting of images of size 84×84 containing randomly scattered MNIST digits of size 14×14. In each training condition, the training images contained different numbers of digits: either 1–5, 6–10 or 11–15 digits. The trained models were then tested on images containing between 1 and 20 digits. To demonstrate that SPAIR models

can be effectively applied to images of different sizes than what they were trained on, the training set actually consists of random crops, each of size 48×48, of full images, while at test time the network was forced to process the full 84×84 images (the network never sees full images during training). Each of the 3 training datasets had 25000 examples, while each of the 20 evaluation datasets contained 1000 examples each. In addition to AP, we also tracked the algorithms' ability to estimate the number of objects in the scene. We did not test AIR or DAIR in this scenario, as their relatively poor performance in the previous section suggests that they are unlikely to succeed in this generalization scenario, which places even greater demands on scalability.

Results of this experiment, shown in Figure 4.5, demonstrate that SPAIR models have significant generalization ability. The performance of all SPAIR models degraded gracefully as the number of digits per test image increased, even well above the maximum number of digits seen during training. There is no significant difference between the performance of models trained on the 6–10 digit condition compared with models trained on the 11–15 digit condition. Models trained on the 1–5 digit condition exhibited lower performance when applied to images containing large numbers of digits, presumably because their training experience did not equip them to deal with densely packed digits.

4.6.4 Downstream Tasks

One principle motivation behind the project of learning to extract objects from scenes is the ability to use those objects in downstream tasks. With this in mind, we perform two experiments confirming that SPAIR networks can be trained to extract objects with enough accuracy to be useful for downstream tasks.

Addition

For the first task, we stay in the domain of scattered MNIST digits, but now rather than asking models to simply locate digits, we ask them to perform an arithmetic operation on the digits, namely addition. We modeled addition as a classification problem, where the

goal is to take in an image and predict a class for the image, where the class corresponds to the sum of the digits in the image. Each training example consists of a 48×48 image containing 5 scattered MNIST digits, paired with an integer label giving the sum of the digits as a one-hot vector. For 5 digits the maximum value of the sum is 45, so there are 46 different classes. We tested models over a number of different training set sizes to probe sample efficiency.

For both SPAIR and AIR, we have an initial training stage where the model ignores the labels and maximizes the ELBO (Equation (2.6)) as usual in order to learn to discover objects. In a second training stage, the output of the inference networks (i.e. the object-like representation) is fed into a downstream classifier, which is trained to minimize cross-entropy classification loss for the addition task. We experimented with two variants of this second training stage. In the first variant (*Fixed*) the weights for the inference network, obtained during the initial training stage, are kept fixed. In the second variant (*Unfixed*) all inference weights are held fixed except for the weights of the object encoder network q_ϕ^{obj}, which are trained to minimize classification cross-entropy loss along with the weights of the downstream classifier.

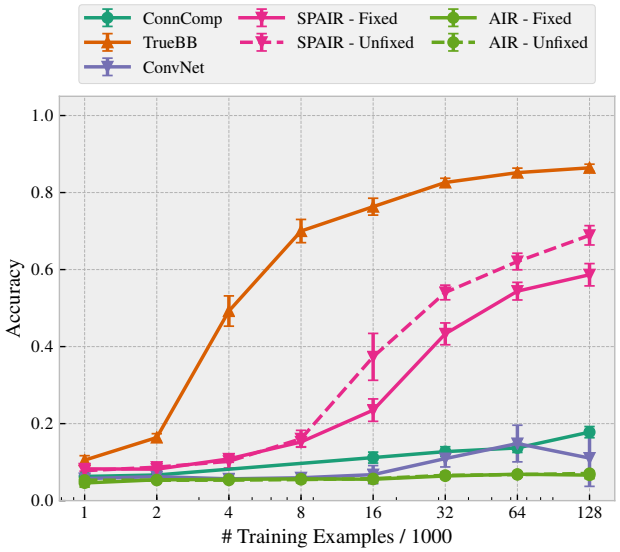

Figure 4.6. Assessing model performance on a task that requires models to take in images containing 5 digits and output the sum of the digits. Each point is an average over 6 random seeds, and error bars represent 95% confidence intervals for the mean.

Beyond AIR and SPAIR, we also tested several other models on this task. The first model, TrueBB (short for True Bounding Box), is given access to ground-truth bounding boxes at both train and test time. Patches are extracted from the image at those bounding boxes and are fed into a downstream classifier. TrueBB is meant to act as an upper-bound on performance, showing what can be achieved with access to perfect object bounding

boxes. The second model, ConnComp, is similar, but obtains bounding boxes using the connected components algorithm described in Section 4.6.2, rather than using the ground-truth boxes. The final model, ConvNet, uses a convolutional network (with the same architecture as SPAIR's convolutional backbone q_ϕ^{backbone}) to extract features from the image, which are once again passed into a downstream classifier. In all cases an LSTM-based [75] downstream classifier was used.

Results of this experiment are shown in Figure 4.6. Both variants of SPAIR significantly outperform all methods other than TrueBB.

The Game of SET

Here we explore using SPAIR as a front-end for learning to play a version of the card game SET [23]. The game of SET works as follows. Let p be the number of *properties* and v the number of *values* per property, for integers p and v. SET is played using a deck of cards; each card has a value for each of the p properties, and the deck contains a single copy of every unique card (and thus has size v^p). A *set* is a collection of v cards such that for each of the p properties, the cards either all have the same value for the property or all have different values for the property. To play a round of SET, the deck is shuffled and a fixed number of cards n are drawn from the top and displayed face up. The goal is to be the first to identify a *set* from the collection of drawn cards, if one exists. The standard game of SET uses $n = 12$, $p = 4$ and $v = 3$, with the four properties being shape, number, color and texture.

Here we consider a simplified version of SET with $n = 7$, $p = 3$, $v = 3$ and shape, number and color as the properties. In this context, we created a dataset designed to train agents to determine, given an image depicting $n = 7$ cards, whether there exists a *set* among those cards. Cards are depicted as colored shapes with number represented by black strokes. Each training example consists of an image of 7 cards and a binary label specifying whether a *set* exists among the cards. To increase perceptual difficulty, we allow the cards to overlap significantly with one another, and use three different background

colors. Training, validation and testing datasets all had equal numbers of positive and negative examples.

The training dataset contained 128,000 examples, while validation and test datasets contained 500 examples each. We tested SPAIR (with a two-stage training schedule similar to the previous MNIST Addition experiment), TrueBB and ConvNet, and used an LSTM with 256 hidden units as the downstream classifier. The results are shown in Table 4.8, and example images and qualitative results from SPAIR are shown in Figure 4.7. SPAIR discovers and detects cards well enough to allow downstream performance equal to what can be obtained using ground-truth bounding boxes. In contrast, the pure convolutional network is unable to make progress on the task. This shows the difficulty of detecting an abstract property of a group of objects for networks that lack an explicit notion of an object.

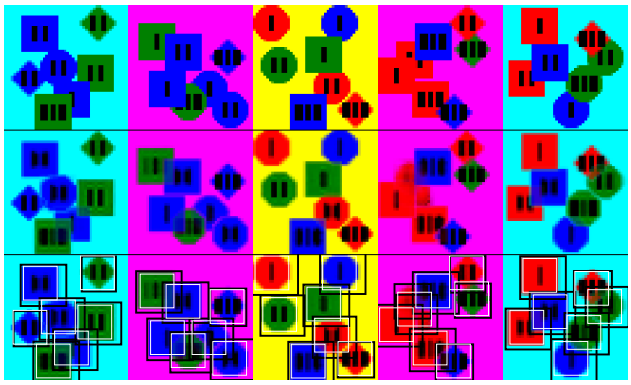

Figure 4.7. Example images for SET. Top: True image. Middle: SPAIR Reconstruction. Bottom: Reconstruction again, with ground-truth bounding boxes in white and predicted boxes in black.

	Test Accuracy (%)
Chance	50.0
SPAIR	57.9 ± 1.99
ConvNet	50.1 ± 0.00
TrueBB	57.7 ± 1.12

Figure 4.8. Performance on the SET dataset, where models are required to compute whether the cards in the input image constitute a *set* (defined in the main text). Each entry is the average over 7 random seeds and the listed uncertainty gives a 95% confidence interval for the mean.

4.6.5 Space Invaders

To push the scaling capabilities of SPAIR further, we trained it on images from the Space Invaders Atari game using the Arcade Learning Environment (ALE) [10], collected with a random policy. The network was trained on random crops of size 48×48; at test time the network processes full images (size 210×160). Qualitative results are given in Figure 4.9.

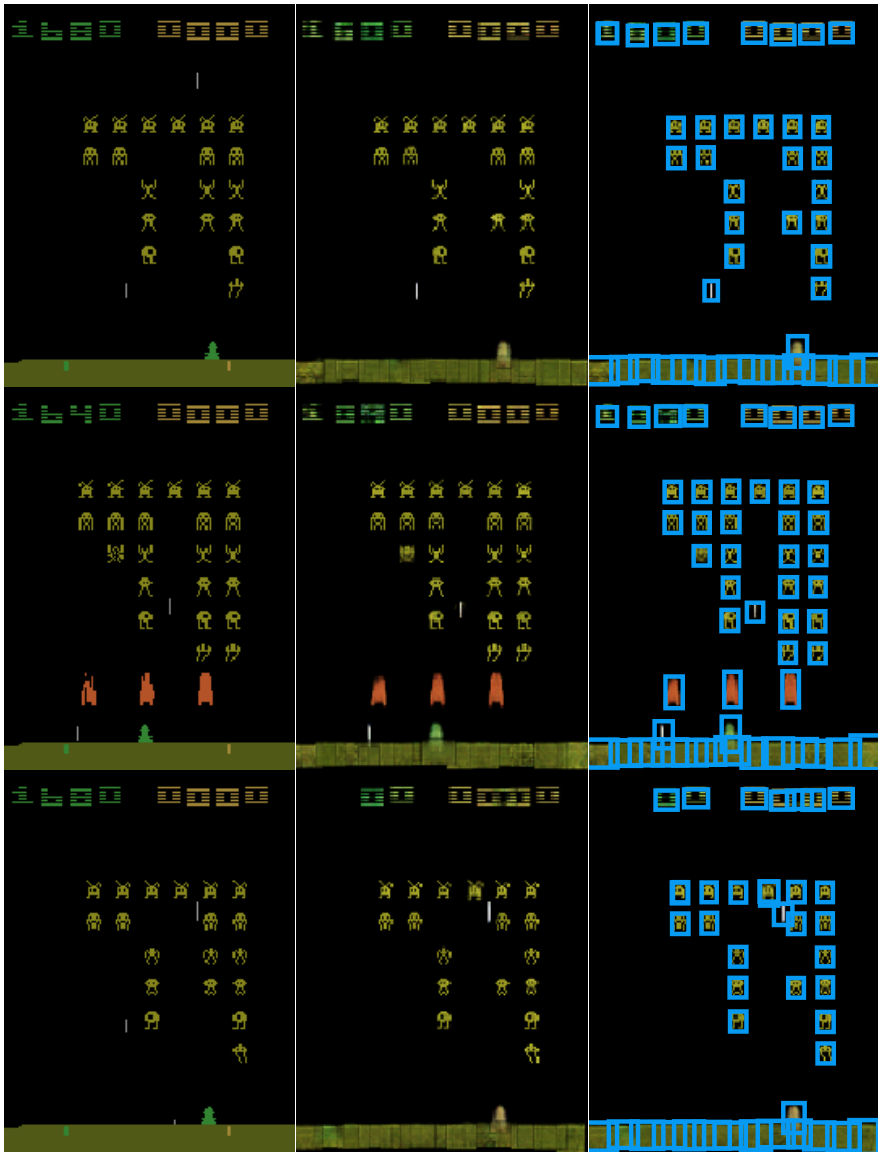

Figure 4.9. Example images from the Space Invaders Atari game (left), and reconstruction (middle) and object bounding boxes (right) yielded by a trained SPAIR model.

4.7 Discussion

In this chapter we introduced SPAIR, a novel architecture for unsupervised object detection which combines features of existing approaches (such as AIR) with the spatial invariance properties of recent supervised object detection architectures such as YOLO and Faster R-CNN. We showed empirically that the addition of this spatial invariance allows for greatly improved scaling. In particular, we showed that our spatially invariant architecture outperforms competing approaches for scenes with many objects, that our approach can generalize to scenes larger and more complex than scenes it was trained on, and that our approach is able to discover and detect objects with enough accuracy to support downstream tasks. In the rest of this section we discuss possible improvements to SPAIR as well as directions for future research.

4.7.1 Grid Cell and Receptive Field Sizes

When applying deep learning in any domain, one of the most important design decisions is the architecture of the neural network. In choosing an architecture for SPAIR, one important consideration is that the parameters of SPAIR's convolutional backbone network determine the size of both the grid cells and the receptive fields.

Recall that in SPAIR we lay a grid of cells over the input image, as discussed in Section 4.4.2. For each grid cell, we instantiate a discovery unit that is responsible for detecting objects whose center is inside the cell. The receptive field of each discovery unit is centered on its assigned grid cell, and usually extends some ways beyond the cell's edges in all directions. Figure 4.1 gives a visual depiction of these relationships. With this picture in mind, we point out a few details to consider when choosing receptive field and grid cell sizes.

First we consider the effect of varying the grid cell size. Note that making the grid cells spatially smaller means that the grid cells and discovery units are more tightly packed, and since the grid needs to cover the whole image, means there are more discovery

units overall. In some respects this is desirable; for example, large numbers of densely packed discovery units are helpful when faced with large numbers of small and tightly packed objects (such as birds in a flock, or fish in a school). On the other hand, having large numbers of discovery units makes running the network (both during training and inference) more computationally intensive, and also creates more potential for neighboring discovery units competing with one another to explain the same objects in the image. In the limit, if we use a backbone network that uses a stride of 1 at every layer, then *each pixel* would be a grid cell, and dealing with the resulting huge number of objects would present a number of difficulties.

Receptive field size is also an important consideration. In Section 2.4.5, we briefly touched on the effect of receptive field size in supervised object detection, and that intuition largely carries over to the unsupervised setting addressed in this chapter. In short, a discovery unit is best at detecting objects that are moderately smaller than the receptive field. If the receptive field is much larger than the object, then there is the potential for the object's visual features to get lost in complex scenes. On the other hand, if the receptive field is smaller than the object, then the discovery unit does not have all the information that it needs to accurately predict the spatial extent of the object.

In sum, if one has prior knowledge about the size and density of objects that our model is likely to encounter, then the size and density of the grid cells and the size of the receptive fields should be chosen accordingly. However, we have previously stated that one of our goals in developing OOWMs is building agents which can autonomously discover objects in novel environments. The requirement of prior knowledge about the density and scale of objects clearly gets in that way of that goal. One possible way around this is to predict objects at multiple layers of the backbone convolutional network, as technique used in a number of supervised object detection networks [111, 132], as discussed in Section 2.4.5. At early layers of the convolutional network, receptive fields are small and grid cells are small and densely packed, making these layers well-suited to discovering small and/or densely packed objects. Deeper layers of the network have larger receptive fields and grid

cells and are thus better suited to predicting larger objects. Different sizes of anchor boxes can be chosen at different layers to ensure that the sizes of the objects predicted at each layer are appropriate for the scale of the layer. One attractive approach is to choose the anchor boxes to be the same size as the receptive field, which would directly enforce the intuition that discovery units should not be predicting objects that are larger than their receptive fields.

4.7.2 Sample Efficiency

As we have stressed throughout this chapter, SPAIR employs an object detection strategy that we call Spatial Divide and Conquer (SDC): it implicitly divides an input image up into separate regions and applies an identical object detector independently to each region. In our experiments we empirically showed a number of benefits from this strategy, such as an improved ability to handle many object scenes, and an ability to generalize to images that are both larger and contain more objects than images seen during training. However, SDC may offer an additional advantage that we did not directly probe, namely improved sample efficiency.

Sample efficiency is a property of a learning algorithm that describes the amount that the algorithm is able to learn from a given amount of training data, as measured by test set performance. A more sample efficient algorithm should be able to achieve higher test set performance from a given training dataset, especially for small training sets. Sample efficiency is an important aspect of any machine learning algorithm, because obtaining training data is usually costly, and so we want to build algorithms that are able to stretch that data as far as possible.

We expect the SDC strategy to give SPAIR improved sample efficiency as compared to AIR. The argument for this conjecture runs as follows. Suppose the training dataset contains N images; the goal is to learn to detect objects in those images without supervision. The benefit of SDC is that it allows us to take the problem of detecting objects in an image and decompose it into a collection of more-or-less independent subproblems, where each

subproblem is to detect objects in a grid cell (i.e. a small patch of the image). Recall that the shape of the grid employed by SPAIR is (H, W), so each image effectively yields HW-many subproblems. Finally, recall that in SDC we are applying a single object detector independently to each subproblem, so in a certain sense we can view that object detector as being trained on NHW-many different examples. Contrast this with AIR, which views each image holistically and thus must be viewed as being trained on N examples. To empirically verify this conjecture, one could train both AIR and SPAIR on a range of image datasets of different sizes (i.e. different values of N). Our prediction is that SPAIR should be able to achieve high test set performance using smaller datasets than AIR, even if control for SPAIR's well-established advantage at detecting objects in many-object scenes (shown in Section 4.6.2) by only considering images with small numbers of objects.

4.7.3 Objects and Backgrounds

For any OOWM, we can ask what criteria it uses to determine what to count as an object. In Section 3.3.1 we discussed one possible criterion for an object-based decomposition of the world, namely modularity [61]. A modular decomposition is one in which 1. the components have internal structure, 2. the components are relatively independent of one another, and 3. the components can appear in many different contexts. Both AIR and SPAIR can be viewed as learning to decompose images into objects by optimizing the modularity of the decomposition imposed by the network. In SPAIR, the internal structure of each object is modeled by the weights of the object encoder and decoder networks, q_ϕ^{obj} and r_θ^{obj}. The objects are independent of one another in that they each account for different portions of the image. Finally, we are applying the same SPAIR network to every image, which clearly allows identical or at least similar objects to appear in different images (and, indeed, for SPAIR to learn properly it must be the case that similar objects appear in many different training images so that its weights can adjust to handling those objects). Similar points can be made for AIR.

Both AIR and SPAIR are helped along to an extent by the fact that the images they segment have solid black backgrounds (or monochrome, as in the experiment with the game of SET in Section 4.6.4). Throughout this work we have considered images in which the objects overlap heavily, so the task of segmenting foreground objects from each other is still non-trivial. However, the restriction to simple backgrounds will have to be remedied as we look to start applying SPAIR-like methods to more realistic datasets.

The challenge with handling images with non-trivial backgrounds is that the model is trained to minimize reconstruction loss. This means that the visual details of the image backgrounds will have to be present in the reconstruction. However, we would not want the objects in SPAIR's latent layer to account for the background details, because then it would not be clear which of the latent objects were modeling objects in the image and which were simply accounting for background details. One possible way to handle this would be to train a standard, unstructured VAE to account for the background, at the same time we as we are training SPAIR itself. When rendering the output images, the SPAIR objects would be rendered "on top of" the image emitted by this background VAE. The issue is, how do we prevent this background network from trying to account for the entire scene, leaving nothing to be modeled by the SPAIR network (and thus preventing it from discovering objects)? One potential approach would be to restrict the capacity of the background network. Indeed, we did a very simple version of this in our SET experiment, where we instantiated a network that learned to model the color of the background; this can be viewed as an extremely low-capacity background VAE that is restricted to producing monochromatic images. Several follow-up works have had further success with this approach, combining a SPAIR-like network for handling foreground objects with a low-capacity background network, enabling the discovery of objects in collections of images with non-trivial backgrounds [85, 109].

4.8 Retrospective

Since the publication of SPAIR in February of 2019, many papers (including work discussed in the next two chapters) have extended it or made use of its insights.

SPAIR's most direct successor is probably a model called SPACE [109]. SPACE equips a SPAIR-like model with a means of inferring a background image from the input image, a feature which allows SPACE to model more complex scenes than SPAIR, including a much larger subset of games from the Atari ALE. SPACE also showed, concurrently with several other models including SILOT [29] (the subject of the next chapter) and SCALOR [85], that conditioning between objects is largely unnecessary, and that removing it can greatly speed up both training and inference.

As we will see in the next section, SILOT and SCALOR both extend SPAIR to be able to track dynamic objects in videos rather than static objects in images. A model called ROOTS [24], which addresses the problem of discovering static objects in 3D scenes, can also be regarded as an extension of SPAIR. It takes SPAIR's strategy of placing a grid over the input image and positing a separate object slot for each cell of the grid, and translates it into 3D, placing a virtual 3D grid over the static 3D scene. In a similar vein, a model called 3DOM [30], the subject of Chapter 6, is a more direct descendant of both SPAIR and SILOT which is able to discover dynamic objects in 3D scenes.

Chapter 5

Exploiting Spatial Invariance for Scalable Unsupervised Object Tracking

5.1 Introduction

In the previous chapter we introduced Spatially Invariant Attend, Infer, Repeat (SPAIR), an Object-Oriented World Model (OOWM) for scalable unsupervised object detection in images. In the current chapter, we aim to translate insights from SPAIR to the video setting, designing a full-fledged OOWM capable of discovering and tracking objects in cluttered, many-object videos. Many of the challenges encountered in designing a scalable OOWM for images are also present when moving to videos, and there are new challenges as well, such as accommodating the extra computational complexity required to train on (potentially long) videos.

Working with videos does have its advantages, however. In some ways the task of discovering objects becomes easier, because coherent motion of visual features across time is a strong indicator of objectness. The field of developmental psychology has identified at least 3 principles that are central to human detection of objects: cohesion (objects are connected and bounded, and their parts move together), continuity (objects move on connected paths, and cannot jump around) and contact (objects cannot interact at a

distance) [146]. In building systems that discover objects, it would seem worthwhile to pay heed to these principles of human object detection, especially if we want the system to discover the same kinds of objects that humans do. However, none of these three principles can be leveraged so long as we restrict ourselves to static images.

In the current chapter, we aim to make progress on this task of discovering and tracking objects in video, which we call *unsupervised object tracking*, with a particular focus on cluttered videos containing large numbers of objects. Our general approach is to formulate a scalable OOWM; through training we expect the OOWM's inference network to detect objects and track them over time. This OOWM is composed of a number of modules that are applied each timestep of the input video, notably a Discovery module which detects new objects in the current frame (with a convolutional architecture similar to SPAIR), and a Propagation module which updates the attributes of objects discovered in previous frames based on information from the current frame. For each input frame, a Rendering module creates a corresponding output frame from the objects proposed by the Discovery and Propagation modules. The network is trained by maximizing the evidence lower bound (ELBO), which encourages the output frames to be accurate reconstructions of the input frames. At the end of training, it is expected that the Discovery module will have become a competent object detector, while the Propagation module will have learned to track objects from one frame to the next.

The high-level architecture just described (i.e. an OOWM divided up into Discovery, Propagation and Rendering modules) was first proposed in a model known as Sequential Attend Infer Repeat (SQAIR) [98] (named for the fact that it extends AIR to the sequential/video setting). However, as we demonstrate empirically, SQAIR struggles at processing spatially large videos that contain many densely packed objects. Similar to AIR, SQAIR's scaling problems stem from an initial, holistic image encoding step that discards spatial structure, and a recurrent mode of predicting objects. We show that we can once again address these issues using object detection and updating schemes based on the

Spatial Divide-and-Conquer strategy introduced in the previous chapter, in combination with a number of new strategies.

In the current chapter, we propose **S**patially **I**nvariant **L**abel-free **O**bject **T**racking (SILOT) (pronounced like "silo"), a differentiable architecture for unsupervised object tracking that is able to scale well to large scenes containing many objects. SILOT achieves this scalability by making extensive use of spatially invariant computations and representations, thereby fully exploiting the structure of objects in images. For example, in its Discovery module SILOT employs a convolutional object detector with a spatially local object specification scheme, and spatial attention is used throughout to compute objects features in a spatially invariant manner. Through a number of experiments, we demonstrate the concrete advantages that arise from this focus on spatial invariance. In particular, we show that SILOT has a greatly improved capacity for handling large, many-object videos, and that trained SILOT networks can generalize well to videos that are larger and/or contain different numbers of objects than videos encountered in training.

5.2 Spatially Invariant, Label-free Object Tracking

Assume we are given a length-T input video $x_{(0:T-1)} = \{x_{(t)}\}_{t=0}^{T-1}$. SILOT is a Variational Autoencoder (VAE) that models each input video as a collection of moving objects (i.e. an OOWM). It is divided into modules: a Discovery module, which detects objects from each frame; a Propagation module, which updates the attributes of previously discovered objects; a Selection module, which selects a small set of objects to keep from the union of the discovered and propagated objects; and a Rendering module which renders selected objects into an output frame. The Discovery, Propagation and Selection modules constitute the VAE inference network, while the Rendering module constitutes the VAE generative network.

5.2.1 Object Representation

The primary data type in SILOT is a set of objects, each set having space for a fixed number of object slots. An object set contains a collection of named variables, called attributes, each of which can be indexed by slot k.

For a generic object set $\mathcal{O}_{(t)}$ defined for time t, the attributes for object k are:

$$\mathcal{O}^{\text{where}}_{(t),k} \in \mathbb{R}^4\,, \quad \mathcal{O}^{\text{what}}_{(t),k} \in \mathbb{R}^{N_{\text{what}}}\,, \quad \mathcal{O}^{\text{depth}}_{(t),k} \in [0,1]\,, \quad \mathcal{O}^{\text{pres}}_{(t),k} \in [0,1]\,, \quad \mathcal{O}^{\text{state}}_{(t),k} \in \mathbb{R}^{N_{\text{state}}}\,.$$

We define $\mathcal{O}^{\text{where}}_{(t),k} = (\mathcal{O}^{y}_{(t),k},\ \mathcal{O}^{x}_{(t),k},\ \mathcal{O}^{h}_{(t),k},\ \mathcal{O}^{w}_{(t),k})$; $(\mathcal{O}^{y}_{(t),k}, \mathcal{O}^{x}_{(t),k})$ gives the location of the object's center on the camera plane, while $(\mathcal{O}^{h}_{(t),k}, \mathcal{O}^{w}_{(t),k})$ gives the object's apparent size. $\mathcal{O}^{\text{what}}_{(t),k}$ acts as a catch-all, storing information about the object that is not captured by other attributes (e.g. appearance, velocity). $\mathcal{O}^{\text{depth}}_{(t),k}$ is used to implement a differentiable approximation of relative object depth, so that objects with smaller depth are rendered on top of objects with larger depth. $\mathcal{O}^{\text{pres}}_{(t),k}$ specifies the extent to which the object exists; smaller values of $\mathcal{O}^{\text{pres}}_{(t),k}$ make the object more transparent during rendering, and objects with $\mathcal{O}^{\text{pres}}_{(t),k} = 0$ do not show up at all. $\mathcal{O}^{\text{state}}_{(t),k}$ is the hidden state of a recurrent network, and summarizes the history of the object up to time t. Note that when the slot index k is left off (e.g. $\mathcal{O}^{\text{what}}_{(t)}$), the expression refers to the collection of attribute values for all object slots in the indicated object set.

Note that since SILOT is formulated as a VAE, each of these object attributes (with the exception of $\mathcal{O}^{\text{state}}_{(t),k}$) has an underlying *latent* variable. Each object variable is a deterministic function of its underlying latent variable. We separate the object variables from the latent variables as we will generally find it useful to define latent variables in a space other than the highly structured space of objects. This kind of separation has been used in a number of past models, including SPAIR [42, 98]. We denote the set of latent variables corresponding to an object set by adding an overbar to the name of the object set, so that $\overline{\mathcal{O}}_{(t)}$ is the set of latent variables for object set $\mathcal{O}_{(t)}$.

5.2.2 Overview

For each timestep t of the input video, we consider a number of sets of variables:

1. Discovered objects $\mathcal{D}_{(t)}$, and corresponding latent variables $\overline{\mathcal{D}}_{(t)}$.
2. Propagated objects $\mathcal{P}_{(t)}$, and corresponding latent variables $\overline{\mathcal{P}}_{(t)}$.
3. Selected objects $\mathcal{S}_{(t)}$.

At a high-level, SILOT is composed of a number of modules, and modules communicate with one another by passing around object sets. This high-level structure, along with the generative model assumed by SILOT, is visualized in Figure 5.1. There we see the relationships between the variables and modules that we have introduced thus far. Both the generative model and high-level network structure are similar to and inspired by SQAIR [98]. Within a timestep t, the flow of computation proceeds as follows:

1. **Propagation.** Input frame $x_{(t)}$ and selected objects from the previous frame $\mathcal{S}_{(t-1)}$ are passed into the Propagation module, which predicts a set of latent attribute updates $\overline{\mathcal{P}}_{(t)}$ and deterministically applies them to $\mathcal{S}_{(t-1)}$, yielding propagated objects $\mathcal{P}_{(t)}$.
2. **Discovery.** Input frame $x_{(t)}$ and propagated objects $\mathcal{P}_{(t)}$ are passed into the Discovery module, which discovers objects in the frame that are not accounted for by any propagated object, first yielding $\overline{\mathcal{D}}_{(t)}$ and then $\mathcal{D}_{(t)}$ via a deterministic transformation.
3. **Selection.** $\mathcal{P}_{(t)}$ and $\mathcal{D}_{(t)}$ are passed into the Selection module, which chooses a subset of the objects to retain going forward, yielding $\mathcal{S}_{(t)}$.
4. **Rendering.** $\mathcal{S}_{(t)}$ is passed into the Rendering module which yields an output frame $\hat{x}_{(t)}$.

On the initial timestep, the Propagation module is not executed and instead a null set of objects is used for $\mathcal{P}_{(0)}$ (a set of objects with all attributes set to 0).

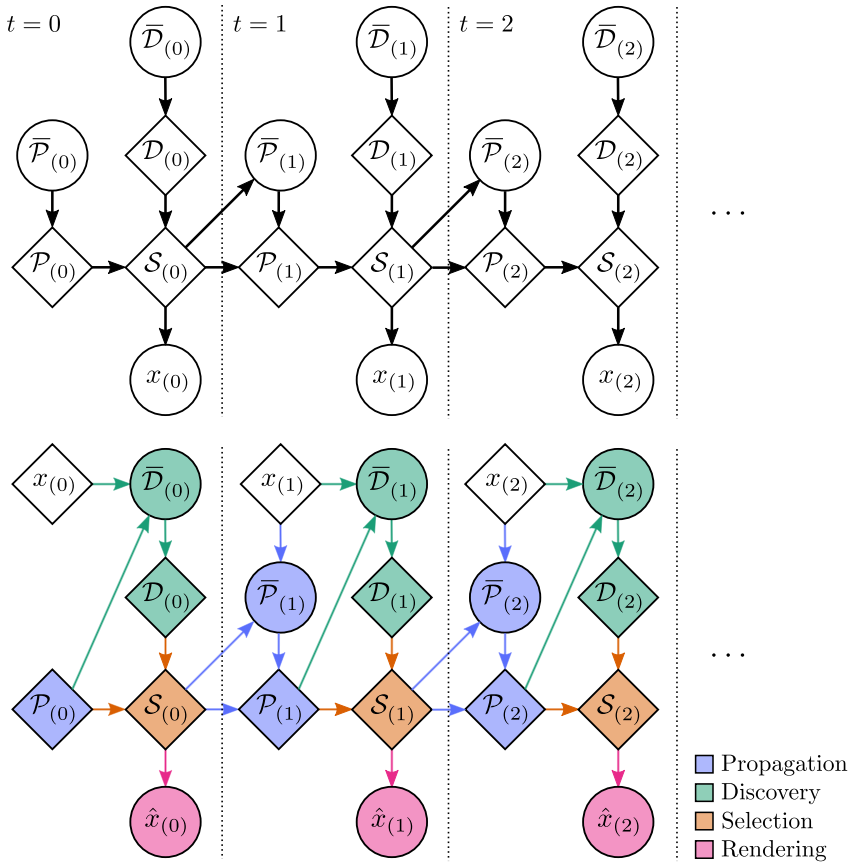

Figure 5.1. High-level overview of SILOT. Left: Generative model assumed by SILOT. Diamonds/circles are deterministic/stochastic functions of their inputs. Right: Structure of the SILOT neural network. Modules are indicated by color. Propagation, Discovery and Selection constitute the inference network $Q_\phi(z|x)$, while Rendering constitutes the generative network $P_\theta(x|z)$.

We now proceed to delve into the details of how each of these modules is implemented, paying particular attention to how each design choice affects the scalability of the overall model.

5.2.3 Discovery

The role of the Discovery module is to take in the current frame $x_{(t)}$ and the set of propagated objects for the timestep $\mathcal{P}_{(t)}$, and detect any objects in the frame that are not yet accounted for by any propagated object. Such objects may be unaccounted for because they appeared for the first time in the current frame, or because they were not previously discovered due to error. Object discovery in SILOT is modeled closely after the

convolutional inference network developed in SPAIR, employing the principle of Spatial Divide-and-Conquer, and we can similarly think of it as a grid of identical object detectors, called discovery units, detecting objects in different local subregions of the input image.

Indeed, for the majority of details about the Discovery module, we refer the reader to the description of SPAIR's convolutional inference network given in Section 4.4.2, with the Discovery latent and object variables $\overline{\mathcal{D}}_{(t)}$ and $\mathcal{D}_{(t)}$ suitably mapped onto SPAIR's latent and object variables z and o. Here we highlight a few additional features of the Discovery module which are missing from SPAIR's inference network.

The first step in Discovery is nearly identical to SPAIR, using a convolutional network d_ϕ^{bu} (analogous to SPAIR's convolutional backbone) to extract "bottom-up" information from the current input frame $x_{(t)}$, mapping to a feature volume $v_{(t)}^{\text{bu}}$:

$$v_{(t)}^{\text{bu}} = d_\phi^{\text{bu}}(x_{(t)}) \, .$$

As in SPAIR, the structure of d_ϕ^{bu} induces a grid of cells over the input image. The cell shape is (c_h, c_w), a function of the strides of the layers in d_ϕ^{bu}. The grid shape and, equivalently, the spatial shape of $v_{(t)}^{\text{bu}}$, are both $(H, W) = (\lceil H_{\text{inp}}/c_h \rceil, \lceil W_{\text{inp}}/c_w \rceil)$, for input images with shape $(H_{\text{inp}}, W_{\text{inp}})$. The input frame is padded so that the receptive field of each spatial location in $v_{(t)}^{\text{bu}}$ is centered on the corresponding grid cell. We predict an object for each spatial location in $v_{(t)}^{\text{bu}}$. For spatial indices ij, the (i, j)-th grid cell, (i, j)-th spatial location in $v_{(t)}^{\text{bu}}$, and (i, j)-th object predictor together constitute a discovery unit, essentially a local object detector. All discovery units are identical, but operate on different local subregions of the input image. The structure of one of SILOT's discovery units is shown in Figure 5.2.

Conditioning on $\mathcal{P}_{(t)}$ with Spatial Attention

The primary difference between the tasks performed by SPAIR's inference network and SILOT's Discovery module is that the Discovery module operates in a temporal context; at each timestep, we have a set of objects $\mathcal{P}_{(t)}$ that the network already knows about, and

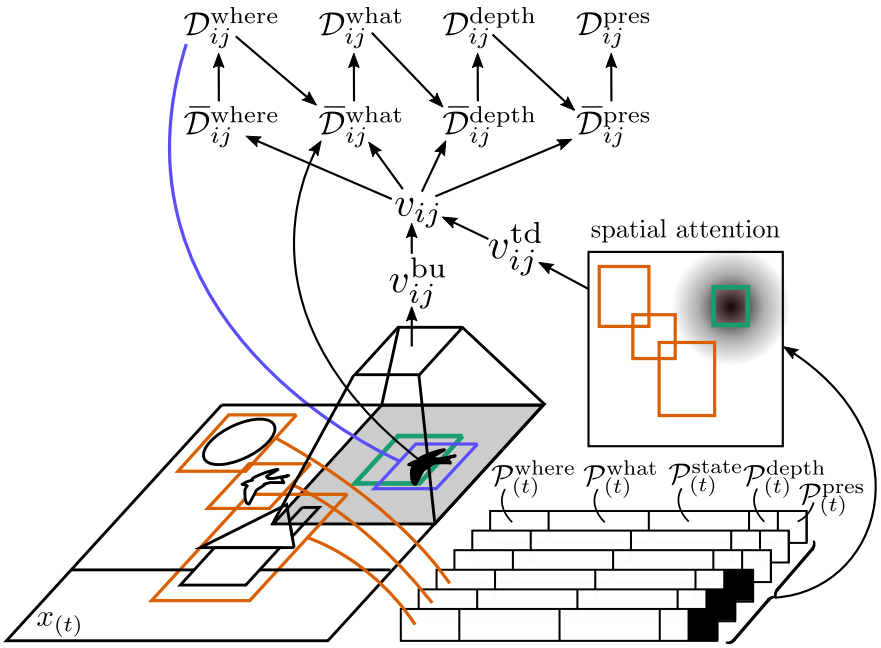

Figure 5.2. Schematic depicting the structure of a discovery unit with indices ij at time t, discovering the black bird which has just come into view from the right. The sun, house and white bird are assumed to have been discovered on a previous timestep, and thus are accounted for in $\mathcal{P}_{(t)}$. Local bottom-up information from the current frame is processed by a convolutional filter (trapezoid), which has a receptive field (grey base of the trapezoid) centered on the discovery unit's grid cell (green rectangle). Next, top-down information about nearby objects propagated from the previous frame (orange boxes) is summarized using spatial attention with a Gaussian kernel centered at the grid cell. Bottom-up and top-down information is then fused and used to autoregressively predict object attributes. $\mathcal{D}_{ij}^{\text{where}}$ is predicted first, specified with respect to the grid cell. The object center is required to be inside the grid cell, which ensures that each discovery unit only attempts to account for objects that are within its receptive field. Next, a Spatial Transformer is used to extract information from the frame at $\mathcal{D}_{ij}^{\text{where}}$ (blue rectangle). Finally, conditioning on this extracted information, we autoregressively predict remaining object attributes $\mathcal{D}_{ij}^{\text{what}}$, $\mathcal{D}_{ij}^{\text{depth}}$ and $\mathcal{D}_{ij}^{\text{pres}}$. To reduce clutter, we have mostly omitted temporal indices (t).

we want the Discovery module to avoid rediscovering objects that are already accounted for in $\mathcal{P}_{(t)}$. In order to achieve this, we give each discovery unit access to information about objects in $\mathcal{P}_{(t)}$ that are near the discovery unit's grid cell. The expectation is that the discovery units will learn to determine when some object that it would normally detect is already accounted for by some object in $\mathcal{P}_{(t)}$, and, in response, set its own *pres* attribute to 0. For example, one discovery unit might see that the input image contains a red square whose center is inside its grid cell. Under normal circumstances, we would expect the unit to output a representation of this red square. However, if the unit also sees that

$\mathcal{P}_{(t)}$ contains an object that is near its grid cell and has features signaling a high degree red-square-ness, then the discovery unit should skip detecting the red square. Note that each discovery unit only requires information about propagated objects that are *near its grid cell* since each unit is only permitted to predict objects inside its grid cell.

Concretely, we inform the discovery units about nearby propagated objects using a form of spatial attention over objects. Here we narrow our focus to a single discovery unit with spatial indices ij. We use an MLP d_ϕ^{attn} to predict a representation of each propagated object in the context of discovery unit ij. For propagated object with index k, $\mathcal{P}_{(t),k}$, we pass into the MLP all of its attributes, with one exception: we replace the object location $(\mathcal{P}^y_{(t),k}, \mathcal{P}^x_{(t),k})$ with location *relative* to grid cell ij. Noting that the center of grid cell ij has location $(c_h(i+0.5), c_w(j+0.5))$, we define the relative position as:

$$y_{k,ij} = \mathcal{P}^y_{(t),k} - c_h(i+0.5)\ ,$$
$$x_{k,ij} = \mathcal{P}^x_{(t),k} - c_w(j+0.5)\ .$$

And then we define:

$$v^{\text{attn}}_{k,ij} = d^{\text{attn}}_\phi(y_{k,ij}, x_{k,ij}, \mathcal{P}^{\setminus yx}_{(t),k})\ ,$$

where $\mathcal{P}^{\setminus yx}_{(t),k}$ is taken to mean the concatenation of all attributes of $\mathcal{P}_{(t),k}$ except for y and x (which have been replaced by their relative counterparts $y_{k,ij}$ and $x_{k,ij}$).

With these vectors in hand, we use attention to compute a final vector summarizing all information about $\mathcal{P}_{(t)}$ that is relevant for discovery unit ij:

$$v^{\text{td}}_{(t),ij} = \sum_{k=1}^{K} G(y_{k,ij}, x_{k,ij}, \sigma) \cdot v^{\text{attn}}_{k,ij}\ ,$$

where K is the number of objects in $\mathcal{P}_{(t)}$ and G is the probability density for a 0-mean 2-dimensional Gaussian. σ is a hyperparameter controlling the standard deviation of the Gaussian, and determines how "focused" the attention is; smaller values of σ place more

weight on objects closer to the grid cell. This is similar to other forms of attention found throughout machine learning, such as Transformers [159], in that it involves a weighted sum of feature vectors, and the feature vectors that receive large weight are interpreted as the vectors being *attended to*. However, whereas those other forms of attention typically allow the attention weights to be a learned function of the feature vectors, in our case the weights are determined entirely by the spatial proximity (and therefore relevance) of the propagated objects to the discovery unit in question.

Finally, we gather the $v^{\text{td}}_{(t),ij}$ vectors into a volume $v^{\text{td}}_{(t)}$ (with spatial shape (H, W)). Then we pass both bottom-up information $v^{\text{bu}}_{(t)}$ and top-down information $v^{\text{td}}_{(t)}$ into a size-preserving convolutional network d^{fuse}_{ϕ}, which fuses both sources of information into a final volume $v_{(t)}$:

$$v_{(t)} = d^{\text{fuse}}_{\phi}(v^{\text{bu}}_{(t)}, v^{\text{td}}_{(t)}) \, .$$

Predicting Object Attributes

From here the Discovery module largely follows SPAIR: it predicts parameters for distributions over the latent Discovery variables $\overline{\mathcal{D}}_{(t)}$, samples from the predicted distributions, and maps the sampled latents to the objects $\mathcal{D}_{(t)}$. This is done on an attribute-by-attribute basis (in order [where, what, depth, pres]), and is autoregressive, so that predictions for later attributes are conditioned on samples for earlier attributes. For full details, see Section 4.4.2 from the previous chapter. Finally, the hidden state for each object is updated using a recurrent neural network (RNN) called RNN_{ϕ}:

$$\mathcal{D}^{\text{state}}_{(t),ij} = \text{RNN}_{\phi}(\mathcal{D}^{\text{where}}_{(t),ij}, \mathcal{D}^{\text{what}}_{(t),ij}, \mathcal{D}^{\text{pres}}_{(t),ij}, s_0) \, ,$$

where s_0 is a default initial state for the RNN. Note that the hidden state, and the recurrent update thereof, was not present in SPAIR. Its primary purpose is to summarize the past

history of the object, which is used in the Propagation module for locating the object in future frames and predicting the object's future trajectory.

Parallel Execution

As discussed near the end of Section 4.4.2, SPAIR places an ordering on the discovery units, and then has discovery units that occur later in the ordering condition on the objects output by nearby discovery units that occur earlier in the ordering. This was intended to provide a means for adjacent discovery units to coordinate with one another and avoid duplicate detections; however, this approach is quite computationally expensive, as it prevents the discovery units from being executed in parallel. While that performance hit was somewhat acceptable in the static image setting addressed in SPAIR, it becomes much more problematic in the video setting explored in this chapter, as the Discovery module has to be executed on each frame of the input.

Thus, in SILOT's discovery module we omit the inter-object conditioning, and instead simply process all discovery units fully in parallel. We can get a degree of coordination between adjacent discovery units by having d_ϕ^{fuse} use kernel/filter sizes greater than 1, giving each discovery unit information about its neighbours *just before* objects are predicted. Having discovery units share information with their neighbours before object prediction, rather than after (as was the case in the sequential scheme used in SPAIR) may be slightly less powerful, but we found we were able to achieve good performance with it, and duplicate detections were not a problem. Moreover, the increase in speed is well worth it. Note that concurrent work on a similar model [85] reached a similar conclusion.

5.2.4 Propagation

Propagation at time t takes the selected the objects from the previous timestep $\mathcal{S}_{(t-1)}$, and propagates them forward in time, using information from the current frame $x_{(t)}$ to update the object attributes, ultimately yielding propagated objects $\mathcal{P}_{(t)}$. The structure of Propagation for an individual object is shown in Figure 5.3.

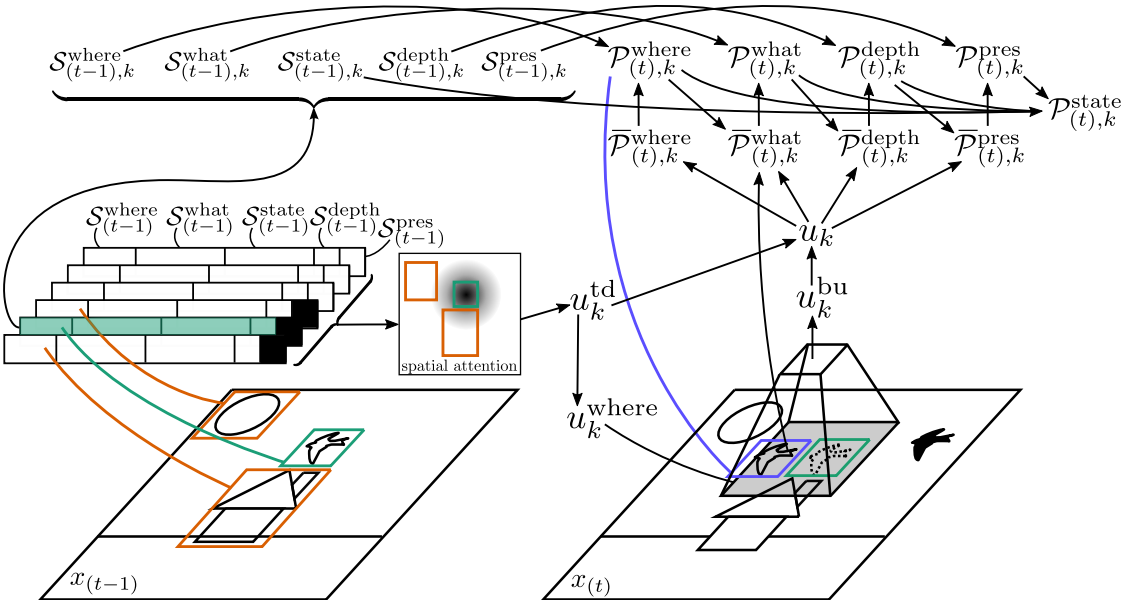

Figure 5.3. Schematic depicting the Propagation module updating an object with index k (indicated by green coloring) which is tracking the white bird. A feature vector for the object, which also takes into account nearby objects, is first created using spatial attention. Next, a bottom-up glimpse (grey region in $x_{(t)}$) is specified with respect to the object's location from the previous time step $\mathcal{S}^{\text{where}}_{(t-1),k}$ (green box in $x_{(t)}$). This bottom-up glimpse is then processed by a neural network (trapezoid) and used to predict an update to the *where* attribute, resulting in $\mathcal{P}^{\text{where}}_{(t),k}$. A glimpse is then extracted at location $\mathcal{P}^{\text{where}}_{(t),k}$ (blue box), and updates to the remaining attributes are predicted and applied autoregressively.

For simplicity we narrow our focus to updating a single object with index k. In order to update the attributes of object $\mathcal{S}_{(t-1),k}$ (and ultimately obtain $\mathcal{P}_{(t),k}$), we first need to *locate* the object in the new frame. To achieve this we leverage two different kinds of information:

1. Top-down information about the object, such as its appearance and an estimate of where it might have moved, as well as information about other objects that may be nearby and may confound our attempts to locate the target object.

2. Bottom-up visual information from the frame, ideally restricted to a relatively small feasible region so that irrelevant visual complexity in the frame can be ignored.

Gathering Top-down Information via Spatial Attention over Objects

For top-down information, our goal is to compute a feature vector summarizing the object's attributes. This feature vector should also contain information about other objects in $\mathcal{S}_{(t-1)}$ that are close to $\mathcal{S}_{(t-1),k}$, as it may be important to compensate for such "distractor" objects in finding the particular object we are interested in. To achieve this, we compute an initial feature vector for each object in $\mathcal{S}_{(t-1)}$ using an MLP. Next, in order to include information about nearby objects, we use a spatial attention process similar to the attention process used in the Discovery module, described in Section 5.2.3. The difference is that whereas in Discovery each discovery unit was computing attention over propagated objects, in the current case the Propagated objects are attending to *each other*.

We begin by using an MLP p_ϕ^{td} to compute an initial feature vector which summarizes the object attributes but does not include information about nearby objects:

$$u_k^{\text{td}'} = p_\phi^{\text{td}}(\mathcal{S}_{(t-1),k}) \; .$$

Then we compute the position of every other object (indexed by ℓ) relative to object k:

$$y_{\ell,k} = \mathcal{S}^y_{(t-1),\ell} - \mathcal{S}^y_{(t-1),k} \; ,$$
$$x_{\ell,k} = \mathcal{S}^x_{(t-1),\ell} - \mathcal{S}^x_{(t-1),k} \; .$$

Next we use an MLP p_ϕ^{attn} to get feature vectors for objects $\mathcal{S}_{(t-1),\ell}$ in the context of target object $\mathcal{S}_{(t-1),k}$:

$$u^{\text{attn}}_{\ell,k} = p_\phi^{\text{attn}}(u_k^{\text{td}'}, y_{\ell,k}, x_{\ell,k}, \mathcal{S}^{\setminus yx}_{(t-1),\ell}) \; ,$$

where $\mathcal{S}^{\setminus yx}_{(t-1),\ell}$ is taken to mean the concatenation of all attributes of $\mathcal{S}_{(t-1),\ell}$ except for y and x (which have been replaced by their relative counterparts $y_{\ell,k}$ and $x_{\ell,k}$).

To obtain the final top-down feature vector for $\mathcal{S}_{(t-1),k}$, we sum the $u^{\text{attn}}_{\ell,k}$ vectors, weighting by a Gaussian kernel centered at $(\mathcal{S}^y_{(t-1),k}, \mathcal{S}^x_{(t-1),k})$, and add the result to $u^{\text{td}'}_k$:

$$u^{\text{td}}_k = u^{\text{td}'}_k + \sum_{\ell=1,\ell\neq k}^{K} G(y_{\ell,k}, x_{\ell,k}, \sigma) \cdot u^{\text{attn}}_{\ell,k} \ .$$

Gathering Bottom-up Information via Glimpse Extraction

The next step is to extract enough "bottom-up" information from the frame in order to locate the object. One approach would be to search within the entirety of the frame; however, the frame may be visually complex, and it is likely that much of the information it contains is irrelevant for updating a particular object k. Thus, we aim to condition only on the subset of the new frame to which the object could plausibly have moved. Our strategy is to extract from $x_{(t)}$ a *bottom-up glimpse*, centered near the object's location from the previous timestep. This glimpse should be as small as possible (so as to eliminate irrelevant visual information) while still being large enough to contain the object after any plausible motion. We obtain the bounding box for this glimpse by predicting an adjustment to the object's bounding box from the previous timestep, using an MLP p^{glimpse}_ϕ:

$$u^{\text{bu-where}}_k = \mathcal{S}^{\text{where}}_{(t-1),k} + p^{\text{glimpse}}_\phi(u^{\text{td}}_k) \ .$$

An alternative option would have been to force $u^{\text{bu-where}}$ to have the same center as $\mathcal{S}^{\text{where}}_{(t-1),k}$, and to obtain its scale as a fixed multiple of the scale of $\mathcal{S}^{\text{where}}_{(t-1),k}$. Allowing the network to predict an adjustment, as we have done here, permits the network to adjust the bottom-up glimpse if it has expectations about how far and in which direction the object will move.

Next we extract a glimpse at $u^{\text{bu-where}}$ using Spatial Transformers (ST), and process it using a neural network p^{bu}_ϕ:

$$g^{\text{bu}}_k = ST(x_{(t)}, u^{\text{bu-where}}_k) \ ,$$
$$u^{\text{bu}}_k = p^{\text{bu}}_\phi(g^{\text{bu}}_k) \ .$$

Finally, with both top-down and bottom-up information in hand, we use a network p_ϕ^{fuse} to fuse both sources of information into a single vector:

$$u_k = p_\phi^{\text{fuse}}(u_k^{\text{bu}}, u_k^{\text{td}}) \ .$$

Predicting Object Location

From here, Propagation proceeds similarly to the Discovery module, and to SPAIR's inference network: we predict a bounding box for the object, extract a glimpse at that bounding box, encode the glimpse, and then predict the remaining attributes. For object localization, we first predict parameters for a Normal distribution over $\overline{\mathcal{P}}_{(t),k}^{\text{where}}$, and then sample from it:

$$\mu^{\text{where}}, \sigma^{\text{where}} = p_\phi^{\text{where}}(u) \ ,$$
$$\overline{\mathcal{P}}_{(t),k}^{\text{where}} \sim N(\mu^{\text{where}}, \mu^{\text{where}}) \ .$$

Next, we compute bounding box parameters as a function of both $\mathcal{S}_{(t-1)}k$ and $\overline{\mathcal{P}}_{(t),k}^{\text{where}}$. We decompose $\overline{\mathcal{P}}_{(t),k}^{\text{where}}$ as $\overline{\mathcal{P}}_{(t),k}^{\text{where}} = (\overline{\mathcal{P}}_{(t),k}^{y}, \overline{\mathcal{P}}_{(t),k}^{x}, \overline{\mathcal{P}}_{(t),k}^{h}, \overline{\mathcal{P}}_{(t),k}^{w})$. Then for the bounding box position we have:

$$\mathcal{P}_{(t),k}^{y} = \mathcal{S}_{(t-1),k}^{y} + A_h \cdot \tanh(\overline{\mathcal{P}}_{(t),k}^{y}) \ ,$$
$$\mathcal{P}_{(t),k}^{x} = \mathcal{S}_{(t-1),k}^{x} + A_w \cdot \tanh(\overline{\mathcal{P}}_{(t),k}^{x}) \ .$$

where (A_h, A_w) are the dimensions of an anchor box, as introduced in SPAIR. Typically anchor boxes are used in predicting object scale rather than position (and we do so below), but they can also be thought of as defining a characteristic scale for changes in position.

For bounding box scale we have:

$$\mathcal{P}^h_{(t),k} = A_h \cdot \text{sigmoid}(\text{sigmoid}^{-1}(\mathcal{S}^h_{(t-1),k}/A_h) + \overline{\mathcal{P}}^h_{(t),k}) \ ,$$
$$\mathcal{P}^w_{(t),k} = A_w \cdot \text{sigmoid}(\text{sigmoid}^{-1}(\mathcal{S}^w_{(t-1),k}/A_w) + \overline{\mathcal{P}}^w_{(t),k}) \ .$$

This update can be interpreted as follows. We take the previous scale, project it back to "logit" space via the inverse sigmoid, apply an update additively in logit space, and then reproject back into "sigmoid" space. Putting these together, the new bounding box for the object is $\mathcal{P}^{\text{where}}_{(t),k} = (\mathcal{P}^y_{(t),k}, \mathcal{P}^x_{(t),k}, \mathcal{P}^h_{(t),k}, \mathcal{P}^w_{(t),k})$.

Next we extract a glimpse at $\mathcal{P}^{\text{where}}_{(t),k}$, and process it using a network p^{obj}_ϕ:

$$g_k = ST(x_{(t)}, \mathcal{P}^{\text{where}}_{(t),k}) \ ,$$
$$u^{\text{obj}}_k = p^{\text{obj}}_\phi(g_k) \ .$$

Predicting Attributes

From here we autoregressively predict new values for the object attributes; this is similar to attribute prediction in the Discovery module, except that rather than directly predicting

attribute values, we predict attribute *updates* and subsequently apply them:

$$\begin{aligned}
\mu^{\text{what}}, \sigma^{\text{what}} &= p_\phi^{\text{what}}(u, u^{\text{obj}}, \mathcal{P}_{(t),k}^{\text{where}}) \,, \\
\overline{\mathcal{P}}_{(t),k}^{\text{what}} &\sim N(\mu^{\text{what}}, \sigma^{\text{what}}) \,, \\
\mathcal{P}_{(t),k}^{\text{what}} &= \mathcal{S}_{(t-1),k}^{\text{what}} + \overline{\mathcal{P}}_{(t),k}^{\text{what}} \,,
\end{aligned}$$

$$\begin{aligned}
\mu^{\text{depth}}, \sigma^{\text{depth}} &= p_\phi^{\text{depth}}(u, u^{\text{obj}}, \mathcal{P}_{(t),k}^{\text{where}}, \mathcal{P}_{(t),k}^{\text{what}}) \,, \\
\overline{\mathcal{P}}_{(t),k}^{\text{depth}} &\sim N(\mu^{\text{depth}}, \sigma^{\text{depth}}) \,, \\
\mathcal{P}_{(t),k}^{\text{depth}} &= \text{sigmoid}(\text{sigmoid}^{-1}(\mathcal{S}_{(t-1),k}^{\text{depth}}) + \overline{\mathcal{P}}_{(t),k}^{\text{depth}}) \,,
\end{aligned}$$

$$\begin{aligned}
\mu^{\text{pres}} &= p_\phi^{\text{pres}}(u, u^{\text{obj}}, \mathcal{P}_{(t),k}^{\text{where}}, \mathcal{P}_{(t),k}^{\text{what}}, \mathcal{P}_{(t),k}^{\text{depth}}) \,, \\
\overline{\mathcal{P}}_{(t),k}^{\text{pres}} &\sim \text{Logistic}(\mu^{\text{pres}}) \,, \\
\mathcal{P}_{(t),k}^{\text{pres}} &= \mathcal{S}_{(t-1),k}^{\text{pres}} \cdot \text{sigmoid}(\overline{\mathcal{P}}_{(t),k}^{\text{pres}}) \,.
\end{aligned}$$

Note that $\text{sigmoid}(\overline{\mathcal{P}}_{(t),k}^{\text{pres}})$ can be viewed as a BinConcrete random variable. Also, notice that Propagation cannot increase the value of the *pres* attribute, due to the form of the update (multiplication by a value < 1). This imposes an inductive bias that objects cannot pop in and out of existence arbitrarily, and ensures that objects are only ever discovered by the Discovery module, which is better equipped for it. As a final step, each object's hidden state is updated to include information on the new attribute values:

$$\mathcal{P}_{(t),k}^{\text{state}} = \text{RNN}_\phi(\mathcal{P}_{(t),k}^{\text{where}}, \mathcal{P}_{(t),k}^{\text{what}}, \mathcal{P}_{(t),k}^{\text{pres}}, \mathcal{S}_{(t-1),k}^{\text{state}}) \,.$$

Comparison with SQAIR

Propagation in SILOT is similar to Propagation in SQAIR, with one significant exception. In SQAIR, objects within a timestep are updated sequentially (similar to AIR's sequential object prediction process); this allows object updates within a timestep to condition on

one another, facilitating coordination between objects and supporting behavior such as explaining away. However, this sequential processing can be computationally demanding when there are large numbers of objects. In contrast, SILOT updates all objects within a timestep in parallel; a degree of coordination between objects is achieved via the spatial attention step.

Comparison with Discovery

The computations that implement the Propagation module have much in common with the operation of the Discovery module. Both roughly follow a pattern of:

1. **Compute top-down information about nearby objects.**
2. **Extract and process bottom-up information required for localizing the object.** In Propagation this is achieved by the combination of the bottom-up glimpse g^{bu} and subsequent processing thereof by network p_ϕ^{bu}. In Discovery, this is achieved using the initial convolutional network d_ϕ^{bu}. Note that Propagation's g^{bu} is thus analogous to the receptive field of a discovery unit.
3. **Predict object location.**
4. **Extract and encode a glimpse at object location.**
5. **Predict remaining attributes.**

The process of updating a particular object k in Propagation is thus analogous to the operations performed by a single discovery unit ij. The primary difference is that the former operates in a temporal context; there is a particular object that we want to find and update attributes for, whereas a discovery unit simply predicts whatever object happens to land in its assigned grid cell.

5.2.5 Selection

Past architectures in this domain, particularly AIR and SQAIR, use discrete Bernoulli random variables for their equivalents of the *pres* attribute, and are thus forced to employ reinforcement learning–based techniques for estimating gradients. In order to avoid this complication and make our architecture fully differentiable, we chose to model the *pres* attributes as real-valued BinConcrete random variables, which are differentiable relaxations of Bernoullis [114] (this approach was also used in SPAIR; see Section 4.4.2).

However, one advantage of the discrete approach used in AIR and SQAIR is that once an object has its *pres* value set to 0, the network can forget about it and stop propagating it forward. In contrast, one consequence of our fully differentiable approach is that the network is not able to "turn off" any object slot; all slots are always present, but to varying degrees. If care is not taken this will result in scaling issues, since the Discovery module yields HW new objects per frame (one object per grid cell); if all these objects are kept and propagated forward, we would end up with a collection of $HW(t+1)$ objects after timestep t, which would quickly become intractable.

To fix this, we use a simple top-K selection strategy wherein we keep only the K objects from the union of $\mathcal{P}_{(t)}$ and $\mathcal{D}_{(t)}$ with highest values for the *pres* attribute, for fixed integer K. While this hard selection step is not differentiable, the number of objects with non-negligible values for the *pres* attribute should be small compared to $HW(t+1)$ (see Section 5.2.7), and we have not found this non-differentiability to cause problems in training as long as K is large enough. As a rule-of-thumb, we typically set K to be roughly 25% larger than the maximum number of objects that we expect to see in a single frame; this ensures there is always room for a reasonable number of objects with low values for *pres* to be propagated and receive gradient feedback. Another simple strategy that would likely work, though may be slightly wasteful, would be to set $K = HW$. Either way, the output of the selection step is the set of selected objects $\mathcal{S}_{(t)}$.

5.2.6 Rendering

The differentiable Rendering module is the sole constituent of the VAE generative network $P_\theta(x|z)$, taking in the selected objects $\mathcal{S}_{(t)}$ and yielding a reconstructed frame $\hat{x}_{(t)}$. Rendering in SILOT is identical to SPAIR's generative network; see Section 4.4.3 from the previous chapter for full details.

5.2.7 Prior Distribution

An important component of an OOWM are the fixed priors over latent variables, $P(\overline{\mathcal{P}}_{(t)})$ and $P(\overline{\mathcal{D}}_{(t)})$ for propagation and discovery respectively, which can be used to influence the statistics of the posterior distributions predicted by the inference network Q_ϕ. For the majority of the latents, we use independent Normal distributions as the fixed priors. However, for the Logistic/BinConcrete random variables $\overline{\mathcal{D}}^{\text{pres}}_{(t)}$ and $\overline{\mathcal{P}}^{\text{pres}}_{(t)}$ we design a prior that puts pressure on the network to reconstruct the video using as few objects as possible (i.e. few objects with large values for *pres*). This pressure is necessary for the network to extract quality object-like representations; without it, the network is free to set all *pres* values to 1, and the extracted objects become meaningless. See Appendix A.2 for details.

For the propagation latent variables, we also include a learned causal prior $P_\theta(\overline{\mathcal{P}}_{(t)}|\mathcal{S}_{(t-1)})$. This amounts to learning a separate *Prior Propagation* module, which is similar in both role and architecture to the main Propagation module, except that it does not have access to the input frame. The Prior Propagation module is thus forced to learn to *predict* how objects will evolve based on their history (summarized in $\mathcal{S}^{\text{state}}_{(t-1)}$), whereas the main Propagation module has the easier job of *inferring* how objects have evolved based on the current frame $x_{(t)}$. The Prior Propagation is trained to match the output of main propagation, with Kullback-Leibler (KL) Divergence used as the loss. In our experiments, we show that this Prior Propagation module learns to make predictions about object trajectories multiple steps into the future.

5.2.8 Training

The Propagation, Discovery and Selection modules form the VAE inference network Q_ϕ, while the Rendering module forms the VAE generative network $P_\theta(x|z)$. The network as a whole is trained by maximizing the variational evidence lower bound (ELBO) [95]:

$$
\begin{aligned}
&\mathcal{L}(x_{(0:T-1)}, u_{(0:T-2)}, \theta, \phi) \\
&= \sum_{t=0}^{T-1} E_{\overline{\mathcal{P}}_{(0:t-1)}, \overline{\mathcal{D}}_{(0:t-1)} \sim Q_\phi} \Big[\\
&\qquad \int Q_\phi(\overline{\mathcal{D}}_{(t)}|x_{(t)}, \mathcal{P}_{(t)}) Q_\phi(\overline{\mathcal{P}}_{(t)}|x_{(t)}, \mathcal{S}_{(t-1)}) \log P_\theta(x_{(t)}|\mathcal{S}_{(t)}) \; d\overline{\mathcal{P}}_{(t)} \; d\overline{\mathcal{D}}_{(t)} \quad \text{(reconstruction)} \\
&\qquad + D_{KL}(Q_\phi(\overline{\mathcal{P}}_{(t)}|x_{(t)}, \mathcal{S}_{(t-1)}) \parallel P(\overline{\mathcal{P}}_{(t)})) \quad \text{(Prop KL with fixed prior)} \\
&\qquad + D_{KL}(Q_\phi(\overline{\mathcal{P}}_{(t)}|x_{(t)}, \mathcal{S}_{(t-1)}) \parallel P_\theta(\overline{\mathcal{P}}_{(t)}|\mathcal{S}_{(t-1)})) \quad \text{(Prop KL with learned prior)} \\
&\qquad + D_{KL}(Q_\phi(\overline{\mathcal{D}}_{(t)}|x_{(t)}, \mathcal{P}_{(t)}) \parallel P(\overline{\mathcal{D}}_{(t)})) \quad \text{(Disc KL with fixed prior)} \\
&\qquad \Big] ,
\end{aligned}
$$

via the ADAM [94] variant of Stochastic Gradient Ascent.

We also make use of two additional tricks during training; these help deal with the extra complexity introduced by training on videos, as well as the need to maintain a balance between the Discovery and Propagation modules.

Curriculum Learning. Following SQAIR [98], we train the network with a form of curriculum learning. We begin by training on only the first 2 frames of each video. We then increase the number of training frames by 2 every N_{curric} update steps. After $(\lceil T/2 \rceil - 1) N_{\text{curric}}$ update steps the network will be training on complete videos. This was observed to help with training stability, possibly because it allows the network to get a solid understanding of object appearances before addressing temporally extended object dynamics.

Discovery Dropout. Early in development we found the network often tried to predict objects for new frames exclusively by way of the Discovery module rather than propagating objects from the previous frame (this reduces to running a SPAIR network on each frame independently). This strategy can yield reasonable reconstruction performance, but will

fail at object tracking since object identities are not maintained from one frame to the next. Moreover, this is a local minimum; once the network starts relying heavily on Discovery, Propagation stops being trained and cannot improve. To discourage this behavior, we designed a technique that we call *discovery dropout*. Each timestep other than $t = 0$, the entire Discovery module is turned off with probability p_{dd}. This forces the network to do as much as possible through Propagation rather than Discovery, since the network is never sure whether the Discovery module will be turned on for the next timestep. Throughout this chapter we use $p_{\text{dd}} = 0.5$.

5.3 Experiments

We tested SILOT in a number of challenging object discovery and tracking tasks, emphasizing large videos containing many objects. Code for running these experiments is available online[1]. To supplement the quantitative results provided in this section, we also provide a small set of qualitative results. Additional qualitative results, in the form of videos depicting the performance of trained SILOT networks, may be found online[2]. Full experiment details, including neural architectures, training schedule, and procedures for hyperparameter selection, can be found in Appendix C.

5.3.1 Metrics

We use 3 metrics to assess model performance: Multi-Object Tracking Accuracy (MOTA) (with IoU threshold 0.5), a standard measure of object tracking performance [119]; Average Precision (AP) (with IoU=0.1:0.1:0.9), a standard measure of object detection performance [44]; and Count Abs. Error, or the absolute difference between the number of objects predicted by the model and the ground-truth number of objects (on a frame-by-frame basis).

[1] `https://github.com/e2crawfo/silot`
[2] `https://sites.google.com/view/silot`

5.3.2 Baseline Algorithm: ConnComp

We compare against a simple baseline algorithm called ConnComp (Connected Components), which works by treating each connected region of similarly-colored pixels in a frame as a separate object, and computing object identities over time using the Hungarian algorithm [101]. The performance of ConnComp can be interpreted as a measure of the difficulty of the dataset; it will be successful only to the extent that objects can be segmented and tracked by color alone. A more detailed discussion of ConnComp can be found in Appendix C.1.4.

5.3.3 Scattered MNIST

In this experiment, each 8-frame video is generated by first selecting a number of MNIST digits to include and sampling that number of digits. Videos have spatial size 48×48 pixels, digits are scaled down to 14×14 pixels, and initial digit positions are randomly chosen so that the maximum overlap allowed is small but non-trivial. For each digit we also uniformly sample an initial velocity vector with a fixed magnitude of 2 pixels per frame. Digits pass through one another without event (which often results in large amounts of inter-object occlusion), and bounce off the edges of the frames.

In order to test generalization ability, we used two training conditions: training on videos containing 1–6 digits, and training on videos containing 1–12 digits. In both cases, we test performance on videos containing up to 12 digits; networks trained in the 1–6 condition will thus be required to generalize beyond their training experience.

We compare SILOT against SQAIR. We tested SQAIR with two different Discovery backbone networks (the network which first processes each input frame): an MLP, and a convolutional network with the same structure as SILOT's convolutional backbone d_ϕ^{bu}. Note that using a convolutional network as a backbone, rather than an MLP, should improve SQAIR's spatial invariance to a degree; however it will still lack, among other features, SILOT's local object specification scheme, which is part of what allows SILOT's

Discovery module to be interpreted as an array of local discovery units. A similar issue was discussed in Section 4.5.1 in the context of SPAIR and AIR.

We also experimented with Tracking by Animation (TbA) [70], but were unable to obtain good tracking performance on these densely populated videos. One relevant point is that TbA lacks a means of encouraging the network to explain scenes using few objects, and we found TbA often using several internal objects to explain a single object in the video; in contrast, both SILOT and SQAIR use prior distributions which encourage the *pres* attributes to be near 0, thereby forcing the networks to use internal objects efficiently.

Results are shown in Figure 5.4. We have omitted both SQAIR networks that were trained on the 1–12 digit condition; SQAIR's simpler Discovery module was unable to handle the more densely packed scenes, resulting in highly varied performance that would make the plot unreadable. From the plot we can see that SILOT outperforms SQAIR by a large margin (especially when extrapolating beyond training experience), and that SILOT trained on 1–6 digits suffers only a minor performance hit compared to SILOT trained on 1–12 digits, a result of SILOT's spatially invariant architecture.

Note the slight improvement in performance of the AP of the SQAIR networks when going from 1 to 2 digits. This reflects an interesting strategy developed by the SQAIR networks. On the first timestep, they often turn "on" several more objects than are in the video; this pushes down the AP score, with a greater negative impact when there are fewer objects. We hypothesize that it settles on this strategy because the backbone networks are unable to get an accurate read on the number of objects present, and so the network settles for a strategy of casting a "wide net" on the first timestep.

Additionally, when testing on videos containing 1–6 digits, the SQAIR networks were used in the same fashion as they were during training. However, when testing on videos containing 7 or more digits, we found that the network's performance at guessing the number of objects to use became very poor; thus, for such test cases we artificially told the network the correct number of objects to use by forcing SQAIR's recurrent network to

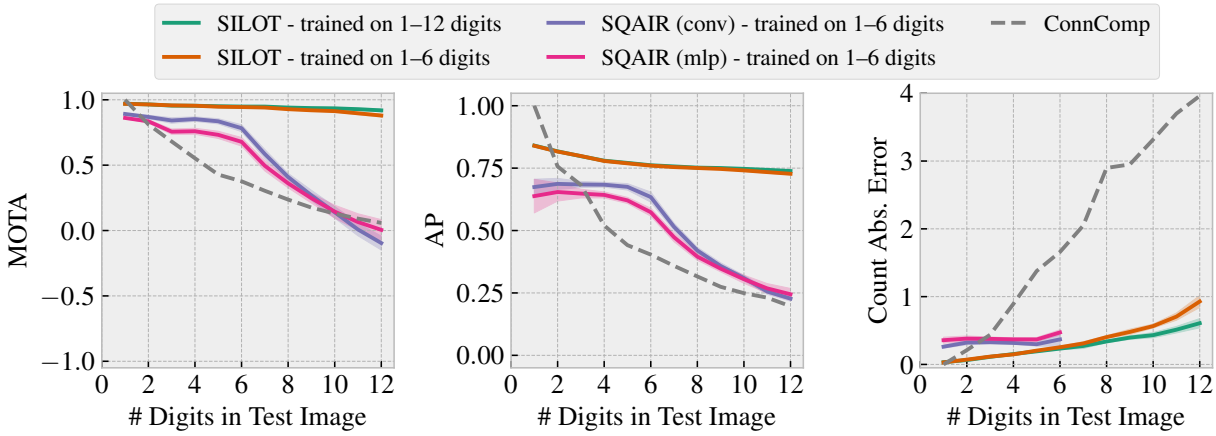

Figure 5.4. Probing object discovery and tracking performance as number of digits per video varies in the Scattered MNIST task. All points are averages over 6 random seeds, filled regions are 95% confidence intervals for the mean (except for the fully deterministic ConnComp algorithm).

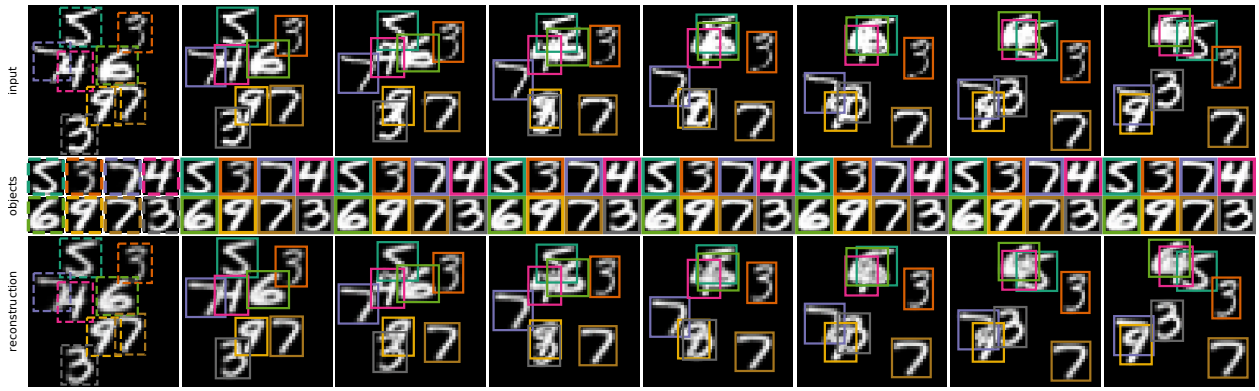

Figure 5.5. Visualizing a forward pass of a trained SILOT network applied to a video from the Scattered MNIST task containing 8 objects. Top / Bottom: Ground truth / reconstructed frames with bounding boxes for detected objects overlaid. Middle: Predicted appearances for detected objects. Box color represents object identity according to the network. Boxes for objects that SILOT has discovered in a given frame are dashed, while boxes for objects propagated from the previous frame are solid. Notice that the network is able to track objects even after they have passed completely through other objects (e.g. 5 with the green box, 3 with the grey box).

iterate for the correct number of steps. This is why the Count Abs. Error above 6 digits is omitted for the SQAIR networks.

Scattered MNIST - Prior Propagation

As detailed in Section 5.2.7, we trained a learned prior for the Propagation latent variables in addition to the fixed prior. This learned prior can be viewed as a duplicate of the main

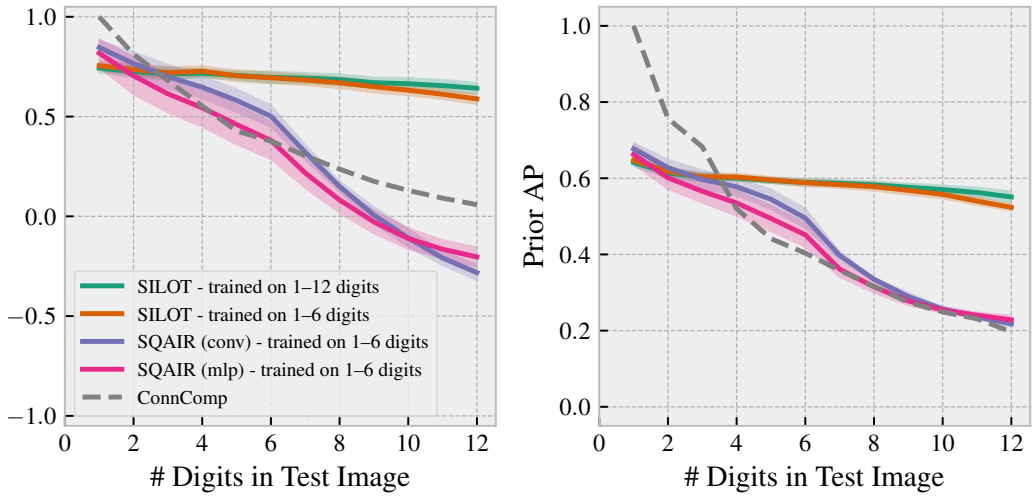

Figure 5.6. Probing ability of the learned Prior Propagation modules to predict object trajectories without access to the input frames. All points are averages over 6 random seeds, filled regions are 95% confidence intervals for the mean (except for the fully deterministic ConnComp algorithm).

Propagation module, except that it does not have access to the frame each time step. Here we test the performance of this *Prior Propagation* module in the Scattered MNIST task.

The evaluation procedure is similar to the one used in SQAIR for the same purpose [98], and runs as follows. For the first 3 timesteps, the regular network is used (with the regular Propagation module), in order to discover objects and estimate their initial trajectory. For the remaining 5 frames the Discovery module is deactivated, and the Prior Propagation module is used instead of the main Propagation module. Evaluation metrics were computed only on the final 5 frames. We are thus testing the ability of the Prior Propagation module to predict the trajectories of the objects detected in the first 3 frames by the main network. Results for both SILOT and SQAIR are shown in Figure 5.6.

5.3.4 Scattered Shapes

In this experiment we tackle larger images with significantly more objects to track. Here each video has spatial size 96×96, and contains a number of moving monochrome shapes. We use 6 different colors and 5 different shapes. Initial shape velocities are uniformly sampled with a fixed magnitude of 5 pixels per frame.

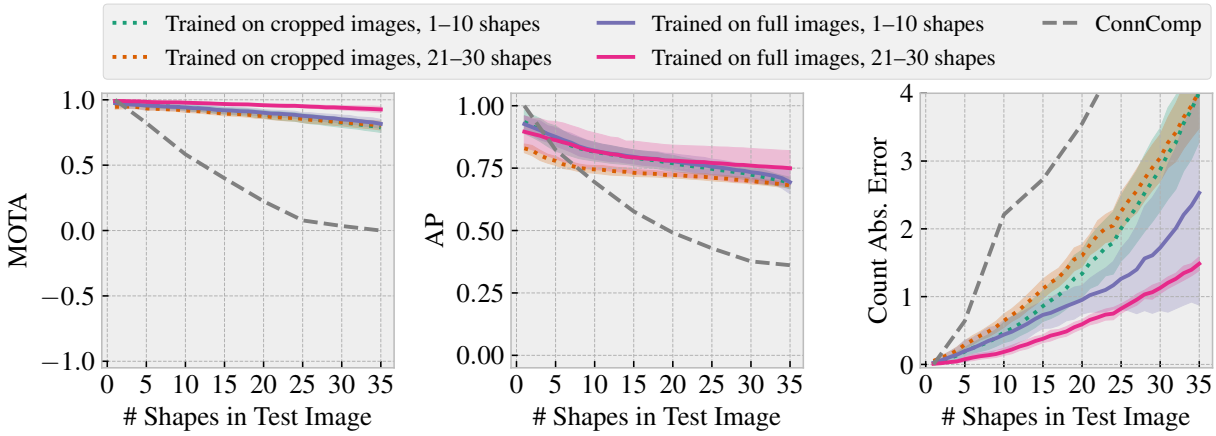

Figure 5.7. Probing SILOT's object tracking performance as the number of shapes per video varies in the Scattered Shapes task. All points are averages over 4 random seeds, and filled regions are 95% confidence intervals for the mean (except for the fully deterministic ConnComp algorithm). We manipulated 2 different aspects of the training setup: training on images that contain 1–10 shapes vs 21–30 shapes, and training on random 60×60 crops vs full 96×96 images (full videos are always used at test time).

Here we test SILOT's ability to generalize to scenes that are larger and/or contain different numbers of objects than scenes encountered during training. To assess the former, networks were trained on either random 60×60 crops of input videos, or full 96×96 videos (note that SILOT's fully convolutional Discovery module allows it to process videos of any size). To assess the latter, networks were trained on videos containing either 1–10 shapes or 21–30 shapes. This yields 4 different training conditions. At test time, we use full 96×96 videos containing up to 35 shapes.

Results are shown in Figure 5.7. There we see that all SILOT networks drastically outperform the baseline. Additionally, there is significant evidence that the networks are able to generalize well to both different numbers of shapes and different sized videos than what was encountered during training. In particular, the network trained on 60×60 crops of videos containing 1–10 shapes achieves reasonable performance when asked to process 96×96 images containing up to 35 shapes.

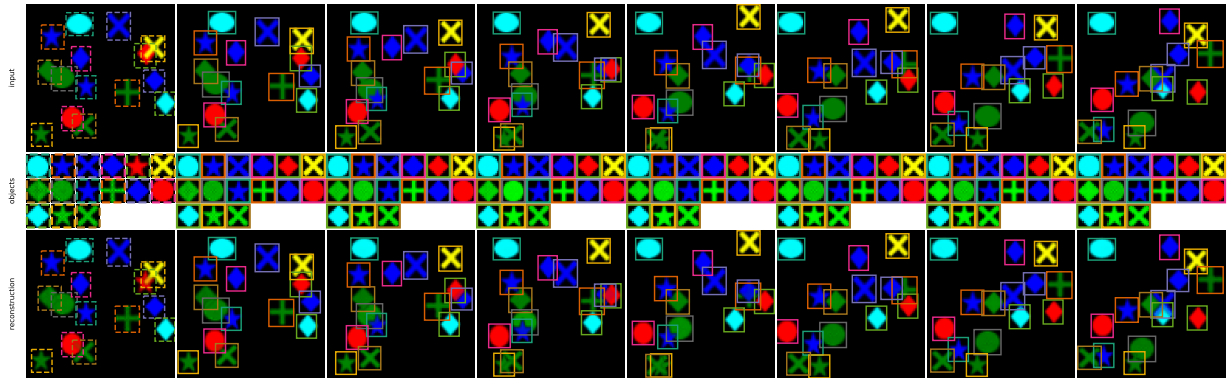

Figure 5.8. Visualizing a forward pass of a trained SILOT network applied to a video from the Scattered Shapes task containing 15 objects. Top / Bottom: Ground truth / reconstructed frames with bounding boxes for detected objects overlaid. Middle: Predicted appearances for detected objects. Box color represents object identity according to the network. Boxes for objects that SILOT has discovered in a given frame are dashed, while boxes for objects propagated from the previous frame are solid. Notice that the network is able to track objects even after they have been heavily occluded by other objects (e.g. green cross with the orange box that starts near the center).

Game	MOTA	AP	Count Abs. Error
Space Invaders	.89	.73	2.94
Asteroids	.67	.67	1.81

Table 5.1. SILOT performance on Atari videos.

5.3.5 Atari

We also applied SILOT to videos obtained from select Atari games, namely Space Invaders and Asteroids [10] (the possibility of applying a SQAIR-style model to reinforcement learning was first suggested in [98]). Models were trained on random crops of size $72\text{pixels} \times 72\text{pixels}$ with $K = 36$ propagated/selected objects, but evaluated on full frames of size $195\text{pixels} \times 160\text{pixels}$ using $K = 128$ propagated/selected objects. Results are shown in Table 5.1. Ground-truth object labels for evaluation were obtained using the ConnComp algorithm (used as a baseline in the Scattered MNIST dataset), which is reasonably effective for the chosen games as the objects therein can generally be segmented by color. Our goal here was to push scalability, rather than ability to deal with e.g. overlapping objects.

5.4 Discussion

In this chapter we proposed SILOT, an architecture for scalable unsupervised object discovery and tracking which improves upon past architectures like SQAIR by incorporating insights from recent supervised object detection architectures. In a number of experiments, we empirically showed a number of the advantages of SILOT. First, we showed that SILOT is significantly better than SQAIR at discovering and tracking objects in densely crowded, many-object scenes. We also showed that SILOT's Prior Propagation module is able to predict object trajectories in these crowded scenes. Next, we showed that SILOT's focus on spatial invariance permits it to effectively discover and track objects in scenes that are both spatially larger and/or densely crowded than scenes encountered during training. In particular, we showed that a network trained on cropped versions of videos containing at most 10 objects was able to generalize well to full (uncropped) videos containing 35 objects. Finally, we showed that SILOT scales well enough to be able to discover and track objects in two Atari games containing large numbers of objects. This last feat could eventually open up the possibility of making use of OOWMs for discovering and maintaining object-oriented representations for use in reinforcement learning, as suggested in [98].

Many interesting considerations and directions left for future research. For one, the current iteration of SILOT largely segments objects by optimizing for modularity (similar to SPAIR; see Section 4.7.3). However, motion is known to be a strong indicator of object identity: image regions that move together are likely to be part of the same object [146]. Some architectures have been designed which make explicit use of motion in this way [161]; however, such methods have difficulty dealing with objects that are always stationary. In future work it would be interesting to combine SILOT's modularity-based approach with these motion-based approaches to obtain the best of both worlds.

It should also be possible to extend the Discovery module to discover objects at multiple different scales, similar to the supervised object detection architecture SSD [111]. Additionally, in order to have SILOT and related approaches work on real videos, it will be

necessary to include a means of dealing with complex, dynamic backgrounds; past work in this area has generally preprocessed videos with an off-the-shelf background subtractor, largely avoiding the problem of modeling the background (though see concurrent work in [85] which makes significant progress on this issue). Finally, as suggested in [98], these architectures are quite complex, and it would be worthwhile to look for ways to simplify them. Much of the complexity stems from the need to be able to deal with different numbers of objects; for example, STOVE [99] is a descendant of SQAIR that is significantly simpler, but at the cost of requiring the user to specify the number of objects beforehand.

5.5 Retrospective

An architecture similar to SILOT, with a similar emphasis on scalability, was concurrently developed under the name SCALOR [85]. A follow-up work, Generative Structured Worlds Model (G-SWM), builds on both SILOT and SCALOR by introducing a hierarchy of per-timestep, per-object latent variables. The resulting model is significantly better than either of its predecessors at modeling the inherent stochasticity of videos, and is able to generate multiple coherent futures given a set of object histories [108].

Chapter 6

Unsupervised Object Detection and Tracking in 3D

6.1 Introduction

As we have seen, Object-Oriented World Models (OOWM) are temporal Variational Autoencoders (VAE) that couple object-oriented internal representations with structured neural networks for inference, dynamics and generation. OOWMs are often able to learn to decompose scenes into object-oriented representations, to coherently track objects over time and to predict future object trajectories, all without requiring object-level supervision at training time. The object-oriented representations of a trained OOWM have a number of advantages over unstructured representations, including being more interpretable to humans, supporting powerful patterns of reasoning and generalization, and being suitable for input to downstream symbolic algorithms. Thus far, the work presented in this thesis has largely addressed the question of building OOWMs that can handle spatially large, many-object images and videos . The present chapter is concerned with extending the applicability of OOWMs in an orthogonal direction, namely modeling 3D scenes.

One feature common to the majority of existing OOWMs (including SPAIR and SILOT, presented in the previous two chapters) is that they are fundamentally 2-dimensional;

objects are modeled as 2D entities existing on the 2D image plane. In the present chapter, we aim to go beyond this assumption, proposing an OOWM that explicitly models objects as 3D entities existing in a 3D world. OOWMs that embrace the 3D nature of the world and extract 3D information about objects have the potential to realize a number of advantages over their 2D predecessors. For instance, many downstream tasks rely fundamentally on such 3D information, including grasping objects, planning paths to retrieve collections of objects, etc. Moreover, representing objects as existing in a 3D world is necessary in order to properly model objects' future behavior (e.g. for use in model-based reinforcement learning), especially in the face of complex object-object and object-environment interactions. To take a very simple example, given two objects moving parallel to the camera plane and *apparently* towards one another, we cannot accurately predict whether they will collide unless we know their 3D shapes and relative depths.

One challenge in making the jump to 3D is that 3D scenes consist of both small/medium-sized dynamic objects (e.g. book, dog, human) and large, static scene elements (e.g. building, tree, mountain); it may be difficult to capture these disparate entities using a single kind of representation. In the current work we propose an OOWM which uses different representational formats for dynamic objects and static scene elements, ultimately learning structured representations of both kinds of entity. For modeling dynamic 3D objects we propose a novel approach, inspired by previous 2D OOWMs such as SQAIR [98], SILOT [29] and SCALOR [85], which can be thought of as a temporal VAE whose latent representation is a set of dynamic 3D objects. For modeling static 3D scene elements, we employ Scene Representation Networks [143], a recently proposed method for learning implicit representations of static 3D environments from posed videos.

Our proposed model, **3D O**bject-Oriented World **M**odel (3DOM), combines these two approaches, resulting in an OOWM capable of learning to extract structured, disentangled representations of both dynamic and static scene elements from posed videos of 3D scenes. Through a number of experiments, we demonstrate that 3DOM can learn to segment

objects from each other and from the static environment, can learn to track objects through 3D space, and makes progress towards accurately predicting objects' future trajectories.

6.2 Related Work

The vast majority of past work on OOWMs can be characterized as either scene-mixture models (representing objects as image-sized segmentation maps) [16, 40, 59, 60, 158, 160] or spatial attention models (representing objects as bounding boxes) [28, 29, 42, 70, 85, 98, 162]; some works even use both kinds of representation [109, 169], exploiting their complementary strengths to handle a wider range of scenes. However, one feature common to all these models is that they effectively treat objects as 2D entities living on the 2D image plane. While a few of them do attempt to infer the relative depth of objects [28, 160], this is only used to determine z-order when rendering the objects to an output image, and does not constitute a true 3D representation; for example, these models do not have a notion of a camera, and would therefore struggle if the camera were to shift or rotate.

A number of very recent works have made progress on designing OOWMs with true 3D object representations. ROOTS [24] is an OOWM which discovers 3D objects in static 3D scenes; it lays a 3D grid over the scene, and allows an object slot for each cell of the grid. However, ROOTS does not allow for moving objects, does not employ a structured 3D representation of the non-object elements of a scene, and may have difficulty handling large scenes due scaling issues associated with the use of a 3D grid. Another model, O3V, makes similar use of a 3D grid but is able to handle moving objects [72]. One downside of O3V is that it is not suitable for use in an online setting (such as model-based reinforcement learning [65, 66, 67, 90]), as it does not provide a means of updating its object representations as new video frames become available. 3DOM, in contrast, is well-suited to online operation as it has an inference network that is able to update object representations from previous timesteps based on information in a new frame. BlockGAN is another related model which learns to generate scenes made up of 3D objects, and does so without

supervision [121]; however, it only accounts for scene *generation*, having no capacity for learning to detect objects from perceptual input (i.e. inference).

Several models have also been proposed which have similar goals to 3DOM but make use of object-level supervision to pre-train parts of the network. One example is POD-Net [37], which formulates a probabilistic model similar to MONet [16], equipped with additional machinery for inferring 3D bounding box representations of objects; POD-Net requires object-level supervision to pre-train several components, including components that map from 2D to 3D and vice versa. Another model in this category requires access to ground-truth depth information and relies on a pre-training step in which the network learns to model individual objects in isolation [39]. In contrast to these approaches, 3DOM is able to learn without any form of object-level supervision, instead relying on a set of reasonable assumptions about the structure of the environment that generates the training data, outlined in the next section.

6.3 Problem Definition

Discovery of objects in free form environments is a difficult task; indeed, it is difficult to even specify what constitutes an object [57]. To make progress on this issue, in this paper we assume that the environment has a particular structure. We assume a setting in which an agent episodically interacts with a fixed collection of static 3D scenes; that each scene is visited multiple times; that the scenes are populated with different configurations of objects on each visit; and that each episode is marked with an index indicating which static scene it came from. As an example, the different static scenes could be different houses one has visited; upon successive visits to a particular house, objects inside the house are likely to have changed and/or been rearranged.

More concretely, let $\{S_1, \ldots, S_N\}$ be a set of N static 3D scenes. At the start of each interaction episode, a scene index $n \in \{1, \ldots, N\}$ is sampled uniformly at random, yielding static scene S_n. The sampled scene is then populated with dynamic elements, specifically

a camera and a set of objects. For timestep t, let $C_{(t)}$ be the camera pose, and let $O_{(t)}$ be the set of dynamic objects. At the beginning of the episode, we sample an initial set of dynamic elements $C_{(0)}, O_{(0)} \sim P_0(C_{(0)}, O_{(0)}|S_n)$. These elements are then propagated forward in time by sampling $C_{(t)}, O_{(t)} \sim P(C_{(t)}, O_{(t)}|C_{(t-1)}, O_{(t-1)}, S_n)$. Each timestep, a pinhole camera model F is used to render an image: $I_{(t)} = F(C_{(t)}, O_{(t)}, S_n)$.

Initialization and dynamics distributions, P_0 and P, are assumed to be unknown, as are the static scene representations S_n. For each episode, the agent is given access only to the index n of the static scene used to generate the episode, as well as image $I_{(t)}$ and camera pose $C_{(t)}$ for each timestep. The training data for an episode is thus:

$$n, (I_{(0)}, C_{(0)}), (I_{(1)}, C_{(1)}), \ldots, (I_{(T-1)}, C_{(T-1)}) \,. \tag{6.1}$$

Note that the learner knows when two episodes come from the same static scene (they have the same value for n), but is not provided with any information on the contents of the scene other than what can be inferred from the images.

The fixed set of static 3D environments for training may together be regarded as a "nursery" environment in which the agent can discover objects and their properties, in preparation for operating in more complex environments in the future. In our experiments, we demonstrate that models trained in these nursery environments are later able to perform well in new, unseen environments. Note that many common environments in reinforcement learning have all episodes coming from a single static environment, and can therefore be viewed as instances of our setting with $N = 1$.

Also note that since we allow multiple static scenes, each of which may be arbitrarily complex, our setting allows environments that are significantly more difficult than many settings addressed by recent 2D OOWMs, which have often used simple, fixed backgrounds common to all images. For example, in the commonly used CLEVR dataset [86], all images take place on a flat gray surface.

Finally, while the assumption of known camera pose may seem restrictive, in many cases it should be possible to estimate camera poses directly from the videos using Structure from Motion pipelines such as COLMAP [139].

In order to learn disentangled, structured representations of the type of environment just described, we propose a novel approach that we call **3D O**bject-Oriented World **M**odels (3DOM). 3DOM consists of two components: one that learns to model large, static scene elements such as walls and structures, and another that learns to model dynamic small/medium-sized objects.

6.4 Learning to Model Static Scene Elements

For modeling static elements of a scene we employ the framework of Scene Representation Networks (SRN) [143]. Under this framework, a 3D scene is modeled as a differentiable, parameterized function $\Phi\colon \mathbb{R}^3 \rightarrow \mathbb{R}^f$ which maps points in 3D space to feature vectors. These feature vectors are taken to represent information about the scene at the given 3D location (e.g. occupancy, color, material).

The SRN framework provides a differentiable raymarching algorithm, denoted here as $\mathrm{Raymarch}$. This algorithm can be used to render an image R depicting the scene represented by SRN Φ from the viewpoint of a camera pose C:

$$R = \mathrm{Raymarch}(C, \Phi) \; .$$

Here we describe how this raymarching algorithm renders a single pixel of image R, and an accompanying schematic is shown in Figure 6.1. Applying this procedure to all pixels (in parallel) yields the rendered image R. Let i be the pixel index, r_i be the unit ray in 3D space emanating from the camera pinhole outward through the pixel's 3D position (both points are determined by C), and let $x_0 = \epsilon r_i$ be the initial marching position, where ϵ is a small random value. We then enter a raymarching loop where, for the m-th iteration,

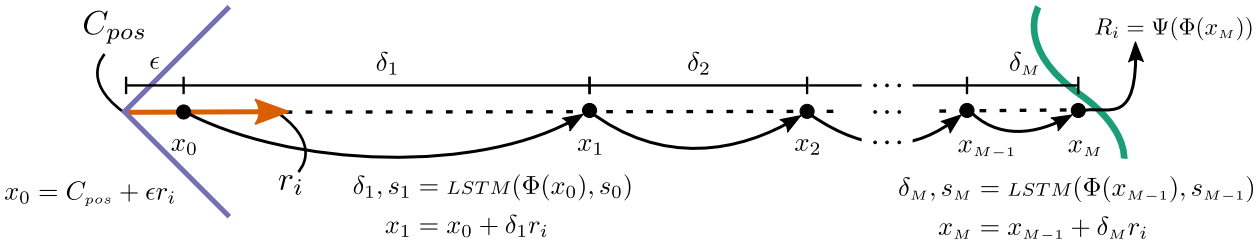

Figure 6.1. Illustration of the differentiable raymarching algorithm used for rendering a scene represented by an SRN Φ given a camera pose C [143]. The algorithm is shown here for a single pixel with index i. The blue triangle represents the camera, the orange arrow represents the unit ray r_i extending from the camera pinhole out through pixel i, and the green curve represents a scene element which Φ has learned to represent and which is currently being rendered. The LSTM step-size predictor predicts smaller and smaller steps as it approaches the scene element, and the final sampled location x_M is near the surface of the scene element.

we perform the following steps: evaluate $\Phi(x_{m-1})$ to get the scene features at position x_{m-1}, pass the result into an LSTM [75] (with a separate state for each pixel) to predict a step length δ_m, and finally update the marching position $x_m = x_{m-1} + \delta_m r_i$. After M iterations, the final feature vector $\Phi(x_M)$ is passed into a pixel generator network $\Psi\colon \mathbb{R}^f \to \mathbb{R}^3$ which predicts an RGB value R_i for the pixel, and $D_i = \epsilon + \sum_{m=1}^{M} \delta_m$ may be taken as a depth prediction.

Importantly, the algorithm is fully differentiable, so we can train the weights of the SRN Φ, the LSTM, and the pixel generator Ψ by minimizing error between a ground-truth image I and the rendered image R using gradient descent. If we train with a diverse dataset of pose-image pairs $\{(C_j, I_j)\}_j$, then we can expect Φ to become an accurate representation of the 3D scene.

Within 3DOM, we make use of the SRN framework as follows. First, we instantiate a hypernetwork H (a neural network whose output is another neural network [63]) which maps from a scene index n to an SRN Φ_n:

$$\Phi_n = H(n) \,,$$

as proposed in [143]. Then, given data from a training episode of the form given in (6.1), we train Φ_n to reproduce image $I_{(t)}$ from camera pose $C_{(t)}$. Letting $R_{(t)} = \text{Raymarch}(C_{(t)}, \Phi_n)$,

we use gradient descent to minimize $\sum_{t=0}^{T-1} \mathcal{L}(R_{(t)}, I_{(t)})$, where $\mathcal{L}$ is some image loss. Since Φ_n is produced by the hypernetwork H, here we are actually training the weights of H rather than Φ_n itself.

One notable feature of SRNs is that they lack any means of representing dynamic or contingent scene elements (e.g. objects that move or change appearance over time, or even static objects which are not present every episode). Notice, in particular, that the functions Φ_n only take a spatial coordinate as input; the features assigned by the network to a given 3D location in a given scene are not permitted to vary with time or episode number. Consequently the SRN will only be able to account for static scene elements, while incurring large error at pixels belonging to dynamic objects. Augmenting SRNs with a system capable of handling these dynamic scene elements is the subject of the next section.

6.5 Learning to Model Dynamic Objects

We designed a novel neural network to address the task of discovering, tracking and forecasting dynamic objects. This network can be viewed as an OOWM: a temporal VAE [95] with an object-like latent representation. Its structure takes inspiration from past OOWMs, including SQAIR [98], SILOT [29] and SCALOR [85]. However, whereas those models conceive of objects as 2D bounding boxes on a 2D plane, our proposed network models objects as 3D entities moving through a 3D world.

6.5.1 3D Object Representation

The primary data type in 3DOM is a set of objects, each set having space for a fixed number of object slots. Each object set contains a collection of named variables, called attributes, each of which can be indexed by slot k. We distinguish between two kinds of attributes: world-frame attributes, defined with respect to the world coordinate frame, and camera-frame attributes, defined with respect to a particular camera pose.

For a generic object set $\mathcal{O}_{(t)}$ defined for time t, the world-frame attributes for object k are:

$$\mathcal{O}^{\text{where-3D}}_{(t),k} \in \mathbb{R}^3 \,, \qquad \mathcal{O}^{\text{what}}_{(t),k} \in \mathbb{R}^{N_{\text{what}}} \,,$$
$$\mathcal{O}^{\text{pres}}_{(t),k} \in [0,1] \,, \qquad \mathcal{O}^{\text{state}}_{(t),k} \in \mathbb{R}^{N_{\text{state}}} \,.$$

$\mathcal{O}^{\text{where-3D}}_{(t),k}$ gives the 3D position of the object, $\mathcal{O}^{\text{what}}_{(t),k}$ is an unstructured vector capturing object features such as appearance and shape, $\mathcal{O}^{\text{pres}}_{(t),k}$ specifies the extent to which the object exists, and $\mathcal{O}^{\text{state}}_{(t),k}$ is the hidden state of a recurrent neural network capturing the object's history.

The camera-frame attributes, defined with respect to camera pose $C_{(t)}$, are:

$$\mathcal{O}^{\text{bbox-2D}}_{(t),k} \in \mathbb{R}^4 \,, \qquad \mathcal{O}^{\text{depth}}_{(t),k} \in \mathbb{R} \,, \qquad \mathcal{O}^{\text{visible}}_{(t),k} \in [0,1] \,.$$

We define $\mathcal{O}^{\text{bbox-2D}}_{(t),k} = (\mathcal{O}^{y}_{(t),k},\ \mathcal{O}^{x}_{(t),k},\ \mathcal{O}^{h}_{(t),k},\ \mathcal{O}^{w}_{(t),k})$; $(\mathcal{O}^{y}_{(t),k},\ \mathcal{O}^{x}_{(t),k})$ gives the location of the object's center on the camera plane, while $(\mathcal{O}^{h}_{(t),k},\ \mathcal{O}^{w}_{(t),k})$ gives the object's apparent size. $\mathcal{O}^{\text{depth}}_{(t),k}$ specifies the straight-line distance to the object from the camera pinhole. $\mathcal{O}^{\text{visible}}_{(t),k}$ specifies whether the object is visible from the perspective of $C_{(t)}$. Defining $\mathcal{O}^{\text{visible}}_{(t),k}$ separately from $\mathcal{O}^{\text{pres}}_{(t),k}$ allows the network to represent objects that it knows exist but which cannot currently be seen (e.g. occluded or off camera).

As with SILOT, each of these object attributes (with the exception of $\mathcal{O}^{\text{state}}_{(t),k}$) has an underlying *latent* variable. Each object variable is a deterministic function of its corresponding latent variable. We separate the object variables from the latent variables as it is useful to be able to define latent variables in a space other than the highly structured space of objects. We denote the set of latent variables corresponding to an object set by adding an overbar to the name of the object set, so that $\overline{\mathcal{O}}_{(t)}$ is the set of latent variables for object set $\mathcal{O}_{(t)}$.

6.5.2 Overview

3DOM is composed of a number of modules, each running once per timestep of the input video. Modules communicate by passing around object sets. We define three primary object sets:

1. Discovered objects $\mathcal{D}_{(t)}$ (and underlying latent variables $\overline{\mathcal{D}}_{(t)}$).
2. Propagated objects $\mathcal{P}_{(t)}$ (and underlying latent variables $\overline{\mathcal{P}}_{(t)}$).
3. Selected objects $\mathcal{S}_{(t)}$.

The flow of computation in 3DOM is depicted in Figure 6.2, and runs as follows:

1. **Discovery** takes in input frame $I_{(t)}$ and predicts a set of 3D objects for the frame, first yielding $\overline{\mathcal{D}}_{(t)}$ and then $\mathcal{D}_{(t)}$ via a deterministic transformation.
2. **Propagation** takes in selected objects from the previous timestep $\mathcal{S}_{(t-1)}$ and predicts a set of latent attribute updates $\overline{\mathcal{P}}_{(t)}$, conditioning on the input frame $I_{(t)}$ and camera pose $C_{(t)}$. These updates are then deterministically applied to $\mathcal{S}_{(t-1)}$, yielding $\mathcal{P}_{(t)}$.
3. **Selection** takes in $\mathcal{P}_{(t)}$ and $\mathcal{D}_{(t)}$, and first applies a *suppression* network that removes duplicate objects. It then selects a subset of objects with highest value of the *pres* attribute, yielding selected objects $\mathcal{S}_{(t)}$.
4. **Rendering** takes in selected objects $\mathcal{S}_{(t)}$, differentiably renders them, and blends them with image $R_{(t)}$ rendered by the SRN, yielding final output image $\widehat{I}_{(t)}$.

We now describe in detail how these modules are implemented.

6.5.3 Discovery

The Discovery module, responsible for detecting objects in each frame; each timestep t, it takes in the input frame $I_{(t)}$ and outputs a set of discovered objects $\mathcal{D}_{(t)}$. Discovery in 3DOM is similar in structure to SPAIR's encoder network, equipped with additional

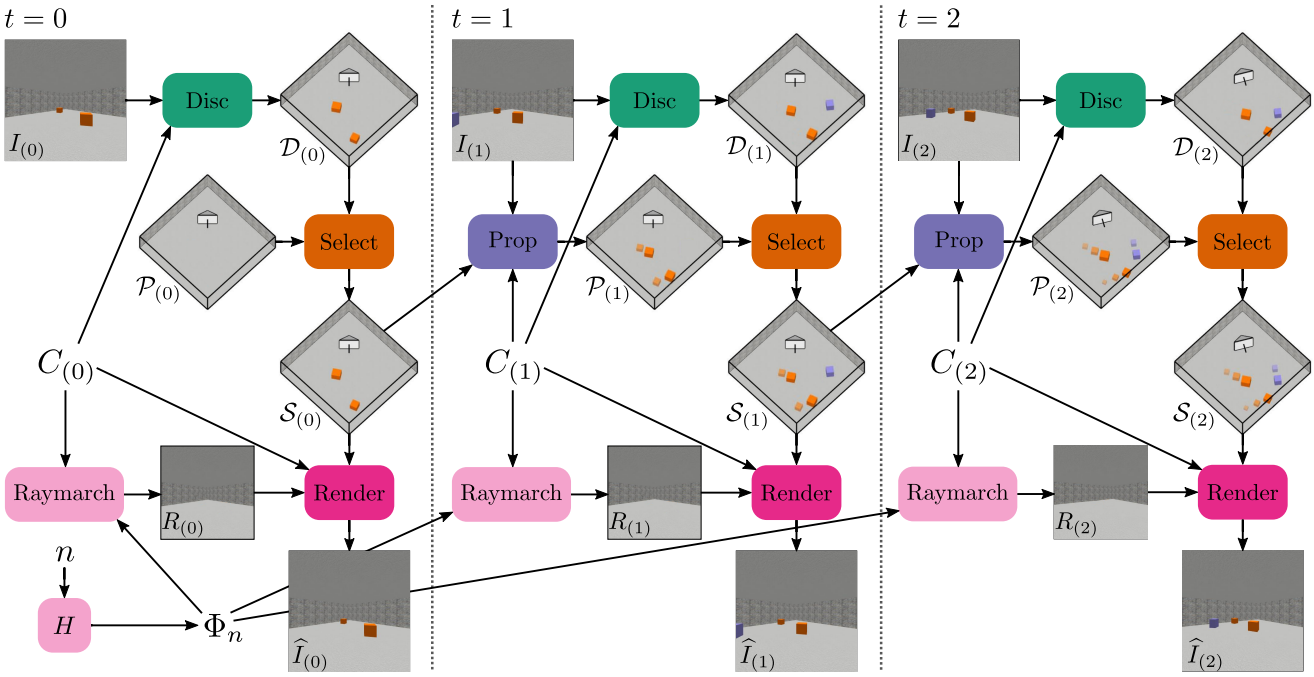

Figure 6.2. High-level overview of 3DOM. Each gray diamond represents a set of 3D internal objects. Within the gray diamonds, the white pyramid represents the known camera pose and the colored blocks are objects tracked by the agent. The translucent object "tails" are meant to indicate where the objects were previously estimated to be, and correspond to perceived object motion.

machinery for predicting the 3D location of objects. Thus, here we mainly give a high-level sketch of its operation, leaving details to Section 4.4.2, though we do go into detail on the aspects that are unique to 3DOM.

Discovery begins by applying a convolutional backbone network:

$$v_{(t)} = d_{\phi}^{\text{backbone}}(I_{(t)}) \,,$$

where $v_{(t)}$ is a convolutional volume with shape (H, W, C). The structure of $d_{\phi}^{\text{backbone}}$ can be interpreted as inducing a grid of cells laid over the input image; if c_h/c_w are the product of the height/width strides (respectively) in d^{backbone}, then each cell has size (c_h, c_w) (in pixels). Moreover, if the input image has shape $(H^{\text{img}}, W^{\text{img}}, 3)$, then $(H, W) = (\lceil H^{\text{img}}/c_h \rceil, \lceil W^{\text{img}}/c_w \rceil)$. Each spatial location (i, j) in $v_{(t)}$, for $i \in \{1, \ldots, H\}$, $j \in \{1, \ldots, W\}$, is associated with the (i, j)-th cell in the grid. Care is taken to pad the image so that the receptive field of each spatial location is centered on its associated grid cell. The network

will predict a separate object for each spatial location. The combination of the (i, j)-th grid cell, (i, j)-th spatial location in $v_{(t)}$, and (i, j)-th object predictor together constitute a discovery unit, which can be thought of as a local object detector.

Keeping our focus narrowed to a single discovery unit with indices (i, j), object prediction begins with predicting the 2D bounding box $\mathcal{D}^{\text{bbox-2D}}_{(t),ij}$ using an MLP; this is done using a coordinate system which ensures that the object center is inside the (i, j)-th grid. Next, a glimpse is differentiably extracted from the input image at $\mathcal{D}^{\text{bbox-2D}}_{(t),ij}$ using a Spatial Transformer [83]. An object encoder network then processes this glimpse, yielding on object encoding v^{obj}_{ij}. Attributes $\mathcal{D}^{\text{what}}_{(t),ij}$ and $\mathcal{D}^{\text{pres}}_{(t),ij}$ are then predicted based on the object encoding. So far, all of this is familiar from SPAIR's encoder network; we now move on to components that are unique to 3DOM.

Predicting Depth

The first 3D attribute that we predict is *depth*. The depth attribute in 3DOM corresponds to a more sophisticated notion of depth than used in either SPAIR or SILOT. In those past models, the depth parameterized a differentiable mixture between objects at pixels where multiple objects overlap; objects with smaller depth received more weight in this mixture. In those cases, the absolute value of depth had no meaning, and all that mattered was the relative ordering of the objects; it may therefore be helpful to think of it as *pseudo-depth*. In contrast, depth in 3DOM has a more definite interpretation: it is the straight-line distance from the camera pinhole to the object center. Indeed, the depth prediction will subsequently be used to compute an estimate of the 3D position of the object.

For predicting depth, we need at least two pieces of information: the absolute size of the object in 3D space, which we assume can be inferred from the object encoding v^{obj}_{ij}, and

the *apparent* size of the object, which is captured by $(\mathcal{D}_{ij}^{h}, \mathcal{D}_{ij}^{w})$:

$$
\begin{aligned}
(\mu_{ij}^{\text{depth}}, \sigma_{ij}^{\text{depth}}) &= d_{\phi}^{\text{depth}}(v_{ij}^{\text{obj}}, \mathcal{D}_{ij}^{h}, \mathcal{D}_{ij}^{w}) \,, \\
z_{ij}^{\text{depth}} &\sim N(\mu_{ij}^{\text{depth}}, \sigma_{ij}^{\text{depth}}) \,, \\
\mathcal{D}_{ij}^{\text{depth}} &= d^{\text{min}} + (\text{sigmoid}(z_{ij}^{\text{depth}}))^2 \cdot (d^{\text{max}} - d^{\text{min}}) \,,
\end{aligned}
$$

where d^{min} and d^{max} are the minimum and maximum allowed depth, respectively.

Computing 3D Location

With the depth prediction in hand, we now have everything we need to predict a multivariate Normal distribution over $\mathcal{D}_{ij}^{\text{where-3D}}$, parameterized by mean $\mu_{ij}^{\text{where-3D}}$ and covariance matrix $\Sigma_{ij}^{\text{where-3D}}$. This is computed as a function of the distributions over $\mathcal{D}_{ij}^{y}$, $\mathcal{D}_{ij}^{x}$ and $\mathcal{D}_{ij}^{\text{depth}}$, in combination with camera pose $C_{(t)}$ and (assumed known) camera focal lengths f_y and f_x.

Computing the Mean. We start with the procedure for computing mean of the multivariate distribution, $\mu_{ij}^{\text{where-3D}}$, achieved by "lifting" the mean 2D position $(\mu_{ij}^{y}, \mu_{ij}^{x})$ into 3D space, using μ_{ij}^{depth} as the depth. First, we map the 2D object location from pixel space to image space[1]:

$$
\begin{aligned}
y_{\text{img}} &= 2 \cdot \mu_{ij}^{y} / H_{\text{img}} - 1 \,, \\
x_{\text{img}} &= 2 \cdot \mu_{ij}^{x} / W_{\text{img}} - 1 \,.
\end{aligned}
$$

[1]The space where $(-1, -1)$ is the top left corner of the image and $(1, 1)$ is the bottom right corner.

Next, we compute camera-space coordinates for the object:

$$
\begin{aligned}
z_{\text{cam}} &= \mu_{ij}^{\text{depth}} \Bigg/ \sqrt{\left(\frac{y_{\text{img}}}{f_y}\right)^2 + \left(\frac{x_{\text{img}}}{f_x}\right)^2 + 1^2} \ , \\
y_{\text{cam}} &= \frac{y_{\text{img}}}{f_y} z_{\text{cam}} \ , \\
x_{\text{cam}} &= \frac{x_{\text{img}}}{f_x} z_{\text{cam}} \ .
\end{aligned}
$$

Finally, we map the camera coordinates to world coordinates. Assuming $C_{(t)}$ is a 4×4 matrix that maps from camera coordinates to world coordinates (i.e. a "cam-to-world" matrix), we have:

$$
\mu_{ij}^{\text{where-3D}} = \begin{bmatrix} x_{\text{world}} & y_{\text{world}} & z_{\text{world}} & \cdot \end{bmatrix} = \begin{bmatrix} x_{\text{cam}} & y_{\text{cam}} & z_{\text{cam}} & 1 \end{bmatrix} C_{(t)}^T \ .
$$

Computing the Covariance Matrix. That gives us the mean of the distribution; all that is left is to compute the covariance matrix $\Sigma_{ij}^{\text{where-3D}}$. We specify $\Sigma_{ij}^{\text{where-3D}}$ by its eigenvectors and eigenvalues. The eigenvectors can be interpreted as the principal semi-axes of the 3D ellipsoids that will be level sets of the probability density function of $N(\mu_{ij}^{\text{where-3D}}, \Sigma_{ij}^{\text{where-3D}})$, and each eigenvalue gives the variance of the distribution along the corresponding eigenvector/semi-axis.

To start, note that predicting depth is inherently more uncertain than predicting 2D location on the camera plane, as the latter can to a large extent be "read off" of the image, whereas the former cannot. Consequently the majority of the uncertainty over 3D position will be along the depth direction; that is, along the ray that starts at the camera pinhole and points at the center of the object. Thus, letting r_{ij} be the unit ray extending from the camera pinhole to the object center, we take r_{ij} as one of the eigenvectors/principal semi-axes of $\Sigma_{ij}^{\text{where-3D}}$. For the other two eigenvectors, we take two randomly selected directions that are orthogonal to both r_{ij} and to one another. For the standard deviation along r_{ij} we use $\sigma_{ij}^{\text{depth}}$. For standard deviation in the other two directions, we take the mean of the standard deviations in camera coordinates induced by σ_{ij}^{x} and σ_{ij}^{y}.

Now let E_{ij} be a 3×3 matrix whose columns are these eigenvectors, and let $\Lambda^{1/2}$ be a diagonal matrix whose diagonal entries are the given standard deviations (assume in both cases that the depth-derived eigenvector and standard deviation come first). Then we can derive the covariance matrix in *camera* coordinates:

$$\begin{aligned}
\Sigma_{ij}^{\text{cam}} E_{ij} &= E_{ij}\Lambda \,, && \text{(definition of eigenvectors)} \\
\Sigma_{ij}^{\text{cam}} &= E_{ij}\Lambda \left(E_{ij}\right)^{-1} \,, && \text{(multiplication on right by } E_{ij}^{-1}\text{)} \\
\Sigma_{ij}^{\text{cam}} &= E_{ij}\Lambda \left(E_{ij}\right)^{T} \,, && \text{(orthogonality of } E_{ij}\text{)}
\end{aligned}$$

Finally, we need to obtain the covariance matrix in *world* coordinates. Let $C'_{(t)} = C_{(t)}[:3, :3]$ be the rotational (non-translational) part of $C_{(t)}$. Then we have:

$$\Sigma_{ij}^{\text{where-3D}} = C'_{(t)} \Sigma_{ij}^{\text{cam}} \left(C'_{(t)}\right)^{T} \,.$$

At last we have our distribution over 3D position in world coordinates, $N(\mu^{\text{where-3D}}, \Sigma^{\text{where-3D}})$. Rather than sampling from this to get a value for $\mathcal{D}_{(t),ij}^{\text{where-3D}}$, we can instead take our sampled values for $\mathcal{D}_{(t),ij}^{y}$, $\mathcal{D}_{(t),ij}^{x}$ and $\mathcal{D}_{(t),ij}^{\text{depth}}$ pass them through the same lifting process that we applied to the mean values.

Finalizing Objects

In predicting $\mathcal{D}_{ij}^{\text{visible}}$, one important note is that the Discovery module has no notion of an object's history over an episode; from its point of view, each object that it outputs is brand new. Thus, objects output by Discovery *must* be visible, as the module has no basis upon which to predict an object other than through visual information in the current frame. Thus we set $\mathcal{D}_{ij}^{\text{visible}} = 1$. We as final step, we update the hidden state:

$$\mathcal{D}_{(t),k}^{\text{state}} = \text{RNN}_{\phi}(\mathcal{D}_{(t),k}^{\text{where-3D}}, \mathcal{D}_{(t),k}^{\text{what}}, \mathcal{D}_{(t),k}^{\text{pres}}, s_0) \,,$$

where s_0 is a default initial state for the recurrent neural network (RNN).

Avoiding Object Rediscovery

In designing the Discovery module, one important consideration is a means of avoiding *object rediscovery*; that is, ensuring that Discovery does not try to account for objects that are already accounted for by some propagated object. In SILOT we solved this by equipping the discovery units with the capacity to perform spatial attention over propagated objects, thereby giving them a chance to turn themselves off (set $\mathcal{D}^{\text{pres}}_{(t),ij} = 0$) when an object that they would normally predict is already in the set of propagated objects (see Section 5.2.3).

This strategy worked reasonably well, but has a few downsides. First, it prevents the Discovery module from being parallelized across timesteps. When training on a fixed dataset, 3DOM operates in an offline mode where full videos are available to it[2]. In such cases, we may hope to parallelize the Discovery module *across timesteps*. However, in SILOT this kind of parallelization is prevented by the fact that the spatial attention creates a dependency on the previous timestep through the propagated objects. A second downside of the spatial attention approach is that the task of the Discovery module is rather complex; not only does it have to learn to detect objects, but has to learn special circumstances in which to *inhibit* detection.

Consequently, in 3DOM we omit this spatial attention step, and instead shift the burden of preventing object rediscovery to the Selection module. 3DOM's Selection module contains a learnable *suppression* step which examines pairs of propagated and discovered objects that overlap with one another, and suppresses the discovered object when it determines that it is accounting for the same object as the propagated object. Thus, rather than forcing Discovery to learn the complex task of when to inhibit detection based on what propagated objects are nearby, we instead allow Discovery to completely ignore the temporal context, thereby greatly simplifying its task and allowing it to be parallelized across timesteps.

[2] As opposed to an online/streaming setting where frames become available one at a time.

6.5.4 Propagation

In the Propagation module, we have a selected object from the previous timestep, $\mathcal{S}_{(t-1)}$, and our aim is to use information from the input frame $I_{(t)}$ to infer updated versions of those objects, $\mathcal{P}_{(t)}$. The Propagation module can be divided into 3 steps:

1. **Object Localization.** Locate the object in the new frame.
2. **Glimpse Extraction.** Extract high-resolution information about the object in the new frame.
3. **Attribute Update Prediction.** Predict and apply updates to the object attributes based on the extracted information.

Note that Propagation in 3DOM has many similarities to Propagation in SILOT; however, several changes have been made in order to adapt it to the 3D setting addressed by 3DOM. In what follows we narrow our focus to updating a single selected object with index k.

Object Localization

In the 2D setting addressed by SILOT, we could relocate an object just by looking in an area around the object's location from the previous timestep. In the 3D setting addressed by 3DOM, this simple strategy is not viable since the camera is permitted to move, so that the projection of the object onto the camera may occur at a very different location (in image space) than on the previous timestep. Indeed, the object may not be on camera at all, as rotational camera movements often cause objects to move out of view.

Thus our strategy for localizing the object of interest is:

1. Compute the location in the new frame $I_{(t)}$ where we would expect to find the object if it were stationary; we achieve this by projecting the previous 3D location $\mathcal{S}^{\text{where-3D}}_{(t-1),k}$ onto the new camera position $C_{(t)}$.
2. Extract a *localization glimpse* centered at the computed location, where the glimpse scale is proportional to the apparent size of the object from the previous timestep.

3. Process the glimpse using a convolutional neural network which conditions on a description of the object (so that it knows which object to look for).

4. Predict object location in a coordinate frame that is a function of the glimpse parameters.

More concretely, we perform the following steps. First we project the previous 3D location onto the current camera (this will be used as the center of the localization glimpse):

$$(y_k, x_k) = \text{Project}(\mathcal{S}^{\text{where-3D}}_{(t-1),k}, C_{(t)}) \; .$$

Next we compute a size for the localization glimpse:

$$h_k = \chi \mathcal{S}^{h}_{(t-1),k} \; , \qquad\qquad w_k = \chi \mathcal{S}^{w}_{(t-1),k} \; .$$

Choosing the size of the localization glimpse based on the apparent size from the previous timestep $(\mathcal{S}^{h}_{(t-1),k}, \mathcal{S}^{w}_{(t-1),k})$ rests on the assumption that the object's apparent size will not change significantly in a single timestep. Here $\chi \geq 1$ is a hyperparameter which implicitly encodes an assumption about the maximum distance that objects are able to move in a single timestep, relative to their apparent size. Throughout this chapter we use $\chi = 2$.

Next we use Spatial Transformer (ST) to extract the localization glimpse at bounding box (y_k, x_k, h_k, w_k):

$$g^{\text{loc}}_k = ST(I_{(t)}, (y_k, x_k, h_k, w_k)) \; .$$

We process this glimpse with a convolutional network p^{loc}_ϕ in order to locate the object within g^{loc}_k. Note, however, that in crowded scenes g^{loc}_k may contain additional objects besides the one we are interested in locating. Thus we need to inform p^{loc}_ϕ as to what kind of object we are looking for. This is done using FiLM [126], a type of conditional batch normalization which allows a convolutional network to modify its forward computation in a non-trivial way based on a context vector. As context vector, we use the previous

hidden state $\mathcal{S}^{\text{state}}_{(t-1),k}$:

$$v_k = p^{\text{loc}}_\phi(g^{\text{loc}}_k, \mathcal{S}^{\text{state}}_{(t-1),k}) \; .$$

p^{loc}_ϕ also makes use of CoordConv, a technique which has been shown to improve the ability of convolutional networks to locate objects in images [110]; it amounts to concatenating (channel-wise) a mesh-grid positional encoding to g^{loc}_k before passing it into p^{loc}_ϕ.

Finally, we predict the location of the object based on the information contained in v_k. A network $p^{\text{bbox-2D}}_\phi$ predicts parameters for a Normal distribution over latent variables $\overline{\mathcal{P}}^{\text{bbox-2D}}_{(t),k}$ which will ultimately determine $\mathcal{P}^{\text{bbox-2D}}_{(t),k}$:

$$\begin{aligned}(\mu^{\text{bbox-2D}}_k, \sigma^{\text{bbox-2D}}_k) &= p^{\text{bbox-2D}}_\phi(v_k) \; , \\ \overline{\mathcal{P}}^{\text{bbox-2D}}_{(t),k} &\sim N(\mu^{\text{bbox-2D}}_k, \sigma^{\text{bbox-2D}}_k) \; .\end{aligned}$$

We decompose $\overline{\mathcal{P}}^{\text{bbox-2D}}_{(t),k}$ as $\overline{\mathcal{P}}^{\text{bbox-2D}}_{(t),k} = (\overline{\mathcal{P}}^{y}_{(t),k}, \overline{\mathcal{P}}^{x}_{(t),k}, \overline{\mathcal{P}}^{h}_{(t),k}, \overline{\mathcal{P}}^{w}_{(t),k})$. Then $\overline{\mathcal{P}}^{y}_{(t),h}$ and $\overline{\mathcal{P}}^{x}_{(t),k}$ parameterize the position of the object according to:

$$\begin{aligned}\mathcal{P}^y_k &= y_k + h_k \cdot (\text{sigmoid}(\overline{\mathcal{P}}^y_{(t),k}) - 0.5) \; , \\ \mathcal{P}^x_k &= x_k + w_k \cdot (\text{sigmoid}(\overline{\mathcal{P}}^x_{(t),k}) - 0.5) \; .\end{aligned}$$

The output of the sigmoids here gives the normalized offset of the object center with respect to the center of the localization glimpse. $\overline{\mathcal{P}}^{h}_{(t),k}$ and $\overline{\mathcal{P}}^{w}_{(t),k}$ parameterize the object size as:

$$\mathcal{P}^h_k = \text{sigmoid}(\overline{\mathcal{P}}^h_{(t),k}) h_k \; , \qquad\qquad \mathcal{P}^w_k = \text{sigmoid}(\overline{\mathcal{P}}^w_{(t),k}) w_k \; .$$

Finally we have $\mathcal{P}^{\text{bbox-2D}}_k = (\mathcal{P}^y_k, \mathcal{P}^x_k, \mathcal{P}^h_k, \mathcal{P}^w_k)$.

Extracting a Glimpse

Here, as in Discovery, we extract a glimpse focused on the object location in order to obtain fine-grained, high-resolution information about the object appearance:

$$g_k = ST(I_{(t)}, \mathcal{P}_k^{\text{bbox-2D}}) \; .$$

The glimpse is subsequently encoded by an object encoder network p_ϕ^{obj}:

$$v_k^{\text{obj}} = p_\phi^{\text{obj}}(g_k) \; .$$

In our experiments a single object encoder is shared between Discovery and Propagation (i.e. $p_\phi^{\text{obj}} = d_\phi^{\text{obj}}$).

Predicting Attribute Updates

Conditioning on the fine-grained appearance information in v_k^{obj}, we next infer an update to *what*, and a new value for *visible*, using MLPs p_ϕ^{what} and p_ϕ^{visible}:

$$\begin{aligned}
(\mu_k^{\text{what}}, \sigma_k^{\text{what}}) &= p_\phi^{\text{what}}(v_k, v_k^{\text{obj}}) \; , \\
\overline{\mathcal{P}}_{(t),k}^{\text{what}} &\sim N(\mu_k^{\text{what}}, \sigma_k^{\text{what}}) \; , \\
\mathcal{P}_{(t),k}^{\text{what}} &= \mathcal{S}_{(t-1),k}^{\text{what}} + \overline{\mathcal{P}}_{(t),k}^{\text{what}} \; ,
\end{aligned}$$

$$\begin{aligned}
\mu_k^{\text{visible}} &= p_\phi^{\text{visible}}(v_k, v_k^{\text{obj}}, \mathcal{P}_{(t),k}^{\text{what}}) \; , \\
\overline{\mathcal{P}}_{(t),k}^{\text{visible}} &\sim \text{Logistic}(\mu_k^{\text{visible}}) \; , \\
\mathcal{P}_{(t),k}^{\text{visible}} &= \text{sigmoid}(\overline{\mathcal{P}}_{(t),k}^{\text{visible}}) \; .
\end{aligned}$$

Note that $\mathcal{P}_{(t),k}^{\text{visible}}$ can be viewed as a BinConcrete random variable.

For the *pres* attribute, we set $\mathcal{P}^{\text{pres}}_{(t),k} = \mathcal{S}^{\text{pres}}_{(t-1),k}$, which encodes the inductive bias (shared with humans [146]) that objects tend not to pop out of existence[3]. We predict a distribution over object depth $\mathcal{P}^{\text{depth}}_{(t),k}$, and subsequently over $\mathcal{P}^{\text{where-3D}}_{(t),k}$, in a manner identical to Discovery. Discovery and Propagation use a shared depth predictor (i.e. $d^{\text{depth}}_{\phi} = p^{\text{depth}}_{\phi}$) similar to the situation with the shared object encoder d^{obj}_{ϕ}. Finally we update the hidden state:

$$\mathcal{P}^{\text{state}}_{(t),k} = \text{RNN}_{\phi}(\mathcal{P}^{\text{where-3D}}_{(t),k}, \mathcal{P}^{\text{what}}_{(t),k}, \mathcal{P}^{\text{pres}}_{(t),k}, \mathcal{S}^{\text{state}}_{(t-1),k}) \, .$$

6.5.5 Selection

After running the Propagation and Discovery modules, the next step is to combine $\mathcal{D}_{(t)}$ and $\mathcal{P}_{(t)}$ into a single set of selected objects $\mathcal{S}_{(t)}$ which will be rendered and sent forward to future timesteps. The Selection module consists of two steps: suppression and selection.

Suppressing Objects

The suppression step is inspired by Non-Maximum Suppression from supervised object detection [76, 132, 134], discussed in Section 2.4.5. It allows pairs of overlapping objects in $\mathcal{D}_{(t)}$ to suppress one another, and allows objects in $\mathcal{P}_{(t)}$ to suppress overlapping objects in $\mathcal{D}_{(t)}$. Importantly, it has learnable components, so that it is able to learn when two predicted objects are similar enough that they pick out the same real-world object, or when one predicted object is a part of some larger object. Suppression is implemented by reducing the *pres* attribute of suppressed objects.

First, we apply a form of learned Non-Maximum Suppression between Discovered objects; this will help suppress duplicate object detections (which can happen, for example, when an object's center is near the border between two Discovery grid cells), or cases where one grid cell detects a whole object while another detects an object part. We find every pair of Discovered objects, with indices ij and $k\ell$ respectively (and $ij \neq k\ell$), that

[3]Some kinds of objects *can* pop out of existence (e.g. bubbles); we leave such cases for future work.

have a non-zero amount of overlap. That is, we find all pairs which satisfy:

$$\mathcal{D}^{\text{bbox-2d}}_{(t),ij} \cap \mathcal{D}^{\text{bbox-2d}}_{(t),k\ell} \neq \emptyset \, .$$

For every such pair, we pass attributes of both objects through an MLP $s^{\text{disc-disc}}_{\phi}$, and compute suppression values:

$$\begin{aligned}
\text{logit}_{ij,k\ell} &= s^{\text{disc-disc}}_{\phi}(\mathcal{D}_{(t),ij}, \mathcal{D}_{(t),k\ell}) \, , \\
\text{suppression}_{ij \to k\ell} &= \max(\tanh(\text{logit}_{ij,k\ell}), 0) \, , \\
\text{suppression}_{k\ell \to ij} &= -\min(\tanh(\text{logit}_{ij,k\ell}), 0) \, .
\end{aligned}$$

The arrow in the subscript indicates the direction of suppression. By this formulation, ij can suppress $k\ell$ ($\text{suppression}_{ij \to k\ell} > 0$), or $k\ell$ can suppress ij ($\text{suppression}_{k\ell \to ij} > 0$), but not both.

Next, we apply a similar set of operations, this time allowing Propagated objects to suppress Discovered objects (but not vice versa). This gives the network a means of rejecting Discovered objects which are already accounted for by some Propagated object; this is important, as the Discovery module itself has no knowledge of which objects the network is already tracking, and so has no choice but to output every object that it has visual evidence for. For every pair of Propagated and Discovered objects, with indices m and ij respectively, we find those which satisfy:

$$\mathcal{P}^{\text{bbox-2d}}_{(t),m} \cap \mathcal{D}^{\text{bbox-2d}}_{(t),ij} \neq \emptyset \, .$$

For every such pair, we pass attributes of both objects through a neural network $s^{\text{prop-disc}}_{\phi}$, and compute suppression values:

$$\text{suppression}_{m \to ij} = \text{sigmoid}(s^{\text{prop-disc}}_{\phi}(\mathcal{P}_{(t),m}, \mathcal{D}_{(t),ij})) \, .$$

Finally, we use both sets of suppression values to overwrite $\mathcal{D}^{\text{pres}}_{(t),ij}$:

$$\mathcal{D}^{\text{pres}}_{(t),ij} = \mathcal{D}^{\text{pres}}_{(t),ij} \cdot (1 - \text{suppression}_{ij}) \text{ ,}$$

where

$$\text{suppression}_{ij} = \max_{\text{idx} \in \text{overlaps}(ij)} \text{suppression}_{\text{idx} \to ij} \text{ ,}$$

where $\text{overlaps}(ij)$ is the set of indices of all objects (from either Propagation or Discovery) whose bounding boxes have non-zero overlap with $\mathcal{D}^{\text{bbox-2d}}_{(t),ij}$.

Selecting a Subset of Objects

The last step in the Selection module is simply selecting a subset of objects to render and to send forward to future timesteps. To see why this is necessary, consider that each timestep, the number of objects yielded by the Discovery module is HW (since the shape of the Discovery grid is (H, W)). If we were to keep all of these objects, the number of objects tracked by 3DOM after timestep t would be $(t + 1)HW$, which would quickly become computationally intractable. To mitigate this, the Selection module picks the K objects from $\mathcal{D}_{(t)} \cup \mathcal{P}_{(t)}$ with largest value for the *pres* attribute, where K is an integer hyperparameter. The selected objects are gathered together as the set of Selected objects $\mathcal{S}_{(t)}$. Note that this implies that the number of objects in $\mathcal{P}_{(t)}$ is also K.

A reasonable rule of thumb for choosing K is to have it be around 25% larger than the maximum number of objects we expect the network to ever have to track at once. This ensures that there are always a reasonable number of object slots which are "off" or "empty" (i.e. have *pres* near 0), which is important for making sure that such slots continue to receive gradient throughout training.

6.5.6 Rendering

The rendering module is the sole constituent of the VAE generative model. It takes in the set of selected objects $\mathcal{S}_{(t)}$ and renders them into a frame.

We start by focusing on a single object with index k. First, a small RGB-alpha image is predicted for the object by an object decoder network r_θ^{obj}, based on $\mathcal{S}_{(t),k}^{\text{what}}$:

$$\begin{aligned}\beta_k^{\text{logit}}, \xi_k^{\text{logit}} &= r_\theta^{\text{obj}}(\mathcal{S}_{(t),k}^{\text{what}}) \,, \\ \beta_k &= \text{sigmoid}(\mu^\beta + \sigma^\beta \beta_k^{\text{logit}}) \,, \\ \xi_k &= \text{sigmoid}(\mu^\xi + \sigma^\xi \xi_k^{\text{logit}}) \,.\end{aligned}$$

The appearance map β_k has shape $(H^{\text{obj}}, W^{\text{obj}}, 3)$, while the partial transparency map ξ_k has shape $(H_{\text{obj}}, W_{\text{obj}}, 1)$, for integers H^{obj}, W^{obj}. Meanwhile $\sigma^\beta, \mu^\beta, \sigma^\xi$ and μ^ξ are scalar hyperparameters that can be used to control the relative speed with which appearance and transparency are trained.

To obtain the final transparency map α_k, the partial transparency map ξ_k is multiplied by $\mathcal{S}_{(t),k}^{\text{pres}} \cdot \mathcal{S}_{(t),k}^{\text{visible}}$; this ensures that objects are only rendered to the image to the extent that they are both present and visible:

$$\alpha_k = \xi_k \cdot \mathcal{S}_{(t),k}^{\text{pres}} \cdot \mathcal{S}_{(t),k}^{\text{visible}} \,.$$

For each object, an inverse spatial transformer parameterized by $\mathcal{S}_{(t),k}^{\text{bbox-2D}}$ is then used to create image-sized versions of these three maps, with the input maps placed in the correct location:

$$\alpha'_k, \beta'_k = ST^{-1}([\alpha_k, \beta_k], \mathcal{S}_{(t),k}^{\text{bbox-2D}}) \,.$$

To obtain the output frame, the image-sized appearance maps are combined using alpha blending. We first sort objects in order of decreasing depth, so that the object with index

$k = 1$ has highest depth, and object $k = K$ has smallest depth. Index $k = 0$ will represent the background, and we set $\alpha'_0 = 1$ and $\beta'_0 = R_{(t)}$ (the image rendered by the SRN). Then alpha blending is implemented as:

$$\hat{I}_{(t)} = \sum_{k=0}^{K} \left(\prod_{i=k+1}^{K} (1 - \alpha'_i) \right) \alpha'_k \beta'_k \, .$$

The output of rendering is an image with dimensions $(H^{\text{inp}}, W^{\text{inp}}, 3)$; to obtain $P_\theta(x|z)$, we use this image to parameterize the mean of a set of conditionally independent Gaussian random variables (one for each pixel and channel), with fixed standard deviation σ^{image}.

Comparison with Previous Rendering Modules

3DOM's rendering module is similar to the one used in SPAIR and SILOT, with one major difference. For all three models, we have access to object locations (in camera coordinates), object appearances, object alphas, and relative object depths, and our goal is to combine them into a single output image. Alpha blending, as we have used here in 3DOM, is one straightforward way to achieve this. However, one property of alpha blending is that it is not differentiable with respect to the depth values, since the output image only depends on the depth values through the (non-differentiable) ordering that they impose on the objects (largest depth to smallest depth). In 3DOM this is not an issue, as we have an alternative signal for training depth (discussed in the next section). However in SPAIR and SILOT there is no alternative means of training depth. Consequently, in those past models we used a more complicated rendering model in which the depth values were used to differentiably parameterize a convex combination between overlapping objects, a design which allowed the depth values to be trained through their influence on the rendered images.

6.5.7 Training the Depth Predictor

For 3DOM to successfully discover and track 3D objects, it must be able to predict the depth of objects based only on their 2D appearance and apparent size. To achieve this, we instantiate an MLP d_ϕ^{depth} which is used to predict object depth inside both Discovery ($\mathcal{D}_{(t),k}^{\text{depth}}$) and Propagation ($\mathcal{P}_{(t),k}^{\text{depth}}$). This MLP takes as input the encoded glimpse of the object, as well as the apparent object size ($\mathcal{O}_{(t),k}^{h}, \mathcal{O}_{(t),k}^{w}$). The output is a mean and standard deviation over depth, (μ^{depth}, σ^{depth}). This was discussed in Section 6.5.3.

As we show empirically in Section 6.7.3, merely treating the depth attribute like any other latent variable is not sufficient for learning quality depth estimates, especially when objects can move. To provide additional signal for training the depth predictor, we designed an additional loss function based on epipolar geometry [68][4].

The depth loss function works as follows. Assume for now that we are dealing with stationary objects. Then note that any depth prediction made for an object at time t induces a prediction about where the object will appear on the camera at time $t+1$. If we are able to relocate the object at time $t+1$, then we can compare our prediction about where the object would appear with where it actually appeared. The Euclidean distance between the predicted and actual locations is called the *reprojection error*. When the camera moves purely rotationally between t and $t+1$, the reprojection error is necessarily 0. However, if there is a component of the camera motion that is translational and not colinear with the ray extending from the camera to the object at time t, then the reprojection error is a strong signal as to the quality of the depth prediction. Thus, we can minimize the reprojection error with respect to the depth prediction in order to train the depth predictor. A diagram illustrating this loss is shown in Figure 6.3.

[4]Epipolar geometry is the study of the geometry of a 3D point or object viewed from two spatially separated cameras

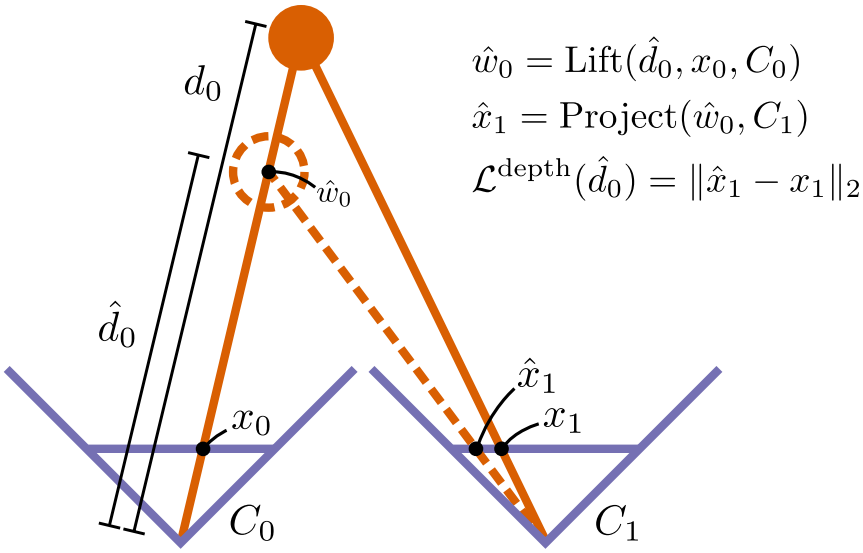

Figure 6.3. Top-down view showing how 3DOM's depth prediction network is trained via a loss function based on epipolar geometry [68]. The blue triangles depict the camera pose at subsequent timesteps. The filled orange circle is the true location of the object of interest, assumed to be stationary, with true (but unknown) depth d_0 at time 0. A depth prediction $\hat{d}_0$ is made based on object size and appearance at time 0. A lifting step then yields a predicted 3D position, $\hat{w}_0$. This is then projected onto C_1 (the camera at the *next* timestep) to obtain a predicted projected location for the object, $\hat{x}_1$. The depth loss is then simply the reprojection error, or the distance between the predicted object projection $\hat{x}_1$ induced by $\hat{d}_0$, and the true object projection x_1 (note that x_1 is a known quantity, provided that the object has been successfully localized at time 1). Gradient flows from this loss back to the depth prediction $\hat{d}_0$. In effect, this loss encourages the network to predict depth values which yield small projection error at the next timestep. For simplicity we show a stationary object here; however we have found that this approach works reasonably well even for objects that are always in motion (one possibility is that the errors introduced by object motion "average out").

Figure 6.3 shows a simplified version of the depth loss function. The proper depth function, defined in terms of the attributes of objects in 3DOM, is given as:

$$\mathcal{L}^{\text{depth}} = \sum_{t=1}^{T-1} \sum_{k=1}^{K} \mathcal{P}^{\text{visible}}_{(t),k} \cdot \mathcal{P}^{\text{pres}}_{(t),k} \cdot \left\| \frac{\text{Project}(\mathcal{S}^{\text{where-3D}}_{(t-1),k}, C_{(t)}) - (\mathcal{P}^{y}_{(t),k}, \mathcal{P}^{x}_{(t),k})}{(\mathcal{S}^{h}_{(t-1),k}, \mathcal{S}^{w}_{(t-1),k})} \right\|_2 . \quad (6.2)$$

We scale by the *pres* and *visible* attributes, since it is necessary for the object to be both present and visible at the current timestep for the loss function to be meaningful. Within the $\|\cdot\|_2$, all values are 2D vectors and operations are applied elementwise. $\text{Project}(\mathcal{S}^{\text{where-3D}}_{(t-1),k}, C_{(t)})$ gives the estimated 3D position of the object at the previous timestep projected onto the camera for the current timestep. $(\mathcal{P}^{y}_{(t),k}, \mathcal{P}^{x}_{(t),k})$ is the "ground-truth"

position of the object on the current camera, assuming that the Propagation module has successfully re-localized the object (note that this *does not* require the ground-truth object position to be given as part of the training data). Finally, we divide by the apparent size from the previous timestep, $(\mathcal{S}^h_{(t-1),k}, \mathcal{S}^w_{(t-1),k})$, to compensate for the fact that objects with smaller apparent size tend to be further away and have smaller values of the reprojection error. Gradient is backpropagated through $\mathcal{S}^{\text{where-3D}}_{(t-1),k}$ to the depth estimates from the previous timestep; for all other values that appear in Equation (6.2), there is an implicit $\text{stop} - \text{gradient}$, as this loss function does not provide a useful training signal for them.

6.5.8 Prior Propagation

3DOM includes one additional module, called Prior Propagation, which is only peripherally involved in tracking objects. Prior Propagation is similar to Propagation, in that it outputs a set of objects $\widehat{\mathcal{P}}_{(t)}$ intended to be a propagated version of $\mathcal{S}_{(t-1)}$. However, unlike standard Propagation, Prior Propagation *does not have access to the input frame*. Thus we think of Prior Propagation as *predicting* attribute updates based on object history (summarized by $\mathcal{S}^{\text{state}}_{(t-1),}$), whereas standard Propagation has the luxury of *inferring* attribute updates based on the input frame. From the perspective of the VAE framework, Prior Propagation is a *learned prior* distribution, with Propagation itself acting as the posterior. Prior Propagation is trained to match the output of Propagation, with the mismatch measured by Kullback-Leibler (KL) Divergence. Note that SILOT had a similar Prior Propagation module; the primary difference is that here, the Prior Propagation module operates in 3D space, making predictions primarily about the world-frame attributes $\widehat{\mathcal{P}}^{\text{where-3D}}_{(t),k}$, $\widehat{\mathcal{P}}^{\text{what}}_{(t),k}$ and $\widehat{\mathcal{P}}^{\text{pres}}_{(t),k}$.

The predictions made by the Prior Propagation module are used in two crucial places. First, consider the following issue that crops up in the main Propagation module: whenever the network predicts a low value for the visibility of an object ($\mathcal{P}^{\text{visible}}_{(t),k} < 0.5$), this indicates that the frame does not contain information with which to update the object, and therefore the predicted attribute updates are unlikely to be useful. For such objects, we fall back on

the attribute predictions made by the *Prior* Propagation module, which, after all, is being trained to predict object attributes without access to visual information. This mechanism can help the network to better keep track of objects that are occluded or off screen.

The second place that Prior Propagation is used is in *filtering* inferences made by the main Propagation module for $\mathcal{P}^{\text{where-3D}}_{(t),k}$. Predicting depth from only object appearance and apparent size is a difficult task, especially in sub-optimal conditions such as when an object is only partially onscreen; this manifests as errors in $\mathcal{P}^{\text{depth}}_{(t),k}$ and $\mathcal{P}^{\text{where-3D}}_{(t),k}$. To help minimize these errors, we combine the posterior location estimate $\mathcal{P}^{\text{where-3D}}_{(t),k}$ with the *prior* location estimate $\widehat{\mathcal{P}}^{\text{where-3D}}_{(t),k}$, in a manner similar to a Kalman Filter [91] (recall that we predict multivariate Normal distributions over $\mathcal{P}^{\text{where-3D}}_{(t),k}$ and $\widehat{\mathcal{P}}^{\text{where-3D}}_{(t),k}$, which integrate well with the Kalman filter framework). We find that this results in significantly smoother inferred object trajectories, and also results in more accurate predictions made by Prior Propagation, since the history that it conditions on becomes much less noisy.

6.6 Training

Training 3DOM starts with an initial stage in which SRN-related parameters (specifically the hypernetwork H (which outputs Φ_n) and learnable elements of the raymarching algorithm) are trained in isolation. Importantly, this stage uses the *same training data* as subsequent training stages in which the dynamic-object parts of 3DOM are also trained.

After the initial SRN-specific stage, the entire model is trained as a VAE, maximizing the evidence lower bound (ELBO) [95] of the training data using the Adam stochastic optimizer [94] (recall that the Propagation, Discovery and Selection modules form the VAE inference network Q_ϕ, while the Rendering module forms the VAE generative network $P_\theta(x|z)$). We also minimize the depth loss discussed in Section 6.5.7. Overall the training loss is:

$$
\begin{aligned}
&\mathcal{L}(x_{(0:T-1)}, u_{(0:T-2)}, \theta, \phi) \\
&= \sum_{t=0}^{T-1} E_{\overline{\mathcal{P}}_{(0:t-1)}, \overline{\mathcal{D}}_{(0:t-1)} \sim Q_\phi} \Big[\\
&\qquad \int Q_\phi(\overline{\mathcal{D}}_{(t)} | x_{(t)}, \mathcal{P}_{(t)}) Q_\phi(\overline{\mathcal{P}}_{(t)} | x_{(t)}, \mathcal{S}_{(t-1)}) \log P_\theta(x_{(t)} | \mathcal{S}_{(t)}) \, d\overline{\mathcal{P}}_{(t)} \, d\overline{\mathcal{D}}_{(t)} \quad \text{(reconstruction)} \\
&\qquad + D_{KL}(Q_\phi(\overline{\mathcal{P}}_{(t)} | x_{(t)}, \mathcal{S}_{(t-1)}) \parallel P(\overline{\mathcal{P}}_{(t)})) \quad \text{(Prop KL with fixed prior)} \\
&\qquad + D_{KL}(Q_\phi(\overline{\mathcal{P}}_{(t)} | x_{(t)}, \mathcal{S}_{(t-1)}) \parallel P_\theta(\overline{\mathcal{P}}_{(t)} | \mathcal{S}_{(t-1)})) \quad \text{(Prop KL with learned prior)} \\
&\qquad + D_{KL}(Q_\phi(\overline{\mathcal{D}}_{(t)} | x_{(t)}, \mathcal{P}_{(t)}) \parallel P(\overline{\mathcal{D}}_{(t)})) \quad \text{(Disc KL with fixed prior)} \\
&\quad + \lambda^{\text{depth}} \cdot \mathcal{L}^{\text{depth}} \, ,
\end{aligned}
$$

where $\mathcal{L}^{\text{depth}}$ is the loss function defined Equation (6.2), and λ^{depth} is a hyperparameter controlling the weight on the depth loss.

We employ the same two training tricks used in SILOT, namely discovery dropout (randomly turning off the Discovery module during training, thereby encouraging the network to prefer using the Propagation module when possible) and a form of curriculum learning in which the length of the training videos is gradually increased from 1 up to the maximum training video length.

6.7 Experiments

To assess the capabilities of 3DOM, we ran a number of experiments in environments created using the Miniworld framework [25]. In particular we are interested in 3DOM's ability to deal with large numbers of objects, to track moving objects through 3D space, to generalize to unseen environments, and to make predictions about object trajectories. Code for running these experiments is available online[5]. Full experiment details, including

[5] `https://github.com/e2crawfo/threed`

neural architectures, training schedule, and procedures for hyperparameter selection, can be found in Appendix D.

6.7.1 Metrics

We used 3 primary metrics to assess models' ability to discover and track objects: Average Precision (AP, a standard object detection metric [44]), and two versions of Multi-Object Tracking Accuracy (MOTA, a standard object tracking metric [119]). 2D-MOTA assesses object tracking on the image plane (treating objects as 2D bounding boxes), while 3D-MOTA assesses object tracking in 3D space (treating objects as points in 3D space). For all 3 metrics, higher is better. All data points in the plots presented below are averages over 4 random seeds, and filled regions denote 95% confidence intervals for the mean.

6.7.2 Simulated Object Picking

As a first step, we aim to verify that 3DOM is capable of learning to segment objects and predict their 3D location, in a setting where all objects are stationary. To achieve this, we designed a simulated environment that loosely mimics an object picking scenario, an important task in robotics that is typically solved using large amounts of annotated data [41]. Each static environment is a "bin" with a different texture. Each episode, one of these bins is selected and populated with a large number of objects (taken from the set of default objects provided with Miniworld). The camera lives on a sphere that is concentric with the bin; between frames the camera moves in a random direction on this sphere, always constrained to look at the center of the bin. Each training/validation/test set contains 60000/100/250 videos, and each video is 2 frames long. Note that since objects are stationary in this environment, longer videos would present no extra difficulty.

We compared 3DOM against two baselines. The first is a model we call 2DOM; this is an ablation of 3DOM that is unaware of the camera, and can thus be treated as representative

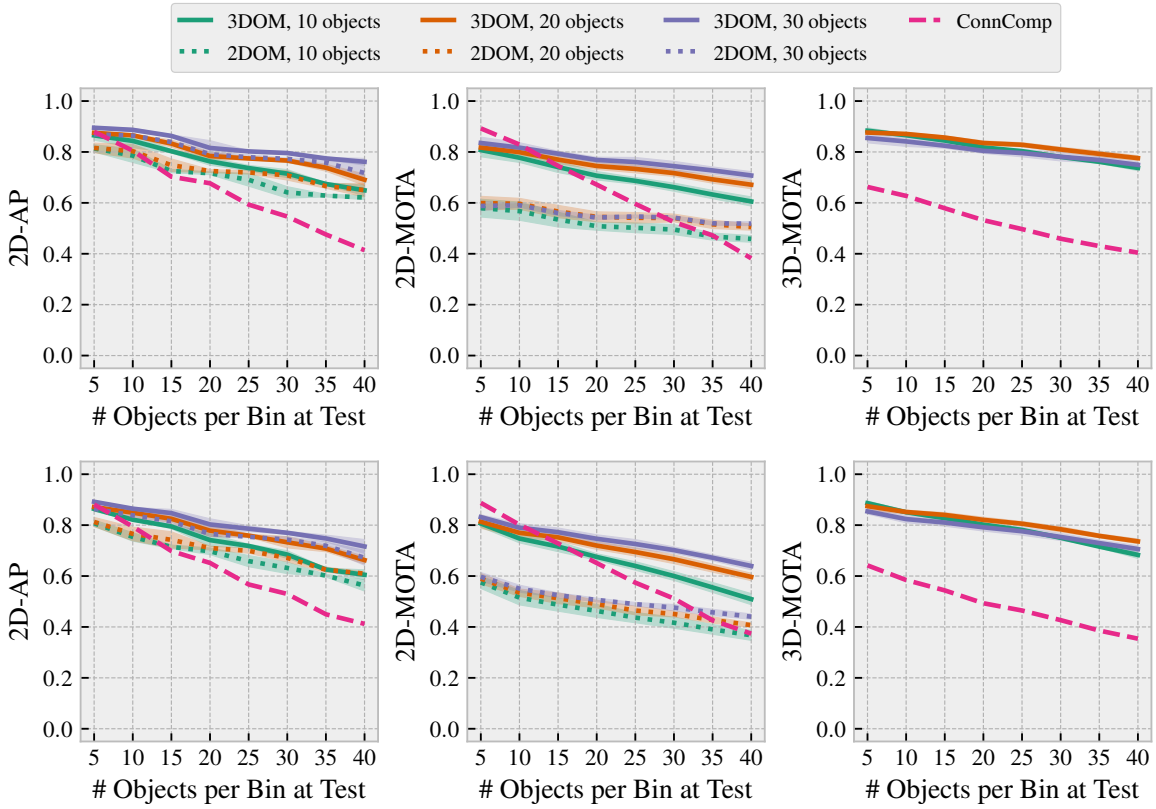

Figure 6.4. Showing performance in Simulated Object Picking, on bins containing different numbers of moving objects. All data points are averages over 4 random seeds (except for the fully deterministic ConnComp algorithm), and filled regions are 95% confidence intervals for the mean. Top: Performance on mixed bins, similar to those seen during training. Bottom: Performance on bins where all objects are identical, never seen during training. The legend indicates the number of objects per bin in the data each network was trained on. 3D-MOTA for 2DOM was well below zero, and so is not shown here.

of past 2D OOWMs which model objects as living on the image plane, such as SQAIR [98], SILOT [29] and SCALOR [85] (in fact 2DOM is nearly identical to SILOT).

The second is a baseline model called ConnComp (Connected Components), which segments objects by computing an error image $|I_{(t)} - R_{(t)}|$ (recall that $R_{(t)}$ is the image rendered by the SRN), thresholding it, and then taking the connected components of this error image as objects. For estimating object depth, ConnComp uses the median ground-truth depth over all pixels in the connected component. 3D tracking is achieved using the Hungarian algorithm [101]. In effect, ConnComp segments objects by computing "blobs" of pixels that differ substantially from the image rendered by the SRN; to be successful,

it requires the SRN to be a good model of the static components in an image, and that objects do not overlap visually (since objects that do overlap will be combined into a single image). ConnComp's threshold parameter is chosen so as to maximize 3D-MOTA on the validation dataset.

We trained models with different numbers of objects per episode, and then evaluated the trained models on bins with different numbers of objects, including bins with more objects than any model would have seen during training. This allows us to determine each model's ability to deal with increasing degrees of clutter and overlap, and to generalize to different numbers of objects than what it was trained on.

The results, shown in the top of Figure 6.4, indicate that 3DOM is able to segment objects (AP), predict their depth (3D-MOTA) and track them (2D/3D-MOTA), and scales gracefully to many-object scenes. Training on large numbers of objects does help performance, but even models trained on only 10 objects per bin are able to generalize reasonably well to bins containing 40 objects. As for 2DOM, we see that it is unable to learn to track objects in 3D space, unsurprising since it is essentially a 2D model. More interesting is the fact that while 2DOM achieves competitive object detection performance (AP), its ability to track objects, even in 2D, is poor (2D-MOTA). One explanation is that 2DOM's neglect of 3D space makes it hard to guess where in the image objects will appear after the camera has moved, making it difficult to maintain object identity over time.

We also tested the models on *homogeneous* bins, i.e. bins in which all objects are identical. Note that such bins are not present in the training data, and thus here we are testing a kind of out-of-distribution generalization. Segmenting objects in homogeneous bins should be somewhat more difficult than the heterogeneous bins that make up the training datasets, since color is no longer as useful a segmentation cue. Results are shown in the bottom of Figure 6.4; performance is degraded by only a small amount compared to performance on mixed bins, despite the fact that the models did not encounter homogeneous bins during training.

Qualitative Results

Qualitative results for 3DOM in the Simulated Object Picking task are shown in Figure 6.5 for heterogeneous bins and Figure 6.6 for homogeneous bins. Together these results confirm that the network is able to segment objects (even in the face of significant clutter), predict their depth, and track them as the camera moves. The versions of 3DOM trained on 30 objects per bin are somewhat better at dealing with clutter than the versions trained on 10 objects per bin. All networks struggle somewhat with the most distant objects, probably because of their small apparent size (and correspondingly small contribution to the reconstruction loss) and the fact that they have the highest degree of inter-object occlusion.

6.7.3 First-Person Maze Navigation

In the next experiment we tested 3DOM in a first-person setting with objects that move. For the static 3D scenes, we randomly generated 20 mazes on a 4x4 grid of rooms, each room with dimensions 4 metres x 4 metres. Each maze has different, randomly selected textures for the floor, walls and ceiling. For the objects that populate these static scenes, we use 4 different shapes (sphere, cube, cone, torus) and 3 different color pairs (red-yellow, green-cyan, blue-magenta), combining for a total of 12 object kinds, each of which is randomly assigned a canonical size. Each episode, one of the 20 mazes is randomly selected and populated with a number of objects. At the start of the episode, a biased coin is flipped for each object to determine whether the object moves. Objects that move proceed in a random 3D direction, bouncing off any surfaces they encounter. Objects are permitted to float, and are not bound by gravity. All objects change color over time, cycling back and forth between the two colors in their color pair, completing a cycle once every 8 frames. Videos are 8 frames long, and each training/validation/test dataset contains 20000/100/250 videos, respectively.

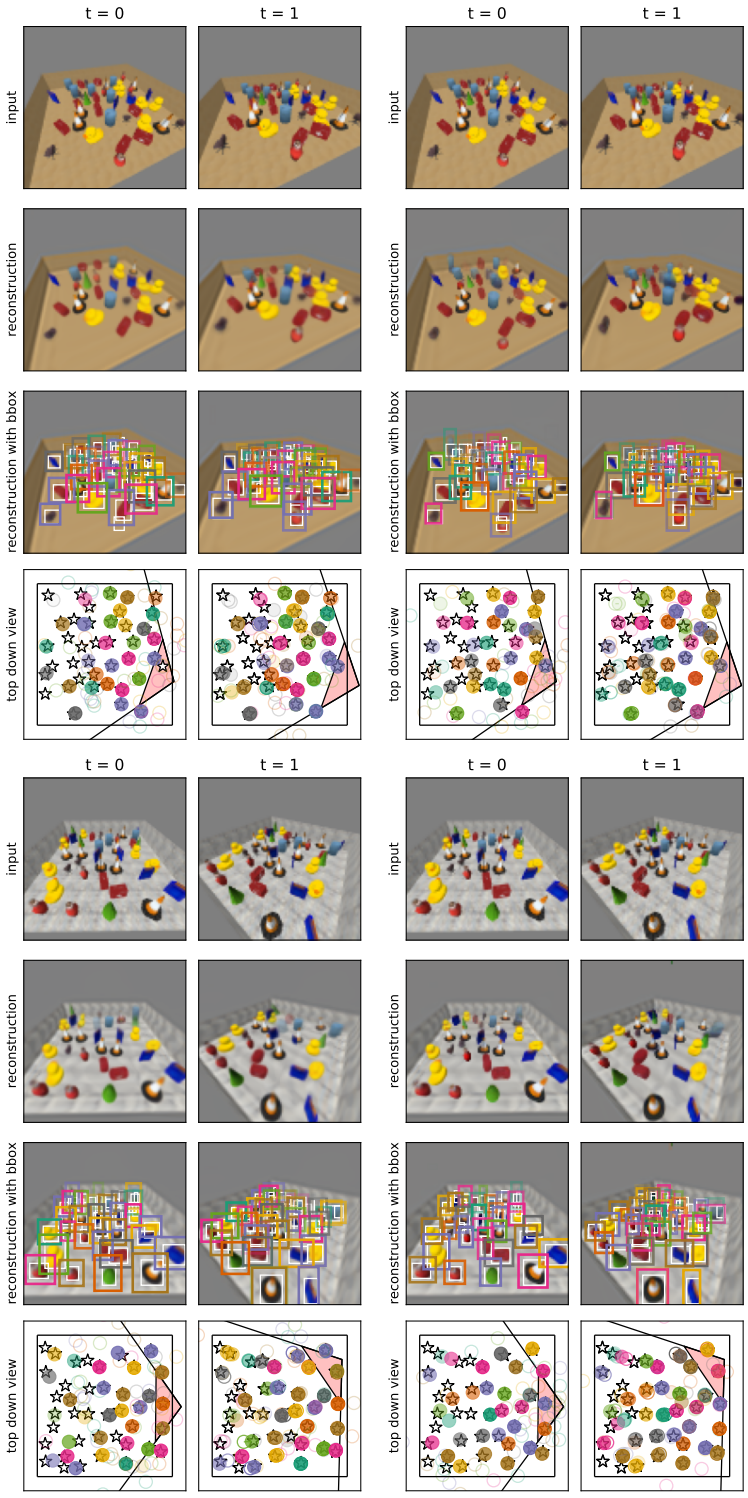

Figure 6.5. Qualitative performance of 3DOM on Simulated Object Picking trained with 10 (left) and 30 (right) objects per bin, running in a test setting with 40 objects per bin. Top and bottom are two different examples. Row 3 shows ground-truth bounding boxes in white, predicted bounding boxes in color. In the top-down view shown in Row 4, ground-truth object locations are represented by stars, while predicted objects are represented by circles. The transparency for the bounding boxes in Row 3 and the fill of the circles in Row 4 is proportional to each object's *pres* value. Color indicates object identity, and colors of the bounding boxes in Row 3 match up with the colors of the circles in Row 4.

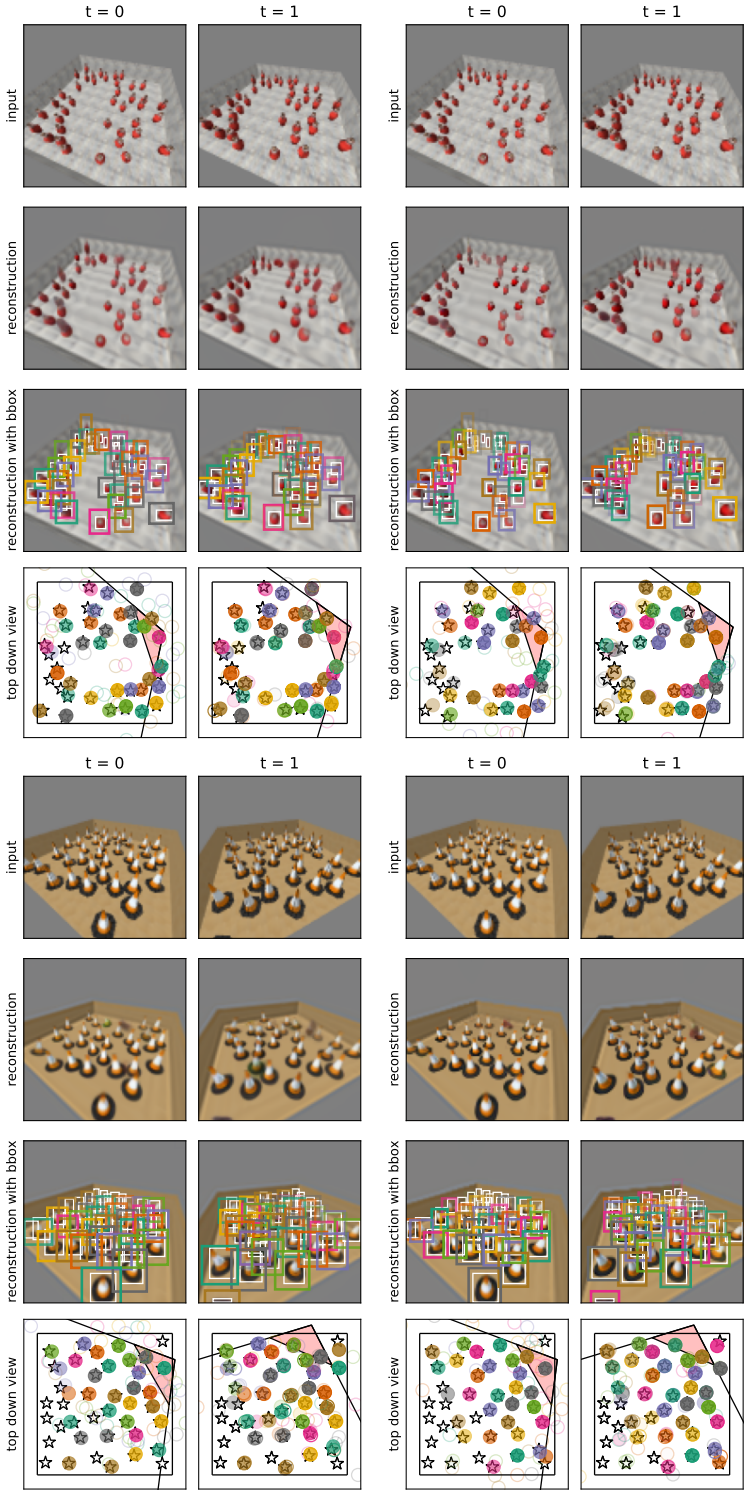

Figure 6.6. Same interpretation as Figure 6.5; see the caption of that figure for details. Here we are showing performance for homogeneous bins in which all objects are identical. The networks were trained only on heterogeneous bins, and thus never encountered homogeneous bins at training time.

This setting is more difficult than Simulated Object Picking in a number of ways. The fact that the agent moves through the maze, rather than having a top-down view, means there is a larger degree of inter-object occlusion, a wider range of object scales/distances to deal with, and forces the network to reason about objects that move out of view (e.g. off camera, behind walls, or behind other objects). Depth learning is made more difficult by the fact that objects can move and float. Additionally, the fact that each of the 12 object kinds is assigned a different size means that the network must combine information about shape, color *and* apparent size in order to accurately predict object depth. Finally, the changing object colors may make tracking more difficult.

Tracking Moving Objects in 3D

Our first aim in this environment is to see how 3DOM's ability to track objects in 3D is affected by the fraction of objects that are moving in the training set; we expect that more objects moving will translate into reduced performance, as learning to predict depth becomes more difficult. To assess this aspect, we generated training datasets with different fractions of moving objects, and then evaluated the trained models in environments with different fractions of moving objects. We are also interested in assessing the importance of the depth loss (discussed in Section 6.5.7), and so here we compare against an ablated version of 3DOM in which the weight on the depth loss is set to 0, denoted as NDL (for No Depth Loss).

Results, shown in the top row of Figure 6.7, show that the depth loss is crucial for the network to learn to track objects in 3D, especially when objects are moving; NDL obtains reasonable 2D tracking performance, but largely fails at tracking in 3D (3D-MOTA for NDL trained on datasets with 50% and 100% moving objects is well below 0). For 3DOM, we see that performance is best when the training set contains a mixture of moving and stationary objects, though reasonable performance is still possible when all objects are moving.

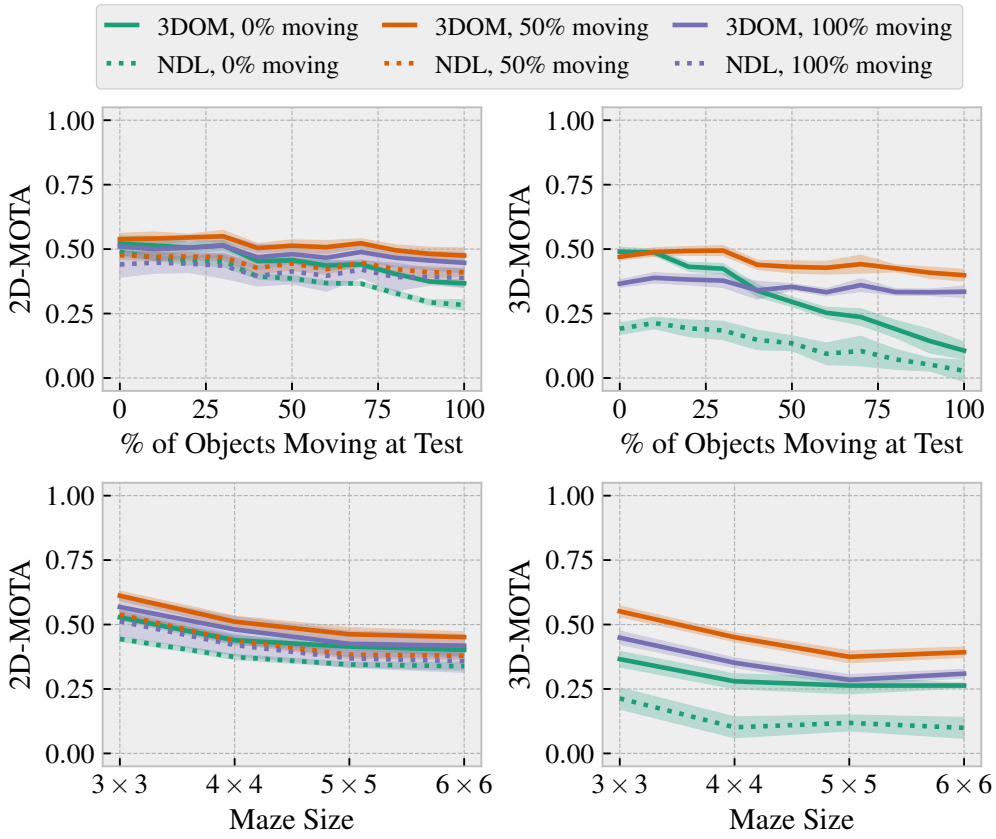

Figure 6.7. Evaluating different aspects of model performance in the Maze Navigation environment. Legend indicates the percent of objects that were moving in the training data. All data points are averages over 4 random seeds, and filled regions are 95% confidence intervals for the mean. Top row: Performance as a function of the fraction of objects that are moving. Bottom row: Performance in different static environments.

Generalizing to Different Static Environments

Recall from Section 6.3 that 3DOM assumes that training data are drawn from a fixed set of static environments; our goal in this section is to confirm that the dynamic object components of a trained 3DOM model (i.e. the parts described in Section 6.5) is able to generalize to new, unseen environments for which no trained SRN is available. To understand how this works, note that 3DOM does not depend on the SRN for predicting the latent object representations. For example, looking at the high-level overview of 3DOM in Figure 6.2, we see that the output of the SRN affects only the rendering process, and the computations of the Discovery, Propagation and Selection modules are largely independent of the SRN. Including an SRN is important during *training* as it helps the

network determine what counts as an object and consequently influences the training of Discovery and Propagation; however, it is not required during evaluation if all one is interested in are the object-oriented representations that 3DOM learns.

Thus we take the dynamic object component of the models trained in the previous subsection (specifically Discover, Propagation and Selection modules), and evaluate their ability to detect and track objects in a collection of new, randomly generated mazes of increasing size. The results, shown in the bottom row of Figure 6.7, show that the 3DOM models perform well in these new environments. Performance does decrease somewhat as the size of the mazes grows larger; this is likely due to the larger mazes having longer hallways and more objects, meaning that objects are further away on average (indeed, some will be further away than any objects encountered in the training data) and will occlude one another to a higher degree.

Predicting Object Trajectories

Beyond tracking objects, one principal task of interest is predicting object motion. This is primarily the domain of the Prior Propagation module, which, as stated previously, does not have access to the current input frame and thus must predict, rather than infer, object motion. However, learning to predict object trajectories in 3D is complicated by the fact we do not have access to ground-truth object depths, and instead must rely on learned depth predictions for supervision, which may be rather noisy. To mitigate this noise, we proposed (in Section 6.5.8) to filter depth predictions in a manner similar to a Kalman filter. In this section we are looking for evidence that this filtering indeed improves the ability of the Prior Propagation module to predict object motion. To that end, we trained an ablated version of 3DOM that did not perform this filtering (denoted as NF for No Filter) in the First-Person Maze Navigation environment. We then compared 3DOM's and NF's ability to predict object motion, as measured by the average cosine similarity between predicted object motion direction and actual object motion direction (all predictions are 1 step only). Results of this comparison are shown in Figure 6.8. There we see that removing the

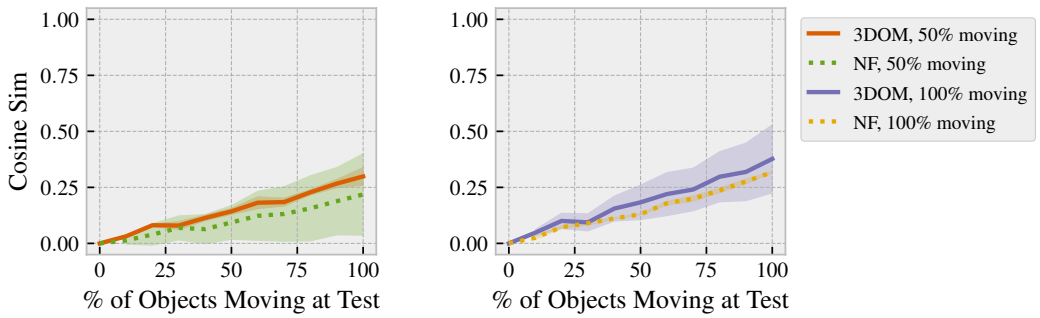

Figure 6.8. Showing performance of the Prior Propagation module at one-step prediction of object trajectories, as measured by the cosine similarity between the predicted movement direction and the actual movement direction in 3D space. All data points are averages over 4 random seeds, and filled regions are 95% confidence intervals for the mean. Left: Results for networks trained with 50% moving objects. Right: Results for networks trained with 100% moving objects.

filtering does appear to mildly hurt prediction ability, though neither result is statistically significant, perhaps due to the relatively small number of seeds used (4).

Qualitative Results

Qualitative results for 3DOM in the First-Person Maze Navigation task are shown in Figures 6.9, 6.10, and 6.11. In each figure, the top half shows performance of a network trained on 0% moving objects, while the bottom half shows performance of a network trained on 50% moving objects; in both cases, we are showing performance in a test scenario with 100% moving objects. Here we highlight a few interesting aspects of networks' performance.

We first compare and contrast the effect of the different fractions of moving objects in the training set. Somewhat surprisingly, the networks that have never seen moving objects (top half of each figure) are nevertheless able to track objects that move, so long as the movement does not cause objects to partially occlude one another. When motion does create occlusion, performance of the network trained without object movement degrades. For example, compare the networks' handling of the green/teal sphere in Figure 6.10, or the green/teal torus in Figure 6.11. In the latter case, the network trained on all stationary objects has difficulty understanding that the torus has passed partly behind the cone, while the network trained on 50% moving objects is relatively unperturbed.

Figure 6.9 demonstrates 3DOM's inductive bias for objects persisting even when not visible. The 3 objects from the first timestep move out of view as the camera rotates from left to right, but the top-down view shows that the network is still representing those objects as existing, and they could likely be reidentified if they became visible again at a later time. A related effect may be seen in the case of the small yellow/red torus in Figure 6.9; there, the networks maintain a representation of the object even after it has passed behind a wall. This is implemented by the network setting the *visible* attribute to 0, while keeping the *pres* attribute near 1; the network still believes the object exists, but has no current visual evidence for it, and so the object is not rendered.

Finally, the First-Person Maze Navigation task is significantly harder than Simulated Object Picking; the fact that objects can move, float, come in and out of view, be completely occluded by other objects, etc., are all contributing factors. One issue that the qualitative results make clear is that the networks struggle at predicting depth for objects that are moving into view, especially large objects. Many instances are in evidence in which the network's depth prediction is initially poor, when only a small portion of the object is in view, and improves over time as more and more of the object comes into view. One clear example is the second cyan sphere that comes in from the left in Figure 6.11. This depth-correction-over-time may create serious issues for training the Prior Propagation module; see Section 6.8.1 below for a discussion of this issue.

6.8 Discussion

In this chapter we have presented 3DOM, an object-oriented world model capable of learning disentangled representations of static elements and dynamic objects that make up 3D scenes. Unlike the majority of past OOWMs, 3DOM explicitly models objects as entities in a 3D world. In a number of experiments we showed the 3DOM is able to learn to segment objects in cluttered environments, predict object depth and 3D position, track objects through 3D space, and can generalize well beyond the nursery environment in

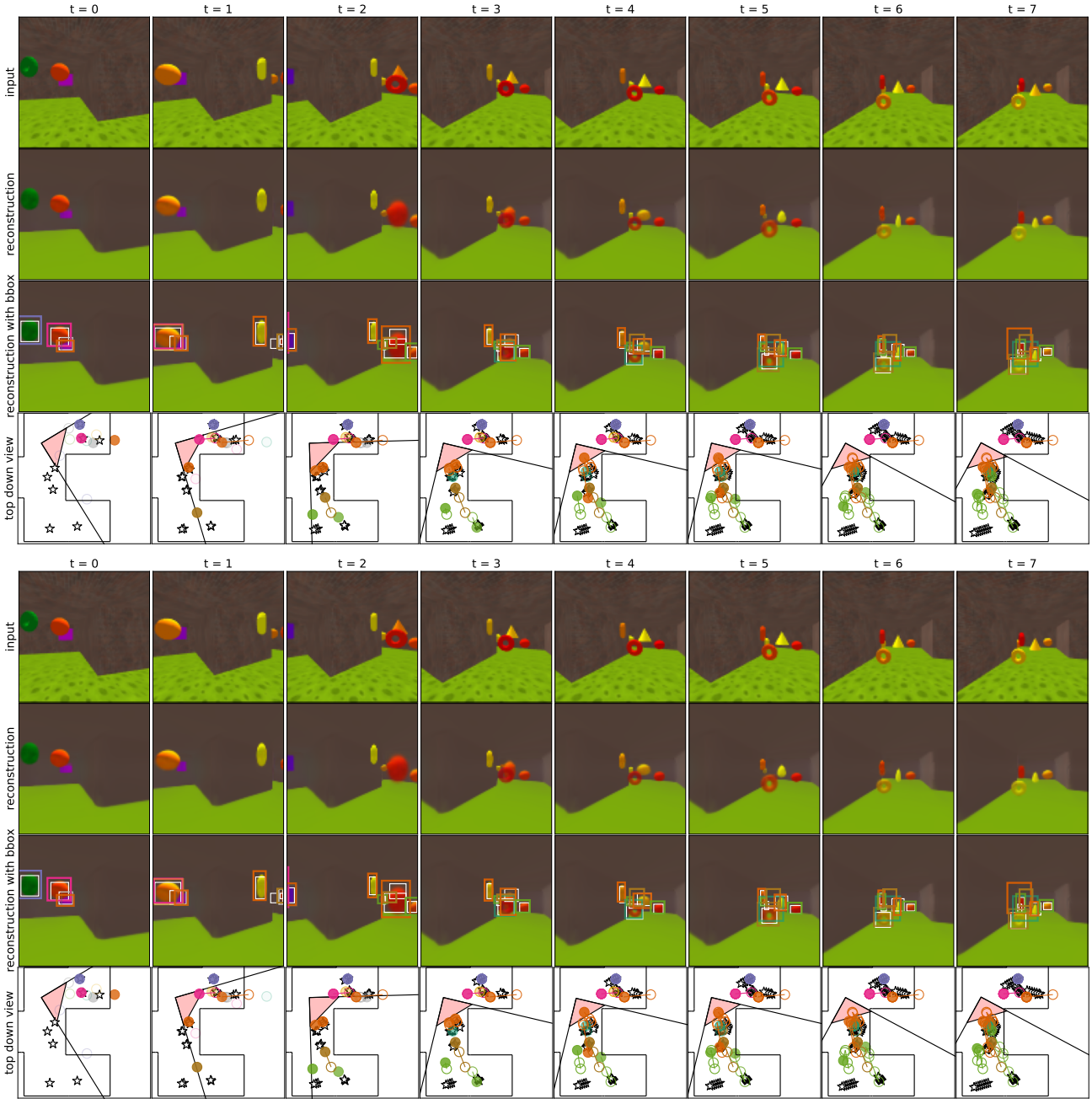

Figure 6.9. Qualitative performance of 3DOM on Maze Navigation, trained with 0% moving objects (top) and 50% moving objects (bottom), running in a test setting where 100% of objects are moving. Row 3 shows ground-truth bounding boxes in white, predicted bounding boxes in color. In the top-down view shown in Row 4, ground-truth object locations are represented by stars, while predicted objects are represented by circles. The transparency for the bounding boxes in Row 3 and the fill of the circles in Row 4 is proportional to each object's *pres* value. Color indicates object identity, and colors of the bounding boxes in Row 3 match up with the colors of the circles in Row 4.

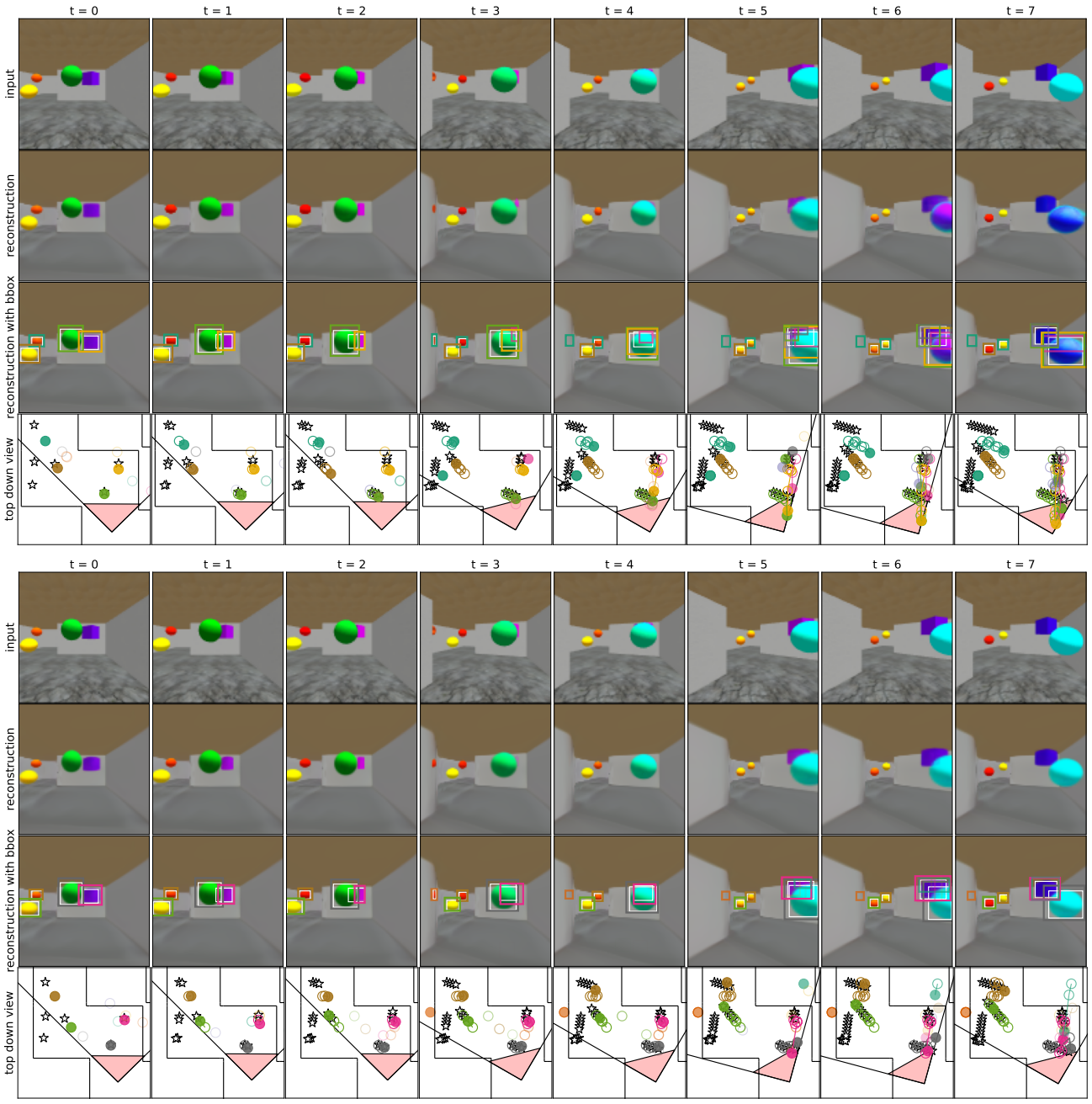

Figure 6.10. Same interpretation as Figure 6.9; see the caption of that figure for details.

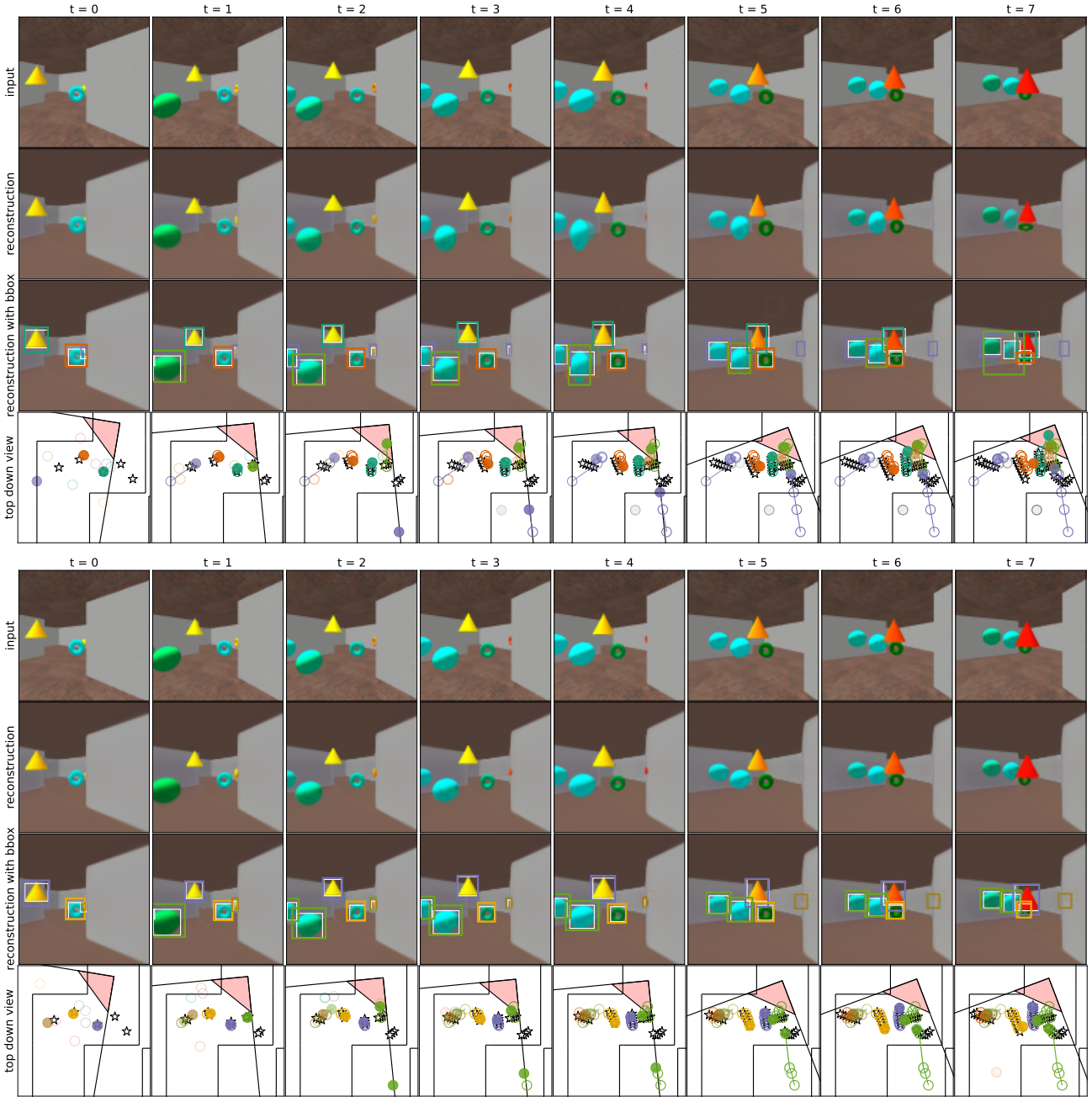

Figure 6.11. Same interpretation as Figure 6.9; see the caption of that figure for details.

which it is trained. In the rest of this section we discuss a number of different aspects of 3DOM, including limitations, possible alternatives to some of the design choices we have made herein, aspects of the architecture that could be improved in future work, and new research questions opened up by 3DOM.

6.8.1 Depth Ambiguity and Phantom Object Motion

As mentioned in our discussion of the results from running 3DOM in the First-Person Maze Navigation environment, one of 3DOM's biggest limitations is its inability to estimate the depth of objects that are only partially in view. In this section we discuss the reasons for this depth ambiguity, some of its negative knock-on effects, and how it might be remedied in future iterations of 3DOM.

Situations in which the depth of an object is fundamentally ambiguous are an inherent property of the monocular (i.e. single camera) visual setting we have chosen to work in. In the alternative setting of binocular vision (also called stereo vision), where at each timestep we have access to two images from spatially separated cameras, we can (usually) compute the spatial disparity between various visual features in the two images in order to compute depth at every location in the image [77]. However, in the monocular case, where we only have access to a single image at each timestep, there is no direct way to compute depth from visual features. Instead, the network must use a more indirect strategy for estimating object depth, which involves memorizing (i.e. learning) a mapping from an object's class and apparent size to a depth value (the training signal for learning this mapping comes from the loss function outlined in Section 6.5.7). Ambiguities in object depth thus occur whenever the object class or apparent size are themselves ambiguous, as will often be the case when an object is only partially visible.

For example, one of the shapes in each Maze Navigation dataset is a magenta cone. Suppose the agent sees a frame with the corner of a magenta cone in view, with the majority

of the cone off-camera. Because the object has a solid color[6], the visible corner does not provide enough information for us to determine the apparent size of the object: the corner could be a small section of a nearby cone, or a large section of a distant cone. This is further exacerbated in our case by the fact that we have included objects with the same color but different sizes. For example, the set of objects might contain both a small magenta cone and a large magenta cube; in this case, an observed magenta corner could belong to either a small cone or a large cube, adding object class ambiguity to the apparent size ambiguity.

This depth ambiguity causes the network to misestimate depth, a phenomenon which, besides being unsatisfactory in its own right, has a pernicious knock-on effect: as the object moves further into view and the object class and apparent size become unambiguous, the network is often able to correct its depth estimate. This creates illusory or "phantom" object motion in the depth direction: it seems, to the network, that the object is *moving* from a position corresponding to the initial (incorrect) depth estimate, to a new position corresponding to the correct depth estimate (in addition to any real motion that occurs as the object comes into view). This is problematic because the estimated object trajectories are used to train the Prior Propagation module (as described in Section 6.5.8), and the phantom motion is thus likely to create serious difficulties in training that module.

Since depth ambiguity is inevitable in a monocular setting, and creates serious issues for training the Prior Propagation module, future work should look to equip 3DOM with better ways of handling that ambiguity. One important step may involve retroactively correcting our depth estimates: once the object comes fully into view and we can form an accurate estimate of both its class and apparent size (and hence its depth), we could then go back and use this information to correct the depth estimates we made when the object was partially out of view. We could then insist on training the Prior Propagation module on only object trajectories corrected in this manner.

Another approach would involve making use of more complex distributions over object depth and 3D position. In practical terms, depth ambiguity translates into complex and

[6]If the object were textured, the network could conceivably use the scale of the texture as a clue about apparent size without needing to see the whole object.

often multimodal posterior distributions over depth. However, the current version of 3DOM uses a unimodal (Gaussian) distribution to model both depth and 3D position, and thus lacks the flexibility to properly model ambiguous cases. Future work could thus modify the model to make use of more sophisticated classes of posterior distribution capable of handling multiple modes, such as a Gaussian Mixture Model [12]. One complication that will have to be addressed when making such a switch, however, is the fact that the network makes explicit use of the mean value of the unimodal Gaussian distribution over depth and 3D position in order to make a guess about where the object will end up in the next frame, and it is not immediately clear what a multimodal analog of this computation would look like.

6.8.2 Handling Stochastic Objects

Stochasticity is an inherent part of modeling video in complicated domains. In many cases, for any given history of observations, the distribution over future trajectories has a high degree of uncertainty. For example, consider a video of a pedestrian approaching a traffic intersection: up until the pedestrian gets to the intersection and makes a choice about which direction to go, there are multiple plausible futures (e.g. at least one future for each of the paths making up the intersection). Thus, when making predictions multiple steps into the future, the network has to allow for stochasticity. This is typically done by including latent variables in the model (i.e. giving the network an intrinsic source of stochasticity to mirror the stochasticity observed in the world), and this is one of the reasons that it is popular to formalize world models as temporal Variational Autoencoders (as described in Section 3.2).

In OOWMs the problem of stochasticity looms even larger than in standard world models, because OOWMs typically include latent variables that are interpretable and correspond to real properties of objects in the world. For example, all models presented in this thesis include latent variables that model the position of objects. This is problematic because such variables will often take on complex, multimodal distributions. For example,

in the pedestrian example given earlier, an OOWM would have a variable explicitly modeling the pedestrian's position, a variable which will take on a complex and multimodal distribution as soon as the pedestrian reaches the intersection (at which point multiple branching paths are possible). In world models with less structure, the latent variables are not required to correspond to real-world variables and thus the network may find a way to get away with simple unimodal distributions such as Gaussians. In this thesis we have largely stuck with unimodal distributions, even for interpretable latent variables, but more recent work shows the problems that can arise from doing that, and how multimodal distributions may be incorporated to fix those issues [108].

In the 2D environments considered by most previous OOWMs, it is typically assumed that all objects stay in sight for the duration of the video, and so the multimodality of object position is complex but manageable. However, in the 3D setting considered in 3DOM, in which we may be viewing only a small part of the environment at any given time, the multimodality of object position becomes much harder to deal with. To see why, first note that the partial observability of 3DOM's setting means that objects may pass in and out of sight, either through occlusion or from simply moving off camera. Moreover, when an object passes from being visible to hidden, and then later from hidden to visible, we would like for 3DOM to be able to *re-identify* the object (i.e. to acknowledge that object from the first sighting *is the same as* the object from the second sighting).

In order to perform this re-identification, 3DOM needs to maintain a distribution over the possible locations of the object during the period it is out of sight. We can illustrate this with an example. Imagine that a batter in a baseball game hits a home run, sending the ball out of the park and out of sight. Seconds later, the umpire hands the catcher a ball so that play may resume. Is the ball in the catcher's glove the same as the home run ball? Intuition tells us that they cannot be the same ball, even if they look identical, because our internal distribution over the position of the (hidden) home run ball assigns low probability to it being in the umpire's possession, given the manner in which it passed out of sight and the small amount of time that has elapsed. In short, the distribution over a hidden object's

position is highly relevant for deciding whether some newly-appeared object is the same as some object that previously moved out of sight.

The real difficulty comes from the challenge of maintaining a distribution over the position of a stochastic hidden object over multiple time steps. Imagine trying to model all the possible trajectories of the home run ball after it passed out of sight: it could have fallen down a sewer, smashed through a car window, been picked up by any number of different people (each of whom would take it to a different location), etc. It is not clear how to scalably model such a complex distribution over positions; it is likely that significant approximations will need to be used. One possibility would be to use a non-parametric method similar to a particle filter [36], approximating the complex distribution over object position by keeping track of a relatively small collection of high-probability object positions at each timestep.

6.8.3 Alternative Models of Object Motion

Our Prior Propagation module is very flexible, and is essentially a recurrent neural network that repeatedly predicts the next step of object motion based on the history of the object. However, because of phenomena such as the "phantom" object motion caused by depth misestimation (discussed in Section 6.8.1), training this Prior Propagation module can be difficult. For example, it may take a long time for the Prior Propagation to even begin learning anything useful, since it will not have useful data to train on until the posterior Propagation module reaches a certain level of competence (since Prior Propagation effectively uses posterior Propagation as a source of training data; see Section 6.5.8).

One way to avoid these issues would be to significantly reduce the complexity of the Prior Propagation's object motion model, perhaps to something with no learned parameters at all, such as the constant velocity or constant acceleration motion models commonly used when applying standard Kalman filters to object tracking [27]. If we had reason to expect that the objects would adhere to such simple motion models, then we could reap significant benefits from switching to such a scheme. For example, we could

expect such models to make good predictions about how objects will behave when out of sight right from the start of training. During development we briefly experimented with this approach; however, we ultimately settled on the more flexible, learned approach to ensure that the model can learn to model arbitrary object motion. Future work should explore ways of obtaining the best attributes of both approaches.

6.8.4 Modeling 3D Object Shape

In this chapter, we have largely restricted ourselves to modeling 3D object *position*. 3D position is important as it allows us to predict where an object is likely to occur in subsequent images. However, we have not properly modeled the 3D *shape* of objects, something that should be remedied in future work (and *has* been addressed, to an extent, though in simpler and/or more heavily supervised scenarios than we consider here [37, 122]). By a proper 3D model of object shape, we mean a model which views an object as being constructed of various colored parts with definite locations in 3D space. One example would be modeling each object as a 3D mesh, as is often done in computer graphics [4]. Another example is the Scene Representation Networks (SRN) [143] that 3DOM uses to model the static 3D environments (i.e. have a separate SRN for each object).

In 3DOM, rather than maintaining a proper 3D model for an object, we instead predict both the appearance and apparent size of the object conditioned on both the object's *what* attribute and the orientation of the camera with respect to the object[7]. We used this method for several reasons. First, it allowed us to ease into the realm of 3D, focusing on the challenge of modeling 3D object position, leaving for future work the complications introduced by using a proper model of 3D shape. Second, it allowed us to stay as close as possible to SILOT, introduced in Chapter 5 (to an extent 3DOM may be viewed as simply a version of SILOT that is aware of the camera and the 3D positions of objects), allowing us to reuse parts of SILOT that worked well. Finally, it is not clear how to train a proper

[7]Interestingly, this ends up being similar to a technique from computer graphics called *billboarding*, a relatively simple approach to modeling object shape and appearance whose computational efficiency made it attractive for modeling characters and objects in early 3D computer games [4].

3D model of objects in a monocular (single camera) setting when one also wants to allow objects to rotate and change appearance over time. Learning a proper 3D model of an object or scene typically requires at least being able to see it from multiple viewpoints during training, without the object moving [166] (learning shape from a single view of an object has been done [71], but only for untextured objects for which surface shading provides a strong indication of object shape). For example, the reason we are able to train SRN models for our 3D environments is that we assume that the environment is static and that we are able to see it from many different viewpoints (we visit each environment multiple times, and the agent moves around throughout each episode). However in the case of objects, since we want to allow objects to move and change appearance over time, we can never be guaranteed to have seen an object from multiple viewpoints. We will of course have instances where we encounter an object on subsequent timesteps, often with different camera positions; however, because objects are dynamic, there is no guarantee that the object did not rotate or change its appearance in the intervening timestep. Fixing this in the future may involve simply making stronger assumptions about what changes an object can undergo between timesteps.

One important benefit of true 3D models is that they allow *novel view generation* (e.g. [47]). In the SRN framework, for example, an SRN trained for a given environment can later be rendered from arbitrary viewpoints, including viewpoints not present in the training data; this is possible because an SRN is a 3D model that assigns features to each point in 3D space, and creates images using a differentiable processing that mimics computer graphics rendering algorithms. The ability to render scenes from arbitrary viewpoints is an ability that would be useful for a number of purposes; for example, it would allow us to imagine how a given seen looks from the perspective of another agent, allowing us to draw conclusions about what information is available to the agent (e.g. whether a certain object is visible to them). Indeed, this is an ability well established in humans, under the name *visual perspective taking* [62].

6.9 Retrospective

At the time of writing, the work presented in this chapter was brand new, and so we save our retrospective analysis for future work.

Chapter 7

Conclusion

7.1 Summary of Contributions

This thesis makes several contributions to the study of Object-Oriented World Models (OOWMs). The purpose of Chapter 3 was to motivate and formalize the concept of OOWMs, and to identify a number of aspects of OOWM performance that have been neglected in past work, thereby providing motivation for models presented in later chapters. We started by pointing out that world models are of fundamental importance to Model-Based Reinforcement Learning, a general framework for building agents whose performance is adapted to a particular environment [65, 66, 67, 90]. We then provided a formalization of world models as temporal Variational Autoencoders (VAEs). Next, we argued that since the physical world is made up of discrete objects, it will often be beneficial to employ world models that have an explicitly object-oriented internal representation, similar to the way that humans use "object files" to mentally represent objects in the world [89]. We then showed how to modify the formulation of standard world models to use object-oriented representations and computations, resulting in the framework of *Object-Oriented* World Models. Finally we conducted a literature review assessing past work in the OOWM space, which allowed us to identify several ways in which existing OOWMs might be improved: they are unable to handle scenes containing large numbers

of heavily overlapping objects, and they do not model objects as existing in 3D space, instead modeling objects as existing on the 2D image plane. The goal of improving these two aspects of OOWM performance served as motivation for the work presented in later chapters.

In Chapter 4 we made progress on the project of building OOWMs capable of handling large numbers of objects. Rather than directly attacking the problem of building a scalable OOWM for video, we instead set our sights on the more modest goal of building a scalable OOWM for *images*; this allowed us to avoid certain challenges that arise when considering video, such as scalably modeling object dynamics. We started by considering Attend, Infer, Repeat (AIR) [42], an early image-only OOWM which, while promising, has difficulty handling images containing more than a few objects, and identified a number of aspects of AIR's design that contribute to its scaling issues. We then proposed a new architecture, called Spatially Invariant Attend, Infer, Repeat (SPAIR), which takes inspiration from supervised object detection networks [51, 52, 111, 130, 131, 132, 134] in order to scalably extract object-oriented representations from images. In particular, SPAIR employs a strategy that we call Spatial Divide and Conquer (divide the image up into patches, apply an identical object detector to each patch independently), implemented in a computationally efficient manner using convolutional neural networks [103]. We performed a number of experiments demonstrating SPAIR's improved scalability in several different contexts. First, we showed that SPAIR outperforms AIR by a significant margin when modeling images containing large numbers (up to 9) of overlapping MNIST digits, and that SPAIR is able to generalize well to images containing larger numbers of digits than seen during training. Next, we showed that SPAIR is able to extract objects in crowded scenes with enough fidelity to realize improved performance on downstream tasks. Finally, we showed that SPAIR scales well enough to be able to extract objects in the Space Invaders, an Atari game which often has 10s of objects on-screen simultaneously [10]. Overall we found SPAIR to be a significant improvement over AIR, and we suggest that it should be used as a default for future work in this area. One limitation of SPAIR

is that its performance is somewhat dependent on using a well-chosen patch size. In the future this may be remedied by performing detections at multiple scales, effectively using a range of patch sizes; such an approach has been used with great success in supervised object detection [111, 132].

In Chapter 5 we addressed the challenge of designing a scalable OOWM for video, building on the progress made for the image-only setting in Chapter 4. We started from Sequential Attend, Infer, Repeat (SQAIR) [98], an OOWM that took core concepts from AIR and applied them to videos. SQAIR divides inference into a Discovery module, responsible for discovering new objects in images (i.e. detecting objects), and a Propagation module, responsible for updating existing object representations as new video frames become available (i.e. tracking objects). Using this architecture, SQAIR is able to learn to discover, detect and track objects in videos containing small numbers of objects. However, it inherits scaling issues from its predecessor AIR, struggling to segment objects in crowded, many-object videos. We thus proposed Spatially Invariant Label-free Object Tracking (SILOT), an architecture which improves upon SQAIR using a number of different strategies including the Spatial Divide and Conquer strategy from SPAIR, Discovery and Propagation modules that parallelize across objects, and scalable inter-object conditioning using spatially local attention. We empirically demonstrated SILOT's improved scalability in a number of experiments. First, we showed that SILOT is significantly better than SQAIR at learning to track objects in videos consisting of moving, densely packed MNIST digits (up to 12 digits of size 14×14 squeezed into a 48×48 image, often with significant overlap). Secondly, in an environment consisting of moving colored shapes, we showed that not only is SILOT capable of learning to track 10s of objects simultaneously, it also generalizes well to videos both larger and containing more objects than videos seen during training. For example, we showed that a network trained on cropped versions of videos containing 10 objects was able to achieve competitive performance on full (i.e. uncropped) videos containing up to 35 objects. As a final test, we showed that SILOT was able to discover and track objects in Space Invaders and Asteroids, two Atari games that often have many 10s of objects

on-screen at once [10]. In the end we found SILOT to be a concrete improvement over SQAIR, and a significant step forward in the study of scalable OOWMs.

In Chapter 6 we switched gears away from scalability issues, focusing instead on building an OOWM that properly models objects as existing in 3 dimensions. We started by noting that most existing OOWMs model objects as 2D entities on the 2D image plane and have no concept of 3D space [16, 28, 29, 40, 42, 59, 60, 70, 85, 98, 109, 158, 160, 162, 169], even when they are applied to images or videos captured in 3D environments. This state of affairs contrasts starkly with human object-oriented representations, which appear to view objects as 3D entities existing in a 3D world [141]. We argued that properly modeling the 3D nature of objects has a number of advantages for humans, such as enabling object permanence [7] and providing a foundation for intuitive physics [6], and that making the jump to 3D would endow OOWMs with similar advantages. We thus proposed the 3D Object-Oriented World Model (3DOM), an OOWM with similarities to SILOT but which properly models the position of objects in 3D space. In order to model the non-object elements of the 3D scenes (e.g. floors, walls, ceilings, sky), we employed Scene Representation Networks (SRN) [143], a framework which enables learning of implicit representations of static 3D environments from posed videos thereof. In a number of experiments we demonstrated that 3DOM is able to learn to discover objects and their position in 3D space, and that doing so provides a number of important advantages. First, we showed that 3DOM is able to learn representations of objects in a simulated bin picking scenario where all objects are static. 3DOM exhibited markedly improved performance over competing methods that lacked a proper 3D representation of objects. Second, we showed that 3DOM is able to discover and track moving objects, and performs well even when objects are *always* moving (a situation that makes depth estimation via epipolar geometry rather difficult). Finally, we showed that 3DOM transfers well to new environments for which a trained SRN was not available. In conclusion, we consider 3DOM to be an important step toward building OOWMs that can be useful in real-world environments. Of course it is *only* a step, and there is a great deal of room for improvement

in this direction; for example, 3DOM only models the 3D positions of objects, ignoring other important 3D attributes such as shape, orientation and scale.

7.2 Limitations

The overall goal of research on OOWMs is to build systems capable of autonomously learning high-quality object-oriented representations and models in novel environments. One aspect of current OOWMs that stand in the way of this goal is their relatively large number of hyperparameters. Some of these hyperparameters have reasonable default values which work well on a wide range of datasets, and so are not problematic. However, all other hyperparameters need to be chosen either by a hyperparameter search or by some combination of hand-tuning and prior knowledge of the dataset. A hyperparameter search requires access to a small annotated validation dataset, which can be constructed in a few minutes of effort by a layperson. However, the curse of dimensionality (i.e. the fact that the number of hyperparameter combinations that must be considered is an exponential function of the number of hyperparameters) severely limits the number of parameters that can be chosen in this way given finite computational resources. Any remaining parameters must be chosen by hand-tuning and/or prior knowledge of the dataset, requiring (possibly extensive) effort from a skilled machine learning practitioner. Therefore the goal of fully autonomously building object-oriented models for arbitrary new environments is still far off. Future work should emphasize building models with small numbers hyperparameters, or at least ensuring that all hyperparameters have robust default values.

The large number of OOWM hyperparameters also presents issues when comparing the performance of different models. In principle, when comparing two models on a given dataset we should search over all hyperparameter combinations for both models. However, in practice such a search will usually be computationally infeasible, and many parameters for both models will have to be hand-tuned. This leaves open the possibility that there could be some combination of the hand-tuned parameters that yield better performance.

This situation is exacerbated by the fact that the person performing the comparison may have different levels of familiarity with the two models in question, making them more adept at hand-tuning the performance of one or the other. One way to mitigate this issue is to simply include it in our understanding of the model comparison: if a model can only be successful with a very specific, hard-to-find, dataset-dependent parameter combination, then that should count against the model, as the existence of a performant combination of hyperparameters is of little consequence if it cannot be found. That said, it may also be worthwhile to be more explicit about sensitivity to hyperparameters in future work; for instance, it would be advantageous for future OOWMs to provide sensitivity analyses indicating the difficulty of locating performant hyperparameter combinations for new datasets [81].

7.3 Open Questions

This thesis has made progress on the goals of building OOWMs that can scale to scenes with large numbers of objects and that properly model objects as existing in a 3D environment. However, it also leaves open a number of questions to be addressed by future work.

Recall that we formulated OOWMs as temporal VAEs; a consequence of this choice is that the main signal for training an OOWM is derived from its ability to reconstruct perceptual data based on its internal object-oriented representation of the world. One problem with relying on reconstruction as a training signal is that it gives different priority to objects of different sizes: a failure to model a large object creates much more reconstruction loss than a failure to model a small object, since the former occupies more pixels than the latter. This phenomenon can result in models which ignore small but semantically important objects, such as the ball in the game Pong [10, 90]. Therefore an important project for future work will be to find ways to mitigate this issue, perhaps by identifying alternate signals that can be used for training OOWMs. One possibility is contrastive learning, which simply optimizes the representation at one timestep to be predictable from the representation at

a previous timestep, and thereby omits reconstruction altogether; this strategy has seen some initial success, though only in relatively simple environments [96]. Another possible way forward is to make use of more sophisticated image dissimilarity functions that do more than simply compare pixels. One promising approach along these lines would be to *train* a loss function using adversarial methods [56]; for example, VAE-GAN shows how to incorporate an adversarially learned loss function into the VAE framework [102].

Another issue for future research will be refining both our definition of what constitutes an object and our notion of an object-oriented representation. In this thesis we have often talked about finding *the* way to decompose a scene into objects; however, it is fairly clear that object-oriented decompositions are not unique, and that most physical scenes can be decomposed into objects in a number of different ways. For example, in this thesis we have roughly defined objects to be "maximal clusters of strongly connected atoms". However, even this relatively straightforward (and likely overly simplistic) definition leaves open the possibility of multiple object-oriented decompositions of a given physical scene, by way of ambiguity in the phrase "strongly connected": how strongly do two sets of atoms have to be connected to one another for us to view them as belonging to a single object? Does glue create a strong enough bond? How about a sewn seam, or a weld? We do not necessarily need to provide concrete answers to these questions here; the point is just that any given physical scene may admit multiple different object-oriented decompositions. Moreover, different decompositions may be more or less useful for different tasks. For example, for most humans most of the time, a complex machine like a car or a computer can be usefully viewed as a single object. However, for an engineer working to fix such a machine, it will often be necessary to descend to a lower level at which the machine is seen to be composed of many thousands or millions of different physical parts. The ideal solution would then be to learn a *hierarchy* of object-oriented representations, with detailed, fine-grained decompositions at lower levels and abstract, coarse decompositions at higher levels, and permit the agents making use of these representations to move up and down the hierarchy to suit their current information processing needs [61].

Even once the definitions of objects and object-oriented representations have been fully worked out, future work will still need to make significant additional progress on the question of how to learn the object-oriented representations from perceptual data, as there are significant issues with our current strategies. Recall that in Chapter 3 we motivated the idea of an object-oriented decomposition of an environment using the concept of *modularity*: a modular decomposition is one in which the components have internal structure, are independent of one another, and tend to recur in different contexts [61]. Many image-only OOWMs, including AIR [42], SPAIR (Chapter 4), MONet [16] and IODINE [59], discover objects by, in effect, optimizing for modular decompositions of the images in the training dataset [61]. An assumption implicit in this approach is that a modular decomposition of an *image* of a physical scene will necessarily correspond to a modular (and thus object-oriented) decomposition of the physical scene itself. This assumption may be valid for the relatively simple datasets that OOWMs have most often been tested on in the past, in which each object usually has a single color and texture; however, in real world scenes individual objects can have arbitrary color and texture variations which may cause this correspondence to be violated [118, p. 270]. A vivid demonstration of this issue was shown when IODINE was trained on ImageNet, a dataset of real-world images [138]: IODINE's strategy of optimizing for a modular decomposition of images caused it to learn to segment images into regions of constant color, frequently dividing individual physical objects into multiple parts [59].

Making use of video rather than static images can alleviate this issue to an extent, since video can provide more reliable information about the boundaries of physical objects: clusters of pixels that move together are likely to be strongly physically connected and, consequently, to belong to the same object [57, 145, 146]. However, video is only helpful to the extent that it actually shows objects moving: a video of a still-life tableau is no more informative than an image. The strategy of using motion as an indicator of object boundaries will therefore struggle at discovering objects which rarely move. Future work

should therefore look for additional properties of objects, beyond modularity and motion, that can act as cues for discovering objects in perceptual input.

In 3DOM (Chapter 6) we made progress on discovering objects in the kinds of scenarios that would present difficulties for motion-centric strategies. As an example of such as situation, consider a kitchen in an average household: most of the objects therein (cutlery, dishes, appliances) will be stationary most of the time (the kitchen will only be in use for a small fraction of each day). A motion-based strategy would have a difficult time here, as it could only learn from those relatively rare moments in which the objects were in use. In 3DOM we made progress by assuming that an agent is learning about a 3D environment (e.g. the kitchen), and that it is able to visit that environment on multiple different occasions (e.g. different hours, days or weeks). The agent then fits a static 3D model to all the data gathered from the environment, and assumes that anything that this static model is not able to account for must be *contingent* and therefore an object. One concrete advantage of this approach is that it does not require the agent to actually witness object motion; instead, the agent just needs to visit the environment both before and after the object motion so that it can infer that the motion has taken place.

However, even 3DOM leaves room for improvement. For one, it may not always be possible to satisfy 3DOM's requirement that the agent visit the same environment on multiple occasions. For another, while 3DOM relieves the agent from the need to directly witness object motion, the agent still needs to wait for the motion to occur, which requires gathering data over long time spans for objects that move very infrequently. Therefore one promising path forward will be to insist on the ability to *interact* with the environment while gathering data. In particular, our goal should be to build systems that are curious about how a new environment divides into objects, are able to form hypotheses about what is and is not an object, and are able to make plans and take action in the world in order to test those hypotheses. This would enable agents to proactively force the environment to reveal how it decomposes into objects, rather than waiting for the environment to reveal this information on its own. It is well-established that human infants take a highly active

approach to learning about objects [136]. In machine learning, such a strategy would fall under the field of *active vision* [5, 8], and some initial progress on the specific task of active object discovery has been made in recent years [125].

Appendix A

Prior for *pres* Latent Variables

A.1 SPAIR

Here we provide a derivation for the prior on z^{pres} (a vector of length HW made up of all z_i^{pres}), discussed briefly in Section 4.4.4. Recall that in Section 4.4.2, we specified how the posterior network predicts a distribution over z_i^{pres}, samples from it, and then maps to the object variable o_i^{pres}:

$$z_i^{\text{pres}} \sim \text{Logistic}(\omega_i^{\text{pres}}) \,,$$
$$o_i^{\text{pres}} = \text{sigmoid}(z_i^{\text{pres}}) \,.$$

Here ω_i^{pres} is a value output by a neural network, z_i^{pres} is a Logistic random variable, and o_i^{pres} can be interpreted as a BinConcrete random variable. It is possible to define a prior distribution over either z_i^{pres} or o_i^{pres}. However, both of these spaces are somewhat difficult to interpret.

To define a prior distribution in a more interpretable space, we first note that the variable that we get by *rounding* o_i^{pres}, i.e. $\overline{o}_i^{\text{pres}} = \text{round}(o_i^{\text{pres}})$, has a Bernoulli distribution where the probability of the positive class is $\text{sigmoid}(\omega_i^{\text{pres}})$. Thus ω_i^{pres} can be viewed as controlling

this distribution over discrete values. For SPAIR we define our prior distribution in the discrete space of Bernoulli variables.

Let $\overline{o}^{\text{pres}} = \text{round}(o^{\text{pres}})$ be the vector of discretized (i.e. rounded) object presence variables. We define a prior over $\overline{o}^{\text{pres}}$ as follows. Let C be a random variable giving the number of non-zero entries in $\overline{o}^{\text{pres}}$. Under the prior, $\overline{o}^{\text{pres}}$ is generated by first sampling C, and then drawing $\overline{o}^{\text{pres}}$ uniformly from all binary vectors that have C non-zero entries:

$$p(\overline{o}^{\text{pres}}) = p(C = nz(\overline{o}^{\text{pres}}))p(\overline{o}^{\text{pres}}|C = nz(\overline{o}^{\text{pres}})) \,,$$

where $nz(\cdot)$ gives the number of non-zero entries in a vector. There are $\binom{HW}{C}$ binary vectors of length HW with C non-zero entries, so:

$$p(\overline{o}^{\text{pres}}|C = nz(\overline{o}^{\text{pres}})) = \binom{HW}{nz(\overline{o}^{\text{pres}})}^{-1} .$$

For the distribution over C we use a Geometric distribution with parameter ψ. We use the Geometric interpretation that puts C as the number of failures before a success, and ψ as the success probability.

$$p(C = nz(\overline{o}^{\text{pres}})) = \psi(1-\psi)^{nz(\overline{o}^{\text{pres}})} .$$

By setting ψ to a high value, we put most of the probability mass at low values of C, resulting in a prior that puts pressure on the network to explain images using as few objects as possible. We then truncate and normalize to ensure C has support $\{0, 1, \ldots, HW\}$:

$$\begin{aligned} p(C = nz(\overline{o}^{\text{pres}})) &= \frac{\psi(1-\psi)^{nz(\overline{o}^{\text{pres}})}}{\sum_{c=0}^{HW} \psi(1-\psi)^c} \\ &= \frac{\psi(1-\psi)^{nz(\overline{o}^{\text{pres}})}}{\psi\frac{(1-(1-\psi)^{HW+1})}{1-(1-\psi)}} \\ &= \frac{\psi(1-\psi)^{nz(\overline{o}^{\text{pres}})}}{1-(1-\psi)^{HW+1}} . \end{aligned}$$

Overall, the prior is:

$$p(\overline{o}^{\text{pres}}) = \frac{\psi(1-\psi)^{nz(\overline{o}^{\text{pres}})}}{(1-(1-\psi)^{HW+1})\binom{HW}{nz(\overline{o}^{\text{pres}})}} \,.$$

A.2 SILOT

For SILOT we use a formulation identical to SPAIR.

A.3 3DOM

In 3DOM we opted for a simpler approach to defining a prior over presence, which trades the intuitive appeal of defining a prior in the space of discrete Bernoullis for a prior in the space of continuous BinConcrete random variables which allows a much more straightforward computation of Kullback-Leibler (KL) Divergence.

To define a prior over presence latent variables, we instantiate a strictly positive function f of the latent variable vector, and then define the prior probability as:

$$P(z^{\text{pres}}) = f(z^{\text{pres}})/Z \,,$$

where $Z = \int f(z^{\text{pres}}) dz^{\text{pres}}$ is the normalization constant for the distribution. The main role of the prior distribution is to constrain and regularize the posterior distribution $Q_\phi(z^{\text{pres}}|x)$, and this is achieved by minimizing the Kullback-Leibler (KL) Divergence between the posterior and the prior as part of maximizing the variational evidence lower-bound (see Sections 2.3.1, 4.4.5 and 5.2.8). The KL Divergence is:

$$\begin{aligned} D_{KL}(Q_\phi(z^{\text{pres}}|x) \parallel P(z^{\text{pres}})) &= \int Q_\phi(z^{\text{pres}}|x) \log \frac{Q_\phi(z^{\text{pres}}|x)}{P(z^{\text{pres}})} dz^{\text{pres}} \\ &= \int Q_\phi(z^{\text{pres}}|x) \log \frac{Q_\phi(z^{\text{pres}}|x)}{f(z^{\text{pres}})/Z} dz^{\text{pres}} \\ &= \log Z + \int Q_\phi(z^{\text{pres}}|x) \log \frac{Q_\phi(z^{\text{pres}}|x)}{f(z^{\text{pres}})} dz^{\text{pres}} \,. \end{aligned}$$

Thus the normalization constant Z appears in the KL divergence as an additive constant which does not affect the optimization. We can thus avoid computing Z (which would be prohibitively expensive) by simply minimizing $D_{KL}(Q_\phi(z^{\text{pres}}|x) \parallel P(z^{\text{pres}})) - \log Z$ instead of $D_{KL}(Q_\phi(z^{\text{pres}}|x) \parallel P(z^{\text{pres}}))$. We have:

$$\begin{aligned}
&D_{KL}(Q_\phi(z^{\text{pres}}|x) \parallel P(z^{\text{pres}})) - \log Z \\
&= \int Q_\phi(z^{\text{pres}}|x) \log \frac{Q_\phi(z^{\text{pres}}|x)}{f(z^{\text{pres}})} dz^{\text{pres}} \\
&= \int Q_\phi(z^{\text{pres}}|x) \log Q_\phi(z^{\text{pres}}|x) dz^{\text{pres}} - \int Q_\phi(z^{\text{pres}}|x) \log f(z^{\text{pres}}) dz^{\text{pres}} \\
&= -H\left[Q_\phi(z^{\text{pres}}|x)\right] - E_{z^{\text{pres}} \sim Q_\phi(z^{\text{pres}}|x)}\left[\log f(z^{\text{pres}})\right] \ , \qquad \text{(A.1)}
\end{aligned}$$

where H is the entropy function. Thus minimizing $D_{KL}(Q_\phi(z^{\text{pres}}|x) \parallel P(z^{\text{pres}})) - \log Z$ can be interpreted as finding a distribution Q_ϕ that has high entropy and assigns high probability to values which function f assigns high values.

For the function f, we choose:

$$f(z^{\text{pres}}) = e^{-\psi \sum_{i=1}^{HW} \text{sigmoid}(z_i^{\text{pres}})} = e^{-\psi \sum_{i=1}^{HW} o_i^{\text{pres}}} \ .$$

When we substitute this into Equation (A.1), we get:

$$D_{KL}(Q_\phi(z^{\text{pres}}|x) \parallel P(z^{\text{pres}})) - \log Z = -H\left[Q_\phi(z^{\text{pres}}|x)\right] + \psi \cdot E_{z^{\text{pres}} \sim Q_\phi(z^{\text{pres}}|x)}\left[\sum_{i=1}^{HW} o_i^{\text{pres}}\right] \ ,$$

and the overall interpretation is that we penalize the 1-norm of the vector o^{pres}, where parameter ψ controls the strength of the penalty.

Appendix B

Experiment Details for SPAIR

B.1 Base Model Details

We start by outlining a base set of details for each model, including hyperparameter values, hyperparameter tuning procedure, and neural network architectures. In subsequent sections we provide additional details about how these are modified for particular experiments.

Throughout this section we describe the architectures of various neural networks that make up the larger models; in doing so, we will use the notation FC([$N_1, \ldots, N_L$], f) to refer to a Fully-Connected Network (FC) / Multi-Layer Perceptron (MLP; see Section 2.2.1) with L hidden layers, with the ℓ-th hidden layer having N_ℓ units, and making use of function f as the nonlinearity for all hidden layers (the final layer of an MLP is always assumed to use the identity function as its nonlinearity). For example, FC([128, 128], ReLU) is an MLP with 2 hidden layers, each having 128 units and making use of the ReLU nonlinearity.

B.1.1 AIR / DAIR

For our implementation of AIR/DAIR, we adapted code from `github.com/aakhundov/tf-attend-infer-repeat`. To backpropagate through the discrete z^{pres} latent variables, we replace them with Concrete variables [84, 114], rather than resorting to the

REINFORCE-style gradient estimator [164] used in the original AIR paper [42] (this is similar to what we have done for SPAIR, see Section 4.4.2).

For the majority of AIR's parameters the defaults were used, and these are expected not to have a significant impact on performance. As stated previously, AIR and DAIR were always provided with the true number of objects in the image (imparted to the network by hard-coding the number of recurrent steps), even at test time, which allowed us to avoid tuning AIR hyperparameters that control the prior distribution over number of objects (i.e. parameters analogous to the ψ hyperparameter for SPAIR, see Section A.1). For all datasets on which AIR was used, we performed random hyperparameter searches over AIR's analogs of $\mu^{h/w}$, $\sigma^{h/w}$, and $\sigma^{y/x}$.

B.1.2 SPAIR

SPAIR was implemented in tensorflow, and is available online at `https://github.com/e2crawfo/auto_yolo`.

To facilitate stability early on in training, we use a schedule, rather than a fixed value, for ψ (recall that ψ controls the prior over z^{pres}, see Section A.1). Early in training we use a value near 0 so that the network was not penalized for using many objects to explain the scene. Over the first few thousand updates, we anneal ψ to ≈ 1, which encourages the network to use few objects to reconstruct the scene. Without an initial period where the network is free to use as many objects as it wants, we found the network would become stuck in local minima where all objects are turned off. One possible explanation for this behavior is that early in training the network is not good at reconstructing objects, and can get a lower reconstruction loss by simply turning all objects off (i.e. setting all $o_i^{\text{pres}} = 0$). Note this trick was adapted from a similar trick used for AIR in the cited repository.

We define the evolution of ψ_t (where t is the index of the update step) over the course of the training run by scheduling its inverse log odds, $v_t = (1 - \psi_t)/\psi_t$, so that $\psi_t = 1/(1 + v_t)$.

Description	Variable	Value
Base bbox size	(A_h, A_w)	(48, 48)
Batch size		32
Grid cell scaling	κ	2
Grid cell size	(c_h, c_w)	(12, 12)
Dim. of z_i^{what}	N_{what}	50
Rendered object size	$(H_{\text{obj}}, W_{\text{obj}})$	(14, 14)
Learning rate		1×10^{-4}
Max gradient norm		1.0
Optimizer		Adam
Prior on z^h, z^w	$(\mu^{h/w}, \sigma^{h/w})$	(-2.2, 0.5)
Prior on z^y, z^x	$(\mu^{y/x}, \sigma^{y/x})$	(0, 1)
Prior on z^{depth}	$(\mu^{\text{depth}}, \sigma^{\text{depth}})$	(0, 1)
Prior on z^{what}	$(\mu^{\text{what}}, \sigma^{\text{what}})$	(0, 1)
Prior on z^{pres}	ψ	See Section B.1.2
Schedule for ψ	$(v_{\text{start}}, v_{\text{end}}, v_{\text{step}})$	(10^6, 0.0125, 10^3)
Appearance offset and scale	$(\mu^\beta, \sigma^\beta)$	(0., 2.)
Transparency offset and scale	(μ^ξ, σ^ξ)	(5., 0.1)

Table B.1. Hyperparameter values for SPAIR.

The schedule for the inverse log odds is given by:

$$v_t = v_{\text{end}} + 0.1^{t/v_{\text{steps}}} \cdot (v_{\text{start}} - v_{\text{end}}) \,,$$

where $v_{\text{end}}, v_{\text{start}}$ and v_{step} can be regarded as hyperparameters.

Additionally, for the first 1000 update steps we do not backpropagate gradients through *where*, *depth*, or *pres* attributes (so the networks that predict these attributes are not trained), another trick that prevents the network from falling into poor local minima where the objects are all turned off or shrunk into oblivion.

The base set of hyperparameters for SPAIR is given in Table B.1.2, while the base architectures of SPAIR's component neural networks are given in Tables B.2 and B.3. The majority of hyperparameters (including neural network architectures) were set to reasonable defaults or hand-tuned. In cases where hyperparameter searches were performed, we searched over values for $\mu^{h/w}, \sigma^{h/w}, v_{\text{end}}$ and v_{step}.

Type	# Filters	Filter Size	Stride	Nonlinearity
Conv	128	4	2	ReLU
Conv	128	4	2	ReLU
Conv	128	4	3	ReLU
Conv	128	1	1	ReLU
Conv	128	1	1	ReLU
Conv	100	1	1	None

Table B.2. Architecture for SPAIR's convolutional backbone network q_ϕ^{backbone}. No pooling is used at any point.

	Description	Architecture
q_ϕ^{backbone}	Computes bottom-up features for discovery units	See Table B.2
q_ϕ^{where}	Predicts params for posterior over z^{where}	FC([100, 100], ReLU)
q_ϕ^{obj}	Processes glimpse centered on object	FC([256, 128], ReLU)
q_ϕ^{what}	Predicts params for posterior over z^{what}	FC([100, 100], ReLU)
q_ϕ^{depth}	Predicts params for posterior over z^{depth}	FC([100, 100], ReLU)
q_ϕ^{pres}	Predicts params for posterior over z^{pres}	FC([100, 100], ReLU)
r_θ^{obj}	Predicts object appearances in rendering.	FC([128, 256], ReLU)

Table B.3. Component neural networks in SPAIR. FC([N_1, N_2], ReLU) is a sequence of 3 fully-connected layers (2 hidden layers with N_1 and N_2 units, respectively, and one output layer). The ReLU non-linearity is applied only at the hidden layers, the output nonlinearity is always the identity function.

B.1.3 ConnComp

See Section 5.1 of the main paper for a description of the ConnComp algorithm. One required step is finding the connected components of a binary image; this was performed using the function `tf.contrib.image.connected_components` from tensorflow 1.8. ConnComp has a single parameter τ serving as a threshold: a pixel is considered non-background if and only if the mean absolute difference between the pixel value and the background value is at least τ. τ was set by doing a grid search, using the value that yielded the best performance on the validation set.

B.2 Experiment Details

Here we provide additional details on each experiment.

B.2.1 Comparison with AIR

We used a single set of hyperparameters for SPAIR for all training conditions, and these hyperparameters were obtained by a hyperparameter search on a dataset containing images that contain between 1 and 9 digits (a mixture of all training conditions). For AIR we tuned the hyperparameters separately for each training condition (recall that each training condition uses images containing a different number of digits). Networks were trained using early stopping: networks were trained until a certain number of update steps have passed without an improvement in the training loss. The required number of steps is called the *patience*, and here it was set to 5×10^4 steps.

B.2.2 Generalization

For SPAIR we used the same hyperparameters as were found in the comparison with AIR. Networks were trained with early stopping and a patience of 5×10^4 steps, with a hard upper limit of 2×10^5 training steps.

B.2.3 Addition

For SPAIR we used the same set of hyperparameters as were used in the comparison with AIR. For AIR we used the hyperparameters found for the training condition containing 5 digits (since the images in this experiment contain 5 digits each).

For TrueBB and ConnComp, each of the object patches (ground-truth patches in the former case, patches found through the Connected Components algorithm in the latter case) were first processed by an MLP identical to SPAIR's object encoder network q_ϕ^{obj},

with architecture FC([256, 128], ReLU). For ConvNet, the convolutional architecture was identical to SPAIR's convolutional backbone q_ϕ^{backbone}.

In all cases, the downstream classifier was structured as follows. First, an MLP with architecture FC([100, 100]) processed the network's internal representation of each object independently; for AIR and SPAIR the internal object representations are the o^{what} vectors, for TrueBB and ConnComp this is the output of the object encoder MLP, and for ConvNet we treat each spatial location in the output volume as representing a separate object. Next the processed representations were fed into an LSTM [75] with 128 hidden units that iterated over objects. Finally the output of the LSTM at the final hidden layer was passed into another MLP with architecture FC([100, 100]), and the output of this MLP was interpreted as a set of classification logits.

For TrueBB, ConnComp, and ConvNet, we initially experimented with a 2-stage training process similar to the one used for AIR and SPAIR. In the first stage, the object encoder network and/or convolutional network would be trained to reconstruct input images (ignoring the classification labels), thereby hopefully learning a useful representation of the input images. However, we found this that this approach yielded negligible performance improvement, and thus for these 3 methods we stuck to the simple case of training all network weights directly on the classification problem.

All networks on all stages were trained with the ADAM optimizer, learning rate of 1×10^{-4}, maximum gradient norm of 1., and a batch size of 32. For AIR and SPAIR, the first stage was run with patience of 5×10^4 steps and a hard maximum of 3×10^5 steps. For all methods, the second stage was run with a patience of 1×10^4 steps and a hard maximum of 2×10^5 steps.

B.2.4 SET

For the SET scenario, we experimented with using a slightly more complicated background configuration than the solid black backgrounds used for the MNIST data. In particular, we gave each image in the dataset a random colored background, either cyan, magenta

or yellow. In order for SPAIR to be able to handle this more complicated setting, we had to give it a means of modeling the background. In particular, we instantiated a *background encoder network* q_ϕ^{bg} with architecture FC([10, 10]), which maps from the input image to an RGB value. This RGB value is then tiled to yield a monochrome image of size $(H^{\text{img}}, W^{\text{img}}, 3)$, used as the background image x^{bg} (see Section 4.4.3). The weights of q_ϕ^{bg} were trained alongside the rest of the SPAIR network (using the reconstruction loss as the training signal).

We did not use ConnComp as a baseline in this setting because the objects were multi-colored (each object has a number of black strokes on its face to indicate the "number" attribute), and ConnComp would consequently have done a poor job of segmenting objects. Both TrueBB and ConvNet were trained in the same manner as in the Addition experiment. SPAIR was trained using a two-stage process similar to the Addition experiment, and all weights of SPAIR's encoder network were kept fixed during the second (classification) stage, corresponding to the *Fixed* variant from the Addition experiment. The downstream classifier was identical to the one used in the Addition experiment, except that it contained 256 hidden units instead of 128.

For SPAIR, the first stage was run with patience of 5×10^4 steps and a hard maximum of 3×10^5 steps. For all methods, the second stage was run with a patience of 2×10^4 steps and a hard maximum of 2×10^5 steps.

Appendix C

Experiment Details for SILOT

C.1 Base Model Details

C.1.1 Training Details

Here we outline several details of training that are common to both SQAIR and SILOT. For both models we train using a curriculum, starting training on the initial 2 frames of each video, and increasing by 2 frames every N_{curric} timesteps until the network is training on full videos. Once the network is training on full frames, we begin an early stopping regime wherein we train until validation loss does not improve for 3×10^4 training steps (i.e. using a "patience" value of 3×10^4), triggering an early stop. Each time an early stop is triggered, we go back to the set of weights that has the best performance so far, divide the learning rate by 3, and resume training. Training ends once early stopping has been triggered 3 times. We take as our final hypothesis the set of weights that achieved the best performance on the validation set.

C.1.2 SQAIR

For our implementation of SQAIR, we used a lightly modified version of the original implementation: `https://github.com/akosiorek/sqair`. Initial tuning of hyper-

Type	# Filters	Filter Size	Stride	Nonlinearity
Conv	128	4	3	ReLU
Conv	128	4	2	ReLU
Conv	128	4	2	ReLU
Conv	128	1	1	ReLU
Conv	128	1	1	ReLU
Conv	128	1	1	None

Table C.1. Architecture for SILOT's Discovery backbone network d_ϕ^{backbone}. No pooling is used at any point.

parameters was performed by hand, starting from the default values provided in the repository, until a reasonable set of values for the Scattered MNIST dataset was identified. As model-selection criteria, we used the MOTA (a measure of object tracking performance) averaged over 1–6 digits for the 1–6 training case, and averaged of 1–12 digits for the 1–12 training case. We ran an additional grid search over select SQAIR hyperparameters that control the how the network learns about the number of objects in a scene.

C.1.3 SILOT

SILOT was implemented in tensorflow, and is available online at `https://github.com/e2crawfo/silot`. The base set of hyperparameters used in our experiments are listed in Table C.2. SILOT contains a relatively large number of component networks. Table C.3 lists these networks, along with descriptions of their role and architecture. The architecture of the backbone convolutional network d_ϕ^{bu} is listed in Table C.1. Note that SILOT's discovery network is very similar to SPAIR's encoder network, and thus in training SILOT we make use of the same set of tricks used in SPAIR, described in Appendix B.1.2.

C.1.4 ConnComp

We compare against a simple baseline algorithm called ConnComp (short for Connected Components); this is similar to the ConnComp algorithm used in the SPAIR experiments (see Section B.1.3), except that it requires added machinery for tracking objects over time.

Description	Variable	MNIST	Shapes	Atari
Learning rate		1×10^{-4}		
Batch size		16		
Max gradient norm		10.0		
Optimizer		Adam		
Patience		3×10^4	3×10^4	1×10^5
# steps per curriculum stage	N_{curric}	4×10^4		
Probability of discovery dropout	p_{dd}	0.5		
Number of propagated/selected objects	K	16	36	36
Dimension of $\overline{\mathcal{D}}_{(t)}^{\text{what}}$ and $\overline{\mathcal{P}}_{(t)}^{\text{what}}$	N_{what}	64		
Base bbox size	(A_h, A_w)	(48, 48)		(24, 24)
d_ϕ^{backbone} receptive field size		(31, 31)		
Grid cell size	(c_h, c_w)	(12, 12)		
Grid cell scale	κ	2		
Std dev of Gaussian for spatial attention	σ	0.1	0.15	0.1
Prior over $\overline{\mathcal{D}}_{(t)}^{h/w}$		$\mathcal{N}(\mu = -2.2, \sigma = 0.5)$		
Prior over $\overline{\mathcal{D}}_{(t)}^{y/x}$		$\mathcal{N}(\mu = 0, \sigma = 1)$		
Prior over $\overline{\mathcal{D}}_{(t)}^{\text{what}}$		$\mathcal{N}(\mu = 0, \sigma = 1)$		
Prior over $\overline{\mathcal{D}}_{(t)}^{\text{depth}}$		$\mathcal{N}(\mu = 0, \sigma = 1)$		
Parameter for prior over $\overline{\mathcal{D}}_{(t)}^{\text{pres}}$	ψ	See Section B.1.2		
Prior over $\overline{\mathcal{P}}_{(t)}^{h/w}$		$\mathcal{N}(\mu = 0, \sigma = 0.3)$		
Prior over $\overline{\mathcal{P}}_{(t)}^{y/x}$		$\mathcal{N}(\mu = 0, \sigma = 0.3)$		
Prior over $\overline{\mathcal{P}}_{(t)}^{\text{what}}$		$\mathcal{N}(\mu = 0, \sigma = 0.4)$		
Prior over $\overline{\mathcal{P}}_{(t)}^{\text{depth}}$		$\mathcal{N}(\mu = 0, \sigma = 1)$		
Schedule for ψ	$(v_{\text{start}}, v_{\text{end}}, v_{\text{step}})$	$(10^6, 0.0125, 10^3)$		
Appearance offset and scale	μ^β, σ^β	(0, 2)	(0, 1)	(0, 2)
Transparency offset and scale	μ^ξ, σ^ξ	(5, 0.1)	(3, 1)	(5, 0.1)
Rendering softmax temperature	λ	0.25		
Rendered object size	$(H_{\text{obj}}, W_{\text{obj}})$	(14, 14)		

Table C.2. Hyperparameter values for SILOT. The leftmost value is the default (i.e. for cells that are missing a value).

	Description	Architecture
d_ϕ^{bu}	Computes bottom-up features for Disc grid cells	See Table C.1
d_ϕ^{spatial}	Computes features of propped objects in Disc attention	FC([64, 64], ReLU)
d_ϕ^{fuse}	Combines top-down and bottom-up info in Disc	FC([100, 100], ReLU)
d_ϕ^{where}	Predicts params for posterior over $\bar{z}^{\text{where}}$	FC([100, 100], ReLU)
d_ϕ^{obj}	Processes glimpse in Disc	FC([256, 128], ReLU)
d_ϕ^{what}	Predicts params for posterior over $\bar{z}^{\text{what}}$	FC([100, 100], ReLU)
d_ϕ^{depth}	Predicts params for posterior over $\bar{z}^{\text{depth}}$	FC([100, 100], ReLU)
d_ϕ^{pres}	Predicts params for posterior over $\bar{z}^{\text{pres}}$	FC([100, 100], ReLU)
p_ϕ^{td}	Computes object features in Prop attention	FC([64, 64], ReLU)
p_ϕ^{spatial}	Computes object-pair features in Prop attention	FC([64, 64], ReLU)
p_ϕ^{glimpse}	Predicts params for initial Prop glimpse	FC([100, 100], ReLU)
p_ϕ^{bu}	Processes initial Prop glimpse	FC([256, 128], ReLU)
p_ϕ^{where}	Predicts params for posterior over $\tilde{z}^{\text{where}}$	FC([100, 100], ReLU)
p_ϕ^{obj}	Processes second glimpse in Prop	FC([256, 128], ReLU)
p_ϕ^{what}	Predicts params for posterior over $\tilde{z}^{\text{what}}$	FC([100, 100], ReLU)
p_ϕ^{depth}	Predicts params for posterior over $\tilde{z}^{\text{depth}}$	FC([100, 100], ReLU)
p_ϕ^{pres}	Predicts params for posterior over $\tilde{z}^{\text{pres}}$	FC([100, 100], ReLU)
p_ϕ^{rnn}	Updates deterministic hidden state	GRU(128)
r_θ^{obj}	Predicts object appearances in rendering.	FC([128, 256], ReLU)

Table C.3. Component neural networks in SILOT. FC([N_1, N_2], ReLU) is a sequence of 3 fully-connected layers (2 hidden layers with N_1 and N_2 units, respectively, and one output layer). The ReLU non-linearity is applied only at the hidden layers, the output nonlinearity is always the identity function.

For detecting objects, a graph is created from each frame in which the pixels are nodes, and two pixels are connected by an edge if and only if they are adjacent and have the same color. We then extract connected components from this graph, and call each connected component an object. This procedure yields a set of objects for each frame. In order to track objects over time (i.e. to assign persistent identifiers to the objects), we employ the Hungarian algorithm to find matches between detected objects in each pair of successive frames [101]. The Hungarian algorithm requires a matching score between any candidate objects, and for this we use the distance between object centroids; we also require that matching objects have the same color (objects pairs with mismatched colors are assigned a cost of ∞). The performance of ConnComp can be interpreted as a measure of the difficulty

of the dataset; it will be successful only to the extent that objects can be tracked by color alone.

C.2 Experiment Details

C.2.1 Scattered MNIST

Each video had spatial size 48×48 pixels. MNIST digits were resized to 14×14 pixels (their original size being 28×28). Initial digit velocities vectors were sampled uniformly, with a fixed magnitude of 2 pixels per frame. Digits passed through one another without event, and bounce off the edges of the frame. Initial digit positions were sampled randomly, one digit at a time. If the cumulative overlap between a newly sampled digit position and existing digit positions exceeded a threshold (98 pixels), the digit position was resampled until the threshold was not exceeded. This allowed us to control the amount of overlap between digits on the first frame. In subsequent frames, no overlap limit is enforced, and indeed because objects pass through one another and bounce off frame borders, the degree of overlap can be quite high in subsequent frames. In such cases, networks need to use information about the locations and trajectories of the overlapping objects at previous timesteps in order to track well.

The 60,000 MNIST digits were divided up into 80% training set, 10% validation set, and 10% test set. Videos for training set were created by uniformly sampling the number of digits to put in the video (from distribution Uniform$(1, 6)$ or Uniform$(1, 12)$ depending on the training condition), and then sampling that number of digits from the set of training digits. Similar approaches were taken for validation and testing; this ensures that at test time, networks are seeing digits that they have never seen before. We created 60,000 training videos and 1000 validation videos for each training condition, and 1000 videos for each testing condition (i.e. for each position on the x-axis of the plots in Figure 5 of the main text).

C.2.2 Scattered Shapes

Videos in the Scattered Shapes dataset were created in a manner similar to Scattered MNIST, with a few exceptions. Videos had spatial size 96×96; in some training conditions models were trained on full-size videos, and other conditions models were trained on random crops. Possible colors were red, green, blue, cyan, yellow, magenta, and possible shapes were circle, diamond, star, cross, x. All shape/color combinations were present in training, validation and test datasets. Initial pixel velocities had a fixed magnitude of 5 pixels per frame rather than 2. Additionally, when adding a shape to a video, it's size was randomized by choosing each spatial dimension from a Normal$(\mu = 14, \sigma = 1.4)$ distribution.

Appendix D

Experiment Details for 3DOM

D.1 Base Model Details

D.1.1 Scene Representation Networks

Both 3DOM and 2DOM make use of Scene Representation Networks [143] (SRNs) for modeling the static elements of 3D scenes. Training starts with an initial phase in which the SRN is by itself (note that this stage uses *the same data* as the later stages which train the dynamic-object parts of the models). Here, the SRN is trained until convergence; this ensures that when we later go to train the dynamic-object parts of the models (i.e. the components described in Section 6.5, the Discovery, Propagation, Selection and Rendering modules), the SRN is well-trained. This is important, because for each training scene, the differences between the training frames and the frames rendered by the SRN act as the primary training signal about what counts as an object (this is because, as discussed at the end of Section 6.4, the SRN is a *static* model of a 3D environment, and thus is not able to account for the dynamic/contingent elements of a scene). SRN-related hyperparameter values are shown in Table D.1.

The architectures of the neural networks involved in SRN are structured as follows. First, there is Φ_n, with architecture $FC(\text{n-inputs} = 3, \text{n-hidden} =$

$[128, 128, 128]$, n-outputs $= 64$). However, recall that the weights of Φ_n are predicted by a hypernetwork H. We assign a 128-dimensional embedding vector e_n to each value n; the entries of the embedding vector are trainable and initial values are chosen from a normal distribution with mean 0 and standard deviation 0.1. Then, since Φ_n has 4 sets of weight matrices, the hypernetwork H is made up of 4 different MLPs, each with architecture FC(n-inputs $= 128$, n-hidden $= [64, 64, 64]$), each of which maps from the embedding vector to a weight matrix. The output dimension of each network is equal to the number of entries in the weight matrix of Φ that it is predicting.

Note that the hypernetwork is probably not strictly necessary; in principle, it should be possible to simply instantiate a separate MLP Φ_n for each $n = \{1, \ldots, N\}$ (ending up with N different MLPs, each with its own set of weights; recall that N is the number of different static environments). However, as the number of environments gets larger, at a certain point using a completely separate MLP for each scene would involve more trainable parameters than a hypernetwork which maps from an embedding vector e_n to weights for Φ_n (in the hypernetwork solution, the weights of the hypernetwork are the only trainable parameters, and the size of the hypernetwork does not grow as N grows). Moreover the hypernetwork solution is the one suggested in the original SRN paper, and, despite some reasonable initial efforts, we were not able to get the "separate MLP" solution to work well (though in principle there is no reason that it should not), and thus settled on the hypernetwork solution.

There are two other trainable parts of the SRN framework. The first is an LSTM that predicts the raymarching step length during rendering. Following [143], we use an LSTM with 16 hidden units; it is beneficial to make the LSTM as small as possible, because it needs to be executed once *per raymarching step per pixel* of the rendering algorithm. The other trainable component is the pixel generator network Ψ, which maps from the feature vector obtained on the final raymarching step to a color. This network is an MLP with architecture FC(n-inputs $= 64$, n-hidden $= [256, 256, 256]$, n-outputs $= 3$). The pixel

Description	Obj Picking	Maze Nav
Learning rate	1×10^{-4}	
Batch size	4	8
Max gradient norm	10.0	
Optimizer	Adam	
Subsample Fraction	0.1	0.5

Table D.1. Hyperparameter values for training the Scene Representation Networks.The leftmost value is the default (i.e. for cells that are missing a value).

generator network is not required to be small, as it is only executed once per pixel (rather than once per raymarching step per pixel).

We render with 10 raymarching steps (i.e. $M = 10$). During the stage that we train the SRN by itself, we randomly sample 10% of the pixels in each image, and train only those. This allows us to include more videos in the batch (i.e. we are using more videos, but training a smaller fraction of the pixels in each video), and reduces the time required to perform an update step. However, in later stages where we train the whole network, we cannot do this subsampling as we need access to the entire image rendered by the SRN, as it is used as a background in 3DOM's Rendering module.

D.1.2 3DOM

The base set of hyperparameters for training 3DOM can be found in Table D.2, and the architectures of 3DOM's component neural networks can be found in Tables D.3, D.4, D.5, and D.6. As with SILOT, in 3DOM we use a form of curriculum learning in which we gradually increase the number of frames that the network is training on (when training with a number of frames that is less than the full episode length, we train with random subsequences of the full training episodes). The network starts by training on single frames. At this point, as there is only one frame, neither the Propagation nor Prior Propagation modules receive useful training signal. After 2×10^4 update steps, we start training on sequences of 2 frames; at this point the Discovery, Selection and Rendering modules should be marginally well-trained, which helps significantly when we increase the number of

training frames and the Propagation and Prior Propagation modules begin to be trained. We then increase the number of training frames by 2 for every 5×10^4 update steps, until the network is training on full videos/episodes (for the Simulated Object Picking task, the episodes are only 2 frames long, so the network is training on full videos after the first 2×10^4 steps). All models are trained for a total of 3×10^5 update steps.

D.1.3 2DOM

2DOM is an ablation of 3DOM, designed to be representative of past models that treat objects as existing on the camera plane rather than in 3D space. Given that 3DOM is basically an extension of SILOT into 3D, when we *remove* 3D capabilities from 3DOM we end up with a model that is very similar to SILOT. 2DOM is obtained from 3DOM in two steps: we remove its access to the ground-truth camera poses that are provided as part of the episode data, and we remove the depth training loss. As for hyperparameters, 2DOM largely uses the same hyperparameter values for 3DOM; most of these hyperparameters are tuned for good object segmentation, and should provide the same object-segmentation benefits for 2DOM as for 3DOM.

D.2 Experiment Details

D.2.1 Metrics

In 3DOM we use three metrics. These are Average Precision (AP), a measure of 2D object detection performance, assesses the quality of the bounding boxes predicted by the networks, but not their ability to track objects over time or to predict position in 3D space [44]. We also use two different forms of MOTA (Multi-Object Tracking Accuracy) [119]: 2D-MOTA, a measure of 2D object tracking performance, assesses how well the networks track the bounding boxes of objects (but not there positions in 3D space), and 3D-MOTA, which assesses networks ability to track the 3D positions of objects through 3D space.

Description	Variable	Obj Picking	Maze Nav
Learning rate		1×10^{-4}	
Batch size		4	
Max gradient norm		10.0	
Optimizer		Adam	
# steps for initial curriculum stage		2×10^4	
# steps for subsequent curriculum stages	N_{curric}	5×10^4	
Probability of discovery dropout	p_{dd}	0.9	0.5
Number of propagated/selected objects	K	80	10
Dimension of $\overline{\mathcal{D}}_{(t)}^{\text{what}}$ and $\overline{\mathcal{P}}_{(t)}^{\text{what}}$	N_{what}	64	
Base bbox size	(A_h, A_w)	(31, 31)	(43, 43)
d_ϕ^{backbone} receptive field size		(31, 31)	(43, 43)
Grid cell size	(c_h, c_w)	(8, 8)	
Grid cell scale	κ	1	
Glimpse shape		(21, 21)	
Prop. localization glimpse shape		(32, 32)	
Prop. localization glimpse scale	χ	2	
Min. Max. Depth		(0.1, 20.)	
Depth Loss Weight	λ^{depth}	100.	
Prior over $\overline{\mathcal{D}}_{(t)}^{h/w}$ and $\overline{\mathcal{P}}_{(t)}^{h/w}$		$\mathcal{N}(\mu = -2., \sigma = 1.)$	
Prior over $\overline{\mathcal{D}}_{(t)}^{\text{depth}}$ and $\overline{\mathcal{P}}_{(t)}^{\text{depth}}$		$\mathcal{N}(\mu = 0, \sigma = 1.5)$	
Parameter for prior over $\overline{\mathcal{D}}_{(t)}^{\text{pres}}$ and $\overline{\mathcal{P}}_{(t)}^{\text{pres}}$	ψ	1.	
Prior over $\overline{\mathcal{D}}_{(t)}^{y/x}$		$\mathcal{N}(\mu = 0, \sigma = 1)$	
Prior over $\overline{\mathcal{D}}_{(t)}^{\text{what}}$		$\mathcal{N}(\mu = 0, \sigma = 1)$	
Prior over $\overline{\mathcal{D}}_{(t)}^{\text{depth}}$		$\mathcal{N}(\mu = 0, \sigma = 1)$	
Prior over $\overline{\mathcal{P}}_{(t)}^{y/x}$		$\mathcal{N}(\mu = 0, \sigma = 1)$	
Prior over $\overline{\mathcal{P}}_{(t)}^{\text{what}}$		$\mathcal{N}(\mu = 0, \sigma = 0.3)$	
Prior over $\overline{\mathcal{P}}_{(t)}^{\text{depth}}$		$\mathcal{N}(\mu = 0, \sigma = 1)$	
Appearance offset and scale	μ^β, σ^β	(0.0, 1.0)	
Transparency offset and scale	μ^ξ, σ^ξ	(1.0, 0.1)	
Rendered object size	$(H_{\text{obj}}, W_{\text{obj}})$	(21, 21)	
Output standard deviation	σ^{image}	0.25	

Table D.2. Hyperparameter values for 3DOM. The leftmost value is the default (i.e. for cells that are missing a value).

	Description	Architecture
Shared between Discovery and Propagation		
d_ϕ^{obj}	Processes glimpses	FC([512, 256], ReLU)
d_ϕ^{depth}	Predicts params for posterior over $\overline{\mathcal{D}}_{(t)}^{\text{depth}}$ and $\overline{\mathcal{P}}_{(t)}^{\text{depth}}$	FC([128, 128], ReLU)
RNN_ϕ	Updates deterministic hidden state of objects	GRU(128)
Discovery		
d_ϕ^{backbone}	Computes features for Discovery units	See Tables D.5 and D.6
$d_\phi^{\text{bbox-2D}}$	Predicts params for posterior over $\overline{\mathcal{D}}_{(t)}^{\text{where}}$	FC([64], ReLU)
d_ϕ^{what}	Predicts params for posterior over $\overline{\mathcal{D}}_{(t)}^{\text{what}}$	FC([64], ReLU)
d_ϕ^{pres}	Predicts params for posterior over $\overline{\mathcal{D}}_{(t)}^{\text{pres}}$	FC([64], ReLU)
Propagation		
p_ϕ^{loc}	Processes Prop localization glimpse	See Table D.4
$p_\phi^{\text{bbox-2D}}$	Predicts params for posterior over $\overline{\mathcal{P}}_{(t)}^{\text{where}}$	FC([128, 128], ReLU)
p_ϕ^{what}	Predicts params for posterior over $\overline{\mathcal{P}}_{(t)}^{\text{what}}$	FC([128, 128], ReLU)
p_ϕ^{visible}	Predicts params for posterior over $\overline{\mathcal{P}}_{(t)}^{\text{pres}}$	FC([128, 128], ReLU)
Selection		
$s_\phi^{\text{disc-disc}}$	Computes suppression between two Discovered objects.	FC([128, 128], ReLU)
$s_\phi^{\text{prop-disc}}$	Computes suppression of a Disc object by a Prop object.	FC([128, 128], ReLU)
Rendering		
r_θ^{obj}	Predicts object appearances in rendering.	FC([256, 512], ReLU)

Table D.3. Component neural networks in 3DOM. FC([$N_1, \ldots, N_L$], ReLU) is a sequence of $L+1$ fully-connected layers (L hidden layers, the ℓ-th having N_ℓ-many units, and one output layer). The ReLU non-linearity is applied only at the hidden layers, the output nonlinearity is always the identity function.

Type	# Filters	Filter Size	Stride	Nonlinearity
Conv	32	4	1	ReLU
Conv	64	4	2	ReLU
Conv	64	4	1	ReLU
Conv	128	4	2	ReLU
Conv	128	4	1	ReLU
FC	128	/	/	ReLU
FC	64	/	/	None

Table D.4. Architecture for the convolutional network in 3DOM that processes the localization glimpse. Makes use of FiLM [126] and CoordConv [110]. No pooling is used at any point.

Type	# Filters	Filter Size	Stride	Nonlinearity
Conv	16	3	2	ReLU
Conv	32	3	2	ReLU
Conv	64	3	2	ReLU
Conv	64	3	1	ReLU
Conv	128	1	1	ReLU
Conv	128	1	1	ReLU
Conv	128	1	1	None

Table D.5. Architecture for 3DOM's Discovery backbone network d_ϕ^{backbone} for the Simulated Object Picking environment. Batch norm is used at every layer, and no pooling is used at any point.

Type	# Filters	Filter Size	Stride	Nonlinearity
Conv	16	3	2	ReLU
Conv	16	3	1	ReLU
Conv	32	3	2	ReLU
Conv	32	3	1	ReLU
Conv	64	3	2	ReLU
Conv	64	3	1	ReLU
Conv	128	1	1	ReLU
Conv	128	1	1	ReLU
Conv	128	1	1	None

Table D.6. Architecture for 3DOM's Discovery backbone network d_ϕ^{backbone} for the First-Person Maze Navigation environment. Batch norm is used at every layer, and no pooling is used at any point.

Each of these metrics relies on thresholds determining when we consider some object predicted by the model to be doing a good job of tracking some ground-truth object. It is common to pick a *range* of thresholds so that the measure takes into account performance across a range of different accuracy requirements.

In both AP and 2D-MOTA, the distance between a predicted bounding box and a ground-truth bounding box is measured by 1-IOU (Intersection over Union); recall that IOU the area of the intersection of the bounding boxes divided by the area of their union (see Section 2.4.2). The IOU distance thresholds we use are: [.55, .65, .75, .85, .95].

For 3D-MOTA, the distance between a predicted 3D object position and a ground-truth 3D object position is measured by the Euclidean distance. The distance thresholds we

used are [0.5, 1., 1.5 2. 2.5] for Simulated Object Picking and [1, 2, 3, 4, 5] for First-Person Maze Navigation (both measured in "metres"; see Sections D.2.2 and D.2.3 below for the dimensions of the environments in this unit of distance).

Finally, both of the MOTA metrics require applying a threshold to the *pres* attribute values; that is, the network needs to make a "hard" decision about whether each object exists (this lies in contrast to AP, which can make proper use of soft *pres* values). For all models we use the pres threshold which maximizes performance on the validation set after the network has been trained, chosen from a small set of possible values between 0 and 1.

D.2.2 Simulated Object Picking

Static Environment. Each static environment is a "bin" 8 metres on each side, with walls 1.5 metres high. We used 3 different bins, each with a different texture (marble, wood and cardboard).

Objects. There are 8 different objects: pine tree, office chair, traffic cone, keycard, barrel, potion, rubber duck and medkit. The traffic cone, keycard, rubber duck, potion and medkit are all multicolored, and thus cannot be segmented based on color alone. Each object class has a different height, the average of which is .94 metres. When populating the bins with objects, object positions were randomly sampled on the floor of the bin, and rejection sampling was used to ensure that no two objects occupied the same region of space.

Agent/Camera. The camera lives on a sphere of radius 6 metres that is concentric with the bin, and is required to always look at the bin center. To obtain the initial camera position, we sampled a horizontal (2D) displacement away from the bin center uniformly at random, and then sampled the pitch of the camera position above the horizontal plane uniformly at random from the range 35° to 55°. The camera was then projected onto the 6 metre sphere, and then a facing direction was computed that had the camera look at the bin center. The camera moves between the first and second frame, the camera takes one of four actions: move up, move down, move left, move right. Vertical movement actions

modify the pitch of the camera position by 5°/10° during training/evaluation, respectively. Horizontal movement actions modify the rotation of the camera position about the bin center by 18°/36° during training/evaluation, respectively. After the action, the camera look-at direction was adjusted so that the camera was once again looking at the bin center.
Datasets. Each frame of the videos is 96 pixels x 96 pixels, and each video is 2 frames long. Each training/validation/test set contains 60000/100/250 videos.

D.2.3 First Person Maze Navigation

Static Environments. For the static environments, we used 20 randomly generated mazes. Maze layouts were randomly generated on a 4x4 grid of rooms (each room 4 metres x 4 metres, and walls are 2.74 metres tall) using a simple algorithm based on randomized depth-first search. Textures for the floor, walls and ceilings were randomly sampled from a pool of 7 textures provided with Miniworld (marble, wood, cardboard, brick wall, stucco, rock and slime). Note that maze layout and textures are generated just once, and the sampled values for a particular maze are used for all episodes that occur in that maze (in particular, it is NOT the case that we re-sample layout or textures on each episode; it would not be possible to train the SRN if that were the case, as training an SRN for a scene relies on having many different views of the scene).
Objects. Each episode, one of the 20 mazes is randomly selected and populated with objects. The number of objects is given by $\lfloor 1.5 \cdot N_{\text{rooms}} \rfloor$, where N_{rooms} is the number of active rooms in the selected maze. The 3D position of each object is randomly sampled from the valid volume of the maze, with rejection sampling used to ensure that no two objects occupy the same space. Objects are allowed to float, so the initial *height* of the object is sampled as part of this process. At the beginning of its life, each object samples a Bernoulli random variable p_{move} indicating whether it is a moving object; p_{move} varies between datasets, but always takes on a value of 0, 0.5 or 1. Objects that move randomly sample a movement direction, and proceed in that direction at a speed of 0.25 metres per timestep for their life, bouncing rigidly whenever they encounter a wall, floor or ceiling.

We use 4 different shapes (sphere, cube, cone, torus) and 3 different color pairs (red-yellow, green-cyan, blue-magenta), combining for a total of 3 x 4 = 12 object kinds. Each object kind is randomly assigned a size. Each object is randomly assigned one of the 12 kinds at birth. Each object changes color over time, cycling back and forth between the two colors in its color pair, completing a cycle once every 12 frames.

Agent/Camera. To pick the initial camera pose, we select a random object in the scene, and re-sample the camera pose until the selected object is in view from the perspective of the camera and is at least 2 metres away. During camera pose sampling, the horizontal position is sampled uniformly at random from the valid volume for the given maze, the height is set to 1.5, the horizontal rotation (yaw) is chosen uniformly at random, the vertical rotation of the camera with respect to the ground plane (pitch) is set to 0 (so that the camera faces in a direction perpendicular to the vertical direction) and the roll is set to 0. This re-sampling helps prevent an abundance of episodes where the agent is simply staring at a wall, providing no useful information about objects.

Motion of the agent/camera throughout an episode is determined by actions taken each timestep. The actions available to the agent are: turn left, turn right, move forward, move backward, move left, move right. The two turning actions rotate the agent 15° in the appropriate direction, while the four movement actions move the agent 0.15 metres in the appropriate direction. Actions are chosen uniformly at random each timestep.

Datasets. Each frame is 96 pixels x 96 pixels, and each video is 8 frames long. Each training/validation/test set contains 20000/100/250 videos.

Generalizing to Different Static Environments

In Section 6.7.3 we describe an experiment in which we test whether the dynamic-object portion of a 3DOM network (i.e. everything except the SRN) can be used successfully in novel static environments, outside the fixed collection of 20 mazes in which the training data were generated. When operating in these new test environments, we will not have a trained SRN available (in each set of new static environments, we use a test set of size 250

episodes; that *might* be enough data to train a new set of SRN, but we could equally-well imagine a test dataset with just 1 episode, which definitely would not be enough data to train the SRN). Fortunately, as discussed in that section, the dynamic-object part of 3DOM does not *need* a trained SRN at evaluation time (a trained SRN is mainly required at training time, for helping the network determine what is and is not an object), and we can therefore use it to process scenes from a novel set of static environments.

To evaluate this, we generated new sets of mazes of varying size. We used essentially the same procedure for maze generation as described above, generating 20 mazes per testing condition, but varied the size of the grid of rooms on which the mazes were generated (3x3, 4x4, 5x5 and 6x6; the 4x4 case is the same size of maze as the training data, but different random seeds were used here so that the generated set of mazes had different structure and textures than the training mazes). The larger mazes have longer hallways, and thus objects tend to be further away on average. Also, when populating a maze with objects for an episode, the number of objects we create is 1.5 $\times$ (the number of valid rooms in the selected maze); however, the generated objects are scattered uniformly throughout the maze, and there is nothing to ensure that each room actually receives an equal number of objects. In the larger mazes, this can create situations where there are large concentrations of objects, making detection and tracking quite difficult.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Andréa Aguiar and Renée Baillargeon. 2.5-month-old infants' reasoning about when objects should and should not be occluded. *Cognitive psychology*, 39(2):116–157, 1999.

[3] Andréa Aguiar and Renée Baillargeon. Developments in young infants' reasoning about occluded objects. *Cognitive psychology*, 45(2):267–336, 2002.

[4] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-time rendering*. Crc Press, 2019.

[5] John Aloimonos, Isaac Weiss, and Amit Bandyopadhyay. Active vision. *International journal of computer vision*, 1(4):333–356, 1988.

[6] Renée Baillargeon. The acquisition of physical knowledge in infancy: A summary in eight lessons. *Blackwell handbook of childhood cognitive development*, 1(46-83):1, 2002.

[7] Renee Baillargeon, Elizabeth S Spelke, and Stanley Wasserman. Object permanence in five-month-old infants. *Cognition*, 20(3):191–208, 1985.

[8] Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

[9] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[10] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[11] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[12] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

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

[14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[15] Lars Buesing, Theophane Weber, Sébastien Racaniere, SM Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

[16] Christopher P Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

[17] Arunkumar Byravan, Jost Tobias Springenberg, Abbas Abdolmaleki, Roland Hafner, Michael Neunert, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Imagined value gradients: Model-based policy optimization with tranferable latent dynamics models. In *Conference on Robot Learning*, pages 566–589. PMLR, 2020.

[18] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[19] Susan Carey. *The origin of concepts*. Oxford University Press, 2009.

[20] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[21] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1721–1730, 2015.

[22] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[23] Kamalika Chaudhuri, Brighten Godfrey, David Ratajczak, and Hoeteck Wee. On the complexity of the game of set, 2003.

[24] Chang Chen, Fei Deng, and Sungjin Ahn. Learning to infer 3d object models from images. *arXiv preprint arXiv:2006.06130*, 2020.

[25] Maxime Chevalier-Boisvert. gym-miniworld environment for openai gym, 2018.

[26] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *Advances in neural information processing systems*, 28:2980–2988, 2015.

[27] Gioele Ciaparrone, Francisco Luque Sánchez, Siham Tabik, Luigi Troiano, Roberto Tagliaferri, and Francisco Herrera. Deep learning in video multi-object tracking: A survey. *Neurocomputing*, 381:61–88, 2020.

[28] Eric Crawford and Joelle Pineau. Spatially invariant, unsupervised object detection with convolutional neural networks. In *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[29] Eric Crawford and Joelle Pineau. Exploiting spatial invariance for scalable unsupervised object tracking. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[30] Eric Crawford and Joelle Pineau. Learning 3d object-oriented world models from unlabeled videos. In *ICML 2020 Workshop on Object-Oriented Learning: Perception, Representation and Learning*, 2020.

[31] The Hien Dang Ha. A guide to receptive field arithmetic for convolutional neural networks. https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807, 2017. Accessed: Aug 7, 2019.

[32] Sven J Dickinson, Aleš Leonardis, Bernt Schiele, and Michael J Tarr. *Object categorization: computer and human vision perspectives*. Cambridge University Press, 2009.

[33] Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 240–247. ACM, 2008.

[34] Andreas Doerr, Christian Daniel, Martin Schiegg, Duy Nguyen-Tuong, Stefan Schaal, Marc Toussaint, and Sebastian Trimpe. Probabilistic recurrent state-space models. *arXiv preprint arXiv:1801.10395*, 2018.

[35] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[36] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3, 2009.

[37] Yilun Du, Kevin Smith, Tomer Ulman, Joshua Tenenbaum, and Jiajun Wu. Unsupervised discovery of 3d physical objects from video. *arXiv preprint arXiv:2007.12348*, 2020.

[38] Mary T Dzindolet, Scott A Peterson, Regina A Pomranky, Linda G Pierce, and Hall P Beck. The role of trust in automation reliance. *International journal of human-computer studies*, 58(6):697–718, 2003.

[39] Cathrin Elich, Martin R Oswald, Marc Pollefeys, and Jörg Stückler. Semi-supervised learning of multi-object 3d scene representations. *arXiv preprint arXiv:2010.04030*, 2020.

[40] Martin Engelcke, Adam R Kosiorek, Oiwi Parker Jones, and Ingmar Posner. Genesis: Generative scene inference and sampling with object-centric latent representations. *arXiv preprint arXiv:1907.13052*, 2019.

[41] Clemens Eppner, Sebastian Höfer, Rico Jonschkowski, Roberto Martín-Martín, Arne Sieverling, Vincent Wall, and Oliver Brock. Lessons from the amazon picking challenge: Four aspects of building robotic systems. In *Robotics: science and systems*, 2016.

[42] Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pages 3225–3233, 2016.

[43] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[44] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[45] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2009.

[46] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[47] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. Deepview: View synthesis with learned gradient descent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2367–2376, 2019.

[48] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. *Advances in neural information processing systems*, 29:2199–2207, 2016.

[49] Keisuke Fukuda, Edward Awh, and Edward K Vogel. Discrete capacity limits in visual working memory. *Current opinion in neurobiology*, 20(2):177–182, 2010.

[50] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

[51] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[52] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[53] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[54] Vikash Goel, Jameson Weng, and Pascal Poupart. Unsupervised video object segmentation for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5683–5694, 2018.

[55] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[56] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[57] Edwin James Green. A theory of perceptual objects. *Philosophy and Phenomenological Research*, 99(3):663–693, 2019.

[58] Edwin James Green and Jake Quilty-Dunn. What is an object file? *The British Journal for the Philosophy of Science*, 2017.

[59] Klaus Greff, Raphaël Lopez Kaufmann, Rishab Kabra, Nick Watters, Chris Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-

object representation learning with iterative variational inference. *arXiv preprint arXiv:1903.00450*, 2019.

[60] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization. In *Advances in Neural Information Processing Systems*, pages 6691–6701, 2017.

[61] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. On the binding problem in artificial neural networks. *arXiv preprint arXiv:2012.05208*, 2020.

[62] Steven M Gzesh and Colleen F Surber. Visual perspective-taking skills in children. *Child development*, pages 1204–1213, 1985.

[63] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[64] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[65] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[66] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[67] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[68] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[69] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[70] Zhen He, Jian Li, Daxue Liu, Hangen He, and David Barber. Tracking by animation: Unsupervised learning of multi-object attentive trackers. *arXiv preprint arXiv:1809.03137*, 2018.

[71] Paul Henderson and Vittorio Ferrari. Learning single-image 3d reconstruction by generative modelling of shape, pose and shading. *International Journal of Computer Vision*, pages 1–20, 2019.

[72] Paul Henderson and Christoph H Lampert. Unsupervised object-centric video generation and decomposition in 3d. *arXiv preprint arXiv:2007.06705*, 2020.

[73] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250, 2000.

[74] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

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

[76] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4507–4515, 2017.

[77] Ian P Howard, Brian J Rogers, et al. *Binocular vision and stereopsis*. Oxford University Press, USA, 1995.

[78] 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 *Advances in Neural Information Processing Systems*, pages 517–526, 2018.

[79] Jonathan Huang and Kevin Murphy. Efficient inference in occlusion-aware generative models of images. *arXiv preprint arXiv:1511.06362*, 2015.

[80] John F Hughes, Andries Van Dam, James D Foley, Morgan McGuire, Steven K Feiner, David F Sklar, and Kurt Akeley. *Computer graphics: principles and practice*. Pearson education, 2014.

[81] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*, pages 754–762. PMLR, 2014.

[82] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[83] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[84] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[85] Jindong Jiang, Sepehr Janghorbani, Gerard De Melo, and Sungjin Ahn. Scalor: Generative world models with scalable object representations. In *International Conference on Learning Representations*, 2019.

[86] 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.

[87] Scott P Johnson and Richard N Aslin. Perception of object unity in 2-month-old infants. *Developmental Psychology*, 31(5):739, 1995.

[88] Scott P Johnson and Richard N Aslin. Perception of object unity in young infants: The roles of motion, depth, and orientation. *Cognitive Development*, 11(2):161–180, 1996.

[89] Daniel Kahneman, Anne Treisman, and Brian J Gibbs. The reviewing of object files: Object-specific integration of information. *Cognitive psychology*, 24(2):175–219, 1992.

[90] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[91] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[92] Ken Kansky, Tom Silver, David A Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *International Conference on Machine Learning*, pages 1809–1818, 2017.

[93] Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick Van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

[94] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[96] Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*, 2019.

[97] Jan J Koenderink. *Solid shape*. MIT press, 1990.

[98] Adam Kosiorek, Hyunjik Kim, Yee Whye Teh, and Ingmar Posner. Sequential attend, infer, repeat: Generative modelling of moving objects. In *Advances in Neural Information Processing Systems*, pages 8606–8616, 2018.

[99] Jannik Kossen, Karl Stelzner, Marcel Hussing, Claas Voelcker, and Kristian Kersting. Structured object-aware physics prediction for video modeling and planning. *arXiv preprint arXiv:1910.02425*, 2019.

[100] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[101] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[102] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *International conference on machine learning*, pages 1558–1566. PMLR, 2016.

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

[104] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. From big to small: Multi-scale local planar guidance for monocular depth estimation. *arXiv preprint arXiv:1907.10326*, 2019.

[105] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Stanford cs231n lecture notes: Convolutional neural networks. https://cs231n.github.io/convolutional-networks. Accessed: Jun 20, 2020.

[106] Xiujun Li, Xi Yin, Chunyuan Li, Xiaowei Hu, Pengchuan Zhang, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, et al. Oscar: Object-semantics aligned pre-training for vision-language tasks. *arXiv preprint arXiv:2004.06165*, 2020.

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

[108] Zhixuan Lin, Yi-Fu Wu, Skand Peri, Bofeng Fu, Jindong Jiang, and Sungjin Ahn. Improving generative imagination in object-centric world models. *arXiv preprint arXiv:2010.02054*, 2020.

[109] Zhixuan Lin, Yi-Fu Wu, Skand Vishwanath Peri, Weihao Sun, Gautam Singh, Fei Deng, Jindong Jiang, and Sungjin Ahn. Space: Unsupervised object-oriented scene representation via spatial attention and decomposition. *arXiv preprint arXiv:2001.02407*, 2020.

[110] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *Advances in Neural Information Processing Systems*, pages 9605–9616, 2018.

[111] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[112] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[113] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. *arXiv preprint arXiv:1701.04128*, 2017.

[114] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[115] Gary F Marcus. *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. MIT Press, 2003.

[116] Joseph Marino, Milan Cvitkovic, and Yisong Yue. A general method for amortizing variational filtering. In *Advances in Neural Information Processing Systems*, pages 7857–7868, 2018.

[117] Joseph Marino, Yisong Yue, and Stephan Mandt. Iterative amortized inference. *arXiv preprint arXiv:1807.09356*, 2018.

[118] David Marr. *Vision: A computational investigation into the human representation and processing of visual information*. WH Freeman, 1982.

[119] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. Mot16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*, 2016.

[120] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[121] Thu Nguyen-Phuoc, Christian Richardt, Long Mai, Yong-Liang Yang, and Niloy Mitra. Blockgan: Learning 3d object-aware scene representations from unlabelled images. *arXiv preprint arXiv:2002.08988*, 2020.

[122] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. *arXiv preprint arXiv:2011.12100*, 2020.

[123] Stephen E Palmer. *Vision science: Photons to phenomenology*. MIT press, 1999.

[124] 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.

[125] Deepak Pathak, Yide Shentu, Dian Chen, Pulkit Agrawal, Trevor Darrell, Sergey Levine, and Jitendra Malik. Learning instance segmentation by interaction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2042–2045, 2018.

[126] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. *arXiv preprint arXiv:1709.07871*, 2017.

[127] Zenon W Pylyshyn. Visual indexes, preconceptual objects, and situated vision. *Cognition*, 80(1-2):127–158, 2001.

[128] Zenon W Pylyshyn and Ron W Storm. Tracking multiple independent targets: Evidence for a parallel tracking mechanism. *Spatial vision*, 3(3):179–197, 1988.

[129] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[130] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[131] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint*, 2017.

[132] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[133] Lucia Regolin and Giorgio Vallortigara. Perception of partly occluded objects by young chicks. *Perception & psychophysics*, 57(7):971–976, 1995.

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

[135] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[136] Philippe Rochat. Object manipulation and exploration in 2-to 5-month-old infants. *Developmental Psychology*, 25(6):871, 1989.

[137] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[138] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[139] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016.

[140] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3528–3536, 2015.

[141] Roger N Shepard and Jacqueline Metzler. Mental rotation of three-dimensional objects. *Science*, 171(3972):701–703, 1971.

[142] Herbert A Simon. *The sciences of the artificial*. MIT Press, 2019.

[143] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, pages 1119–1130, 2019.

[144] Matthew H Slater and Andrea Borghini. Carving nature at its joints: Natural kinds in metaphysics and science. chapter Introduction: Lessons from the scientific butchery. MIT Press, 2011.

[145] Elizabeth S Spelke. Principles of object perception. *Cognitive science*, 14(1):29–56, 1990.

[146] Elizabeth S Spelke and Katherine D Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

[147] Karl Stelzner, Robert Peharz, and Kristian Kersting. Faster attend-infer-repeat with tractable probabilistic models. In *International Conference on Machine Learning*, pages 5966–5975, 2019.

[148] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[149] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

[150] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[151] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10781–10790, 2020.

[152] Andrew Tao, Karan Sapra, and Bryan Catanzaro. Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821*, 2020.

[153] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, et al. State of the art on neural rendering. *arXiv preprint arXiv:2004.03805*, 2020.

[154] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[155] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy. In *Advances in Neural Information Processing Systems*, pages 8252–8262, 2019.

[156] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[157] Eloisa Valenza, Irene Leo, Lucia Gava, and Francesca Simion. Perceptual completion in newborn human infants. *Child Development*, 77(6):1810–1821, 2006.

[158] Sjoerd van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*, 2018.

[159] 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.

[160] Rishi Veerapaneni, John D Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua B Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. *arXiv preprint arXiv:1910.12827*, 2019.

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

[162] Duo Wang, Mateja Jamnik, and Pietro Lio. Unsupervised and interpretable scene discovery with discrete-attend-infer-repeat. *arXiv preprint arXiv:1903.06581*, 2019.

[163] Nicholas Watters, Loic Matthey, Matko Bosnjak, Christopher P Burgess, and Alexander Lerchner. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*, 2019.

[164] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[165] Taufik Xu, LI Chongxuan, Jun Zhu, and Bo Zhang. Multi-objects generation with amortized structural regularization. In *Advances in Neural Information Processing Systems*, pages 6619–6629, 2019.

[166] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. *arXiv preprint arXiv:1612.00814*, 2016.

[167] Quan-shi Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018.

[168] Shengyu Zhao, Yilun Sheng, Yue Dong, Eric I Chang, Yan Xu, et al. Maskflownet: Asymmetric feature matching with learnable occlusion mask. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6278–6287, 2020.

[169] Guangxiang Zhu, Jianhao Wang, Zhizhou Ren, Zichuan Lin, and Chongjie Zhang. Object-oriented dynamics learning through multi-level abstraction. *arXiv preprint arXiv:1904.07482*, 2019.