Elements of Unsupervised Scene Understanding: Objectives, Structures, and Modalities

Zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.)
vorgelegte Dissertation von Karl Stelzner aus Dortmund
Tag der Einreichung: 24. August 2023

1. Gutachten: Prof. Dr. Kristian Kersting
2. Gutachten: Adam R. Kosiorek, PhD
3. Gutachten: Prof. Dr. Antonio Vergari
Darmstadt, Technische Universität Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science
Department
Artificial Intelligence and
Machine Learning Lab

Elements of Unsupervised Scene Understanding: Objectives, Structures, and Modalities

Doctoral thesis by Karl Stelzner

1. Review: Prof. Dr. Kristian Kersting
2. Review: Adam R. Kosiorek, PhD
3. Review: Prof. Dr. Antonio Vergari

Date of submission: August 24, 2023

Darmstadt, Technische Universität Darmstadt

Erklärungen laut Promotionsordnung

§8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 24. August 2023

Karl Stelzner

Acknowledgements

First and foremost, I would like to thank my PhD advisor Kristian Kersting. Throughout my work on this thesis, Kristian has supported me with countless pieces of advice, research ideas, and opportunities to connect with other researchers. Above all, he has consistently prioritized the ability of myself and my fellow students to freely pursue their research interests, and worked to minimize outside distractions.

Second, I owe many thanks to Adam Kosiorek. Over the past 4 years, he has consistently supported me with thoughtful advice, interesting research discussions, and in-depth collaborations. Had Adam been my official PhD advisor, I could not have asked any more of him – making it even more remarkable that he assumed this role for me, a junior PhD student with whom he had no formal affiliation.

Third, I want to thank my co-authors and collaborators, Robert Peharz, Antonio Vergari, Alejandro Molina, Xiaoting Shao, Martin Trapp, Fabrizio Ventola, Steven Braun, and others. Each of them has been a pleasure to work with, and treated each project they were involved in as seriously as if it was their own. I would also like to thank my students, who have greatly helped me in my research. In particular, I am grateful to Jannik Kossen, Claas Voelcker, and Marcel Hussing for their fruitful collaboration on the STOVE project, and Idris Nematpur for his work on ObSuRF. I would also like to thank my other colleagues with whom I shared much of the journey of growing a brand-new lab to one of over 30 PhD students, including Patrick Schramowski, Zhongjie Yu, Wolfgang Stammer, Arseny Skryagin, Johannes Czech, Quentin Delfosse, and Ira Tesar. Each of you made the journey much more enjoyable.

Finally, I would like to thank my family and friends, who have stood with me throughout the ups and downs of this work. Most importantly, I could not have done it without the dedication and emotional support from my girlfriend Carly.

I would like to dedicate this thesis to the memory of my friend Martin Rettelbach, who first got me into computer science.

Abstract

Enabling robust interactions between automated systems and the real world is a major goal of artificial intelligence. A key ingredient towards this goal is *scene understanding*: the ability to process visual imagery into a concise representation of the depicted scene, including the identity, position, and geometry of objects. While supervised deep learning approaches have proven effective at processing visual inputs, the cost of supplying human annotations for training quickly becomes infeasible as the diversity of the inputs and the required level of detail increases, putting full real-world scene understanding out of reach.

For this reason, this thesis investigates unsupervised methods to scene understanding. In particular, we utilize generative models with structured latent variables to facilitate the learning of object-based representations. We start our investigation in an autoencoding setting, where we highlight the capability of such systems to identify objects without human supervision, as well as the advantages of integrating tractable components within them. At the same time, we identify some limitations of this setting, which prevents success in more visually complex environments. Based on this, we then turn to video data, where we leverage the prediction of dynamics to both regularize the representation learning task and to enable applications to reinforcement learning. Finally, to take another step towards a real world setting, we investigate the learning of representations encoding 3D geometry. We discuss various methods to encode and learn about 3D scene structure, and present a model which simultaneously infers the geometry of a given scene, and segments it into objects. We conclude by discussing future challenges and lessons learned. In particular, we touch on the challenge of modelling uncertainty when inferring 3D geometry, the tradeoffs between various data sources, and the cost of including model structure.

Contents

List of Figures

List of Tables

1. Introduction

Building machines that understand the world around them is a longstanding objective of artificial intelligence. The emergent capability of neural networks to infer the content of images based on human supervision kickstarted the deep learning revolution (Krizhevsky et al., 2012; LeCun et al., 2015) a decade ago. While the success of machine learning continues, the limits of human supervision quickly became apparant: Manually labelling data to cover the diversity and and detail of the natural world is simply too laborious and expensive. This shifted the focus to unsupervised representation learning approaches, which learn about the world using data collected at scale, without human labor. Recently, dramatic successes have been achieved in large scale unsupervised learning on text (Brown et al., 2020) and image-caption pairs (Ramesh et al., 2022; Rombach et al., 2021a) gathered from the open Internet, enabling unprecedented general reasoning (Bubeck et al., 2023). While these systems yield impressive results on text and image synthesis, they are not capable of *inferring* the contents of a given image to the accuracy necessary for robust interaction with the natural world. As a result, domains such as robot grasping or autonomous driving are still dominated today by small scale, supervised approaches (Kleeberger et al., 2020; Yurtsever et al., 2020).

In this thesis, we discuss unsupervised methods to visual scene understanding. These differ from other methods in two important aspects: First, they are trained using data not collected via human labor, and instead obtained from the world using sensors, such as cameras. For the purpose of this thesis, we also consider data incidentally produced by human activity as unsupervised data, such as images or text gathered on the open Internet. Second, they should yield a representation detailed enough to facilitate interaction with the world. This means it should be powerful enough to allow for the identification of objects, their locations, and their physical properties. Before we further characterize these approaches, we first discuss the limits of supervised learning to highlight their necessity.

1.1. Limits of Supervised Learning

For many decades, the capabilities of machine learning for prediction tasks have been limited by the expressiveness of available models. This started to change with the appearance of deep learning in the early 2010s. Since then, the original recipe of overparameterized functions optimized via stochastic gradient descent has been continuously refined with new ingredients, such as residual connections, attention, and improved hardware support. This has led to an exponential increase in model size, as measured by both parameter count and compute budget (Sevilla et al., 2022).

At the same time, the size of supervised vision datasets has stayed roughly constant: Despite their age, Imagenet (10 Million images with class labels) and COCO (200k images with object segmentations) remain some of the largest and most popular image datasets featuring human annotations (Deng et al., 2009; Lin et al., 2014). Unsupervised datasets on the other hand have grown consistently, and now feature billions of images (Schuhmann et al., 2022), highlighting the widening gap between unsupervised and supervised data.

This lack of dataset scaling makes it difficult to see how the generalization ability observed in large scale unsupervised models could be replicated for supervised methods. Many real-world tasks of interest feature power law distributions, with a long tail of diverse phenomena that will only appear in very large datasets. Additionally, the conception of standard supervised learning tasks often sets limits to generalization from the outset: Object classification or image segmentation tasks presuppose a finite set of known discrete phenomena which need to be understood. The same is true for object pose prediction, as this task is only well defined when canonical poses for the considered objects are defined. This is in contrast to the real-world requirements of tasks such as autonomous driving, where we may encounter a virtually unlimited variety of phenomena. Similarly, we would like machines to match the ability of humans to understand an observed object's geometry and how to grasp it, even if it does not belong to any known category.

These issues compound when we consider the degree of detail that needs to be captured to unlock these capabilities. While annotating an image with a class label is relatively fast and inexpensive, it is also a very coarse representation of the content of an image. Many real-world tasks require much richer representations: Robot grasping depends on an accurate understanding of object geometry, surface material, and weight. Human level autonomous driving requires understanding not just the position and velocity of

other vehicles, but also needs to anticipate the actions of bicyclists and pedestrians, taking into account their body language.

Annotating such information manually is prohibitively expensive for all but the smallest datasets. For instance, datasets used for supervised robot grasping models featuring accurate 3D models, object locations, and camera poses, are typically no larger than a thousand scenes (Fang et al., 2020; Chao et al., 2021). Manual labelling is also prone to mistakes and inaccuracies, especially when the goal is to determine exact poses as opposed to broad class labels. As such, the performance of systems following this approach tends to be limited by the scale and fidelity of human annotations.

1.2. Overview of Unsupervised Representation Learning

Despite these limitations, research in supervised learning has produced model architectures which are remarkably successful at extracting meaningful information from input images, such as convolutional neural networks, residual networks, and vision transformers (LeCun et al., 1995; He et al., 2016; Dosovitskiy et al., 2021). The main challenge in creating unsupervised vision systems therefore does not lie in identifying suitable inference architectures, but in finding alternative ways to train them, to ensure that even in the absence of human supervision, useful representations will be produced. The most basic technique is to ask the model to reproduce its input data from a learned, low dimensional representation, thereby encouraging it to encode as much of that data as possible. We focus on such *autoencoding* systems in Chapter 3.

A downside of this approach is that it does not ensure that the representation is suitable for any particular inference task: All we ask of it is being low dimensional. Another common technique is therefore to train via *proxy tasks*. Even if we do not possess the exact information which we want the representation to contain, we may train the model on a task which may only be solved by encoding this information. For instance, predicting future frames in a video depicting physically interacting objects requires understanding the physical properties of the objects in the scene, and will therefore encourage the model to learn about them. We discuss such dynamics models in Chapter 4. Similarly, synthesizing novel views of a given scene when provided with arbitrary camera parameters is only possible when the model understands the 3D geometry of that scene. Even if we do not have any data on 3D geometry, we may therefore train a model to learn about it on multi-view datasets. We discuss such models in Chapter 5.

In the literature, these types of training methods are also occasionally referred to as *self-supervised*. Lacking a clear dividing line between the two concepts, we avoid the term here and stick with the term *unsupervised*. They are also characterized as *generative*, since they are trained by generating images to match the training data. We give a brief overview of non-generative approaches to unsupervised representation learning in Section 2.5.

Beyond the ability to leverage larger datasets, unsupervised approaches also have some conceptual advantages. Compare for example a model trained via novel view synthesis to one for image classification. If the latter receives an object from an unseen category at test time, there is no useful answer it can possibly give. The novel view synthesis task on the other hand is still perfectly valid, giving the model a chance at generalizing. These are properties shared with successful unsupervised models in other domains (Brown et al., 2020; Rombach et al., 2021a).

Finally, we note that the idea of intelligent behavior emerging based on a system attempting to predict its input signals is also one of the leading theories of brain function, termed *predictive coding* (Clark, 2013). The study of representation learning is therefore relevent to the wider scientific question of how a high-level mental model of the world around us may be built using nothing but low-level observations, in either a machine or a brain. While the systems we study here cannot serve as models of the brain directly, they nevertheless provide evidence regarding what is feasible using predictive coding as a learning objective.

1.3. Using Structure to Evaluate and Visualize Representations

A key question in representation learning is how to judge the quality of learned representations. One option is to evaluate them on the downstream tasks for which we would like to use them. However, this often requires complicated experimental setups involving interactive environments, robotics, reinforcement learning, etc. It would also go against the idea that learned representations should be useful for a wide variety of tasks instead of a narrow set of benchmarks.

In this thesis we follow an alternative approach and instead evaluate whether our representations capture one of the most fundamental abstractions used to reason about scenes by machines and humans alike: *objects* (Lake et al., 2017; Greff et al., 2020b). Rigid objects have a number of properties that make them particularly useful for scene

representations: They occupy a continuous volume of geometry, move continuously through space while their shape remains constant, and are typically the primary target of interaction in control tasks. Of course, there are real-world phenomena which do not fit this abstraction: fluids, powders, and elastic materials, just to name a few. Nevertheless, scenes containing rigid objects provide a useful testbed to explore the ability of representation learners to arrive at useful abstractions, while also being relevant to many real-world tasks.

If we can show that a learned representation appropriately segments a given scene into its objects, this is therefore strong evidence that it is an appropriate abstraction for its contents. To demonstrate that our learned representations have this property, we will introduce structure to them, for instance by dividing them into a set of components. We will then examine the learned structure, and evaluate how closely it corresponds to the known structure of the input scene. If we find that it has indeed captured the objects present in the scene, this not only provides strong evidence that a meaningful representation has been learned, but also gives downstream tasks convenient access to the scene structure.

An alternative approach to this would be to train a simple predictor, such as a linear model, to demonstrate that the desired information, such as object segmentations, can be easily recovered from the learned representation. This approach is often referred to as (linear) probing. It does not impose any assumptions on the structure of the learned representation, but requires some amount of training data to train the predictor. In addition, it relies on the argument that the predictor is a weaker model than the representation learner, and that therefore the majority of the inference work is accomplished by the latter. This is sometimes hard to quantify, and being able to directly inspect a structured representation provides much more direct evidence.

1.4. Structure of this Thesis

We will start by introducing the basic techniques of unsupervised representation learning in the following chapter. The main body of the thesis is structured into three chapters, each investigating a different modality. Chapter 3 considers autoencoders, i.e. systems trained by reconstructing individual images. In Chapter 4, we extend these results to the dynamics domain, yielding models operating on videos. As a third step, we discuss systems learning to represent 3D geometry from multi-view data in Chapter 5. Each of these three main chapters is structured around an original work of research, which is

contextualized with the state of the art at the time, but also the developments since the original publication. Finally, we conclude the thesis in Chapter 6 with lessons learned and directions for future research.

Throughout this thesis, we will assume that the reader is familiar with Bayesian statistics, as well as commonly used machine learning algorithms and model components. For a general introduction to these topics, we recommend Murphy (2023). We provide more specific references where appropriate.

2. Foundations of Representation Learning

In this chapter, we introduce the foundational concepts of representation learning. We will start with briefly defining notation, before discussing the goals and challenges of representation learning. As the primary technical tool at our disposal, we will then introduce deep generative models.

2.1. Notation

While the focus of the thesis is unsupervised learning, we will frequently use supervised prediction models as components for larger systems, especially to formulate proxy tasks. In order to facilitate the discussion of such models, we will write the task of predicting random variable y based on input features x as $x \to y$.

We will generally take a probabilistic perspective on prediction tasks. That is, for a prediction task $x \to y$, we aim to learn models which are not just point predictors $f : x \to y$, but rather conditional probability distributions

$$p(y \mid x) = p_\theta(y), \quad \theta = f(x),$$

which are trained by maximizing the log-likelihood $\log p(y \mid x)$. In this sense, the arrow notation $x \to y$ may be viewed as an edge in a Bayesian network.

Notably, this probabilistic perspective is a strict generalization of the deterministic one: Minimizing the mean square error $(x - f(x))^2$, as is typically done for deterministic regression, is equivalent to maximizing the Gaussian log-likelihood (with fixed standard deviation σ), up to a constant factor:

$$\log p(y \mid x) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(x-\mu)^2, \quad \mu = f(x).$$

Similarly, the standard cross entropy loss for classification is equivalent to maximizing the log likelihood of a conditional categorical distribution.

The benefit of the probabilistic perspective is that it allows modeling the relationship between random variables using more complex distributions. We will frequently find that the conditional distribution of a variable is multimodal. In such cases, a deterministic (or Gaussian) estimate is likely to predict the mean of the true modes, resulting in inaccurate, or, in the case of images, blurry results. Predicting complex multimodal distributions allows properly accounting for the uncertainty inherent to the learning problem.

2.2. Representation Learning

Supervised learning shows us what sort of network architectures are suitable for inferring desired information y from input data x. The key challenge of representation learning is to define learning objectives which facilitate the learning of convenient representations z, along with an inference model $x \to z$, which contain similarly useful information without the need to provide direct supervision.

The most common way to accomplish this is through proxy prediction tasks: We choose observable features y' for which training data is available, and which likely also contain the desired information y. We then train the model to predict $z \to y'$, ideally pushing it to also encode y. In terms of information theory, the direct effect of this objective may be characterized by *mutual information* $I(z; y')$ between z and y' approximately equalling $I(x; y')$. If we choose an appropriate y' with sufficient connection to y, we will ideally also have $I(z; y) \approx I(x; y)$.

In this thesis, we are particularly interested in predictables y which facilitate visual scene understanding while being observable at scale without the involvement of human labor. In the simplest case, $y = x$, the task is simply to reconstruct the input data. While this objective by itself does not bias the model towards useful representations, we will see in Chapter 3 that with the appropriate structural constraints, it still allows us to learn meaningful abstractions. In Chapter 4, we explore predicting future frames in videos, resulting in models capable of dynamics prediction in latent space. Finally, in Chapter 5, we discuss how to obtain representations encoding the 3D geometry of the scene depicted in the input image, without requiring access to ground truth geometry as supervision. In this case, a suitable proxy task is to predict novel views of the scene,

as doing so is only possible with a good understanding of 3D geometry, and hence the model is forced to learn to encode the relevant information.

While such learning objectives ensure that z contains the relevent information from x, additional work is required to ensure that it is also a more useful representation - if all we wanted was access to the information in the input data, we could also be working with x directly. The intended benefit of representation learning is to infer a representation which is more convenient to work with for downstream tasks requiring specific information y than the original data x. This usefulness can manifest itself in a variety of ways:

- Less model capacity is required to predict $z \to y$, compared to $x \to y$.
- Fewer training examples are required to train the predictor $z \to y$, compared to $x \to y$.
- z is lower dimensional than x, and therefore requires less memory (and typically less compute) to process.
- z is structured in a way which simplifies working with it.

A variety of techniques for achieving these goals will be discussed throughout the thesis. Typically, they will involve some sort of bottleneck in the representation z, which ensures that it must encode the desired information in a low-dimensional, and possibly structured, manner.

2.3. Generative Representation Learning for Vision

Computer vision, i.e. the task of extracting useful information from natural images, is the primary domain to which representation learning is applied. This is partly because of its great importance for real-world applications - cameras remain by far the cheapest and most comprehensive method for obtaining detailed observations of the world around us, and understanding such observations is critical for virtually all autonomous robotic applications. But in addition, vision is extraordinarily well suited to benefit from representation learning: Compared to other types of data, images are particularly high-dimensional and redundant, motivating techniques for dimensionality reduction. At the same time, finding a suitable low dimensional representation for the contents of a scene is a highly non-trivial inference task. The promise of representation learning is to frontload this inference effort, such that concise and meaningful scene descriptions

may be handed to downstream tasks for further processing, massively reducing the difficulty of these learning problems (Bengio et al., 2013).

Another striking feature of vision is that we actually have a relatively precise understanding of what the *true* latent factors $\hat{z}$ behind the pixels x of a real-world image are: Objects occupy 3D space and scatter incoming light, which is then captured by the camera. We even possess advanced 3D engines capable of producing photorealistic images when provided with these latents in the form of (typically) polygonal meshes, thereby mapping $\hat{z} \to x$. An attractive and longstanding approach to vision is therefore to attempt to reverse this known process. This idea is known as *vision as inverse graphics* (Grenander, 1976). While it essentially reduces vision to a search problem, it is unfortunately an extremely hard one: The relationship between scene latents and images is highly nonlinear and even discontinuous. Consider for instance how moving an object effects the pixels of an image: In the simplest case, we might directly see it move in the image. However, it might also effect other areas of an image in highly non-linear ways, e.g. via shadows or reflections. It might even be the case that no change is visible at all, since the object is fully occluded. As a result, treating the problem of inverse graphics as a classical search problem quickly finds its limits in the face of an exponentially sized search space and an ill-conditioned objective function, motivating the involvement of machine learning. Some lines of work attempt to facilitate this by producing continuous, differentiable relaxations of standard 3D engines, in an effort to make the inverse problem easier (Liu et al., 2022). While these relaxations are differentiable in a technical sense, this is only achieved at the cost of introducing rendering noise, leading to difficult tradeoffs between rendering quality and ease of optimization.

Generative representation learning follows a different approach: In this paradigm, we treat available imagery not just as inputs to our vision model $x \to z$, but also as prediction targets y' for our proxy task $z \to y'$. This allows us to jointly *learn* a renderer $z \to y'$ and a vision model $x \to z$, both represented by some type of neural network. This sidesteps the issues with existing, monolithic graphics pipelines, and instead allows us to formulate the image generation process in a differentiable way. As we will see, we are still able to impose prior knowledge about the nature of the rendering process through architectural constraints on the generative network. We will start our investigation of this approach by introducing the probabilistic models which may be used to formulate the complex probability distributions involved.

2.4. Deep Generative Modelling

Except in very simple domains, attempting to understand a scene from 2D observations is almost always subject to uncertainty. If parts of the scene are occluded, there are typically multiple equally plausible hypotheses for the contents of the hidden areas. Similarly, a 2D observation may allow for multiple different 3D geometries explaining it, meaning that $p(z|x)$ will be non-deterministic and multi-modal. These sources of uncertainty are inherent to the task, and also experienced by humans. Extreme cases of the latter phenomenon, and human responses towards it, have been extensively studied in cognitive science under the terms *unstable perception* and *ambiguous figures* (Kornmeier & Bach, 2012). Additional uncertainty results from an inevitable lack of processing power: If we are faced with a task such as predicting the appearance of a scene from a different viewpoints, doing so with high per-pixel accuracy requires a detailed understanding of the underlying 3D geometry. If the scene contains complicated objects such as plants with hundreds of leaves, this quickly becomes infeasible, even for humans. A deterministic predictor will therefore predict the mean of possible plants, resulting in a blurry green cloud. A probabilistic model however would ideally be capable of sampling possible plants consistent with the limited information being processed from the input image.

The idea behind deep generative image modeling is to postulate a latent z and a learnable rendering process $p(x|z)$ producing images based on z. The way these distributions are represented though has a severe impact on the types of inferences that will be possible. In this section, we provide an overview over the most important methods to represent complex probability distributions using deep networks, and their inference capabilites. An overview is given in Table 2.1.

2.4.1. Variational Autoencoders

A straightforward way to employ a neural network f to specify an arbitrarily complex probability distribution over x is the *density network* (MacKay, 1995): Let z be a vector of latents with a simple prior distribution $p(z)$, such as a standard Gaussian. We may now use f to map z to the parameters of a conditional distribution $p(x|z)$. In the simplest case, $p(x|z)$ is another simple distribution, such as a product of independent Gaussians, in which case f will output a vector of means and standard deviations.

	VAE	GAN	Diffusion	SPN	Flow	Autoregressive
Likelihood $p(x_1, \ldots, x_n)$	$\sim$ †	✗	$\sim$ ‡	✓	✓	✓
Marginals $p(\mathbf{x_i})$	✗	✗	✗	✓	✗	$\sim$
Conditionals $p(\mathbf{x_i}\|\mathbf{x_j})$	✗	✗	✗	✓	✗	$\sim$
Sampling from $p(x_1, \ldots, x_n)$	✓	✓	✓	✓	✓	✓

Table 2.1.: Overview of the hardness of inference tasks in different families of deep probabilistic models. A ✓ indicates that the task can be solved exactly in polynomial time (in the size of the model), a ✗ means that it cannot. † May be approximated via importance weighting (IWAE) (Burda et al., 2016). ‡ May be similarly estimated via a variational lower bound (Kingma et al., 2021). Table is based on an illustration by Robert Peharz.

This setup now defines a joint distribution $p(z, x) = p(z)p(x|z)$ over z and x. The representational power of the neural network is transferred to the marginal distribution $p(x)$, meaning that it is possible to fit complicated datasets, such as images. Unfortunately, $p(x)$ is also intractable, as computing it exactly would require solving the integral $p(x) = \int p(z)p_\theta(x|z)dz$ over the neural network. This raises two questions: First, how may we learn the parameters of the neural network such that it fits the distribution of a given dataset, and second, how may we perform inference under this network, such that we may estimate the distribution $p(z|x)$ over latent representations z given an input example x?

Variational autoencoders (VAEs) (Kingma & Welling, 2014; Rezende et al., 2014) address both questions using *variational inference*, a technique which reframes the intractable inference tasks above as an optimization problem. For a thorough review of the topic, we refer the reader to Blei et al. (2017), and proceed here by sketching its application to VAEs. Let $q_\phi(z|x)$ be an approximation to the posterior, parameterized by ϕ. It is typically implemented as another neural network outputting the parameters of a simple distribution, similar to $p(x|z)$. We may now optimize the *evidence lower bound (ELBO)*

$$ELBO = p(x) - \mathrm{KL}(p(z|x)||q(z|x)), \tag{2.1}$$

where $\mathrm{KL}(p, q)$ stands for the KL-divergence between the distributions p and q, a non-negative measure of the difference between the two.

Maximizing the ELBO therefore has two effects: First, the likelihood $p(x)$ of the data is increased, fitting the model to the data. Second, the (non-negative) KL-divergence

is reduced, which means that $q(z|x)$ is becoming a closer approximation of the true posterior $p(z|x)$, allowing us to use it for inference. In this manner, variational inference manages to solve both tasks described above as a single optimization problem, whose global maximum is reached when the model $p(x)$ matches the data distribution, and $q(z|x) = p(z|x)$, meaning KL$(p(z|x)||q(z|x)) = 0$.

On its face, the ELBO appears intractable, and both of its components, in fact, are. However, as a whole, it can be shown that

$$ELBO = \mathbb{E}_{q(z|x)} \left[p(x, z) - q(z|x) \right], \tag{2.2}$$

a term that may be readily estimated by sampling from $q(z|x)$. Additionally, for many choices of distribution for $q(z|x)$, its gradient may be estimated using the *reparametrization trick* (Kingma & Welling, 2014). As a result, maximizing the ELBO is the most commonly used method for training VAEs.

We will frequently make use of VAEs in this thesis, as they do not just allow for modeling complex probability distributions, but are also inherently suited to representation learning, as they operate by encoding the input data in the latent variable z. In this setting, the variational estimate $q(z|x)$, which we introduced above as a crutch to deal with the intractability of the model, becomes its key deliverable: At test time, it allows us to quickly encode input data into a lower dimensional representation, which is what we desire for vision.

2.4.2. Sum-Product Networks

While VAEs are very effective at leveraging the expressive power of neural networks to yield flexible distributions, a wide variety of probabilistic inference tasks remain intractable in them. In particular, computing arbitrary marginal distributions (and, consequently, conditional distributions) exactly is exponentially hard. A natural question to ask is therefore: What is the most powerful model in which such inference tasks may be solved in polynomial time, relative to the size of the model? This question is explored by the literature on *tractable* probabilistic models, see e.g. Choi et al. (2020) for an overview.

One of the most commonly discussed types of model are *sum-product networks* (SPNs): Starting from the realization that the hardness of exact (discrete) inference can be measured by the number of sum and product operations necessary to compute e.g. $P(x) = \sum_z P(z)P(x|z)$, they represent probability distributions as rooted directed

acyclic graphs (DAGs) consisting of sum nodes, product nodes, and primitive distributions at the leaves. In an SPN over random variables $\mathbf{X}$, each node N represents a distribution $p(\text{scope}(\mathsf{N}); \mathsf{N})$ over a subset of $\mathbf{X}$ refferred to as $\text{scope}(\mathsf{N}) \subseteq \mathbf{X}$. The overall distribution represented by the model as a whole may be read off at the root node R as $p(\mathbf{X}; \mathsf{R})$. Each leaf node stands for a primitive distribution, such as Gaussian or categorical, over one or more random variables. Mixing discrete and continuous distributions in the same model is possible (Molina et al., 2018). A product node P multiplies the distributions of its children $\mathbf{ch}(\mathsf{P})$, i.e., $p(\text{scope}(\mathsf{P}); \mathsf{P}) = \prod_{\mathsf{C_i} \in \mathbf{ch}(\mathsf{P})} p(\text{scope}(\mathsf{C_i}); \mathsf{C_i})$. To ensure that this is a proper distribution, the children's scopes need to be disjoint, and $\text{scope}(\mathsf{P}) = \bigcup \text{scope}(\mathsf{C_i})$. A sum node S computes a mixture model $p(\text{scope}(\mathsf{S}); \mathsf{S}) = \sum_{\mathsf{C_i} \in \mathbf{ch}(\mathsf{S})} w_i p(\text{scope}(\mathsf{S}); \mathsf{C_i})$ over its children's distributions, where the weights w_i are learnable nonnegative parameters with $\sum w_i = 1$. The children's scopes need to be identical to ensure a proper distribution. The full distribution $p(\mathbf{X} = \mathbf{x})$ may be evaluated in linear time in the size of the network by evaluating the likelihood of $\mathbf{x}$ under the leaf distributions, and propagating the results to the root according to the rules above.

The main benefit of SPNs is their capability for computing arbitrary marginals in linear time as well: Summing out an arbitrary variable X reduces to doing so in the primitive leaf distributions, as the sum operator is propagated through the interior nodes as follows:

$$\sum_X p(\text{scope}(\mathsf{P}); \mathsf{P}) = \sum_X \prod_{\mathsf{C_i} \in \mathbf{ch}(\mathsf{P})} p(\text{scope}(\mathsf{C_i}); \mathsf{C_i}) \tag{2.3}$$

$$= \prod_{\mathsf{C_i} \in \mathbf{ch}(\mathsf{P})} \sum_X p(\text{scope}(\mathsf{C_i}); \mathsf{C_i}) \tag{2.4}$$

$$\sum_X p(\text{scope}(\mathsf{S}); \mathsf{S}) = \sum_X \sum_{\mathsf{C_i} \in \mathbf{ch}(\mathsf{S})} w_i p(\text{scope}(\mathsf{C_i}); \mathsf{C_i}) \tag{2.5}$$

$$= \sum_{\mathsf{C_i} \in \mathbf{ch}(\mathsf{S})} w_i \sum_X p(\text{scope}(\mathsf{C_i}); \mathsf{C_i}) \tag{2.6}$$

In the case of product nodes, this follows from the fact that X only appears in at most one of the children's scopes, for sum nodes, it is simple algebra. Therefore, when easy to marginalize leaf distributions are used (or even univariate ones), marginal inference can be accomplished in a single linear pass through the network. Similarly, computing a conditional probability $p(\mathbf{Y}|\mathbf{Z}) = p(\mathbf{Y}, \mathbf{Z})/p(\mathbf{Z})$ also takes linear time, as it reduces to the computation of two marginals. As each of these terms is tractable and differentiable, they may also straightforwardly be used for gradient-based likelihood optimization.

Traditionally, the main downside of SPNs is their relative lack of expressive power, which makes it difficult to capture complex distributions, such as image datasets. This stems from the fact that they model correlations between variables via discrete enumeration (i.e. , mixtures), rather than functional dependencies as in neural networks. To push the boundaries of tractable modeling, a number of frameworks have been developed to scale the size of SPNs with the tools of deep learning (Peharz et al., 2019; 2020)[1]. These feature large, uniform model structures which are well-suited to GPU computation. Recently, further improvements have been made by *distilling* the latent structure of neural generative models into SPNs (Liu et al., 2023). Nevertheless, a performance gap relative to the other methods in this section remains, particularly for high-dimensional data such as images.

2.4.3. Generative Adversarial Networks

While VAEs and SPNs are the main types of generative models which we will be using in this thesis, we will briefly introduce two other popular methods for context, the first of which being *generative adversarial networks (GANs)* (Goodfellow et al., 2014). GANs provide an alternative way to train density networks, which they refer to as the *generator*. This is accomplished by introducing a *discriminator* network, whose task it is to distinguish images sampled from the generator, and images from the training dataset. The generator's objective is then to fool the discriminator, i.e. to maximize the probability that its samples are classified as coming from the training dataset. Both networks can be trained jointly, and the global optimium of the resulting optimization is reached when the generator exactly matches the training distribution.

GANs have been used to syntesize high-resolution photorealistic images (Karras et al., 2021). They benefit from the very limited number of assumptions underlying the training procedure: In principle, any differentiable sampling procedure may be used as a generator, no matter its structure or probabilistic form. However, this also severely limits their inference capabilities: In their basic form, GANs purely allow for sampling new examples from the distribution they represent, but not for any kind of inference. It is possible to train an inference network for $x \rightarrow z$ post-hoc, but this complicates the already complex training setup.

[1] While a significant amount of work leading up to this thesis was done in contribution to these and related works (see Chapter 7), we do not discuss them here in detail, as they are tangential to the topic of scene understanding.

2.4.4. Diffusion Models

Diffusion models (Ho et al., 2020) have recently emerged as the most powerful model to fit distributions over images (Dhariwal & Nichol, 2021), and have been employed on large scale unsupervised datasets (Rombach et al., 2021a). They function by postulating a diffusion process, in which images are gradually blurred by adding Gaussian noise of increasing variance. A neural network is then trained to reverse this process, receiving artificially blurred images as inputs, and supervision in the form of the corresponding originals. It can be shown that in the limit, feeding this denoising network with pure Gaussian noise at test time will allow sampling from the data distribution (Song & Ermon, 2019). Structurally, this setup bears similarities to variational autoencoders, with the diffusion process specifying a generative model, and the denoising network taking the role of a variational posterior. Kingma et al. (2021) leverage this insight to formulate a variational bound on the sample likelihood.

While diffusion models currently offer unmatched image generation performance, they also come with some downsides. First, image generation is somewhat slow, as both diffusion and denoising are iterative processes, with the latter typically requiring hundreds of evaluations of the inference network. Second, training relies on access to examples from the target distribution, which makes it difficult to learn distributions over latent variables. Rombach et al. (2021a) show that latent diffusion is possible, but do so in a very particular latent space for which they have access to a high quality one-to-one mapping to the observed dataset.

2.5. Other Approaches to Representation Learning

Before we proceed with discussing concrete examples of generative representation learning, we give a brief overview over alternative approaches to representation learning which do not rely on predicting or reconstructing images.

One widely used idea is *contrastive learning*. Here, instead of predicting images, the proxy task used for training is chosen to be a classification task: The model is asked to identify data associated with the input scene (*positive examples*) from unrelated *negative examples*. For instance, positive examples might be obtained by modifying the inputs image using operations such as cropping, rotation, or adding noise, whereas negative examples are typically unrelated images from the dataset. If a learned representation z allows the model to successfully solve this task, it must contain relevant information

about the image contents. More sophisticated formulations of this idea swap the basic classification objective for losses pushing associated, positive examples to stay close together in representation space, whereas unrelated netative examples are penalized for doing so. Chen et al. (2020a) showed with their SimCLR system that representations learned in this manner on ImageNet allow for a simple linear classifier to yield competitive image classification performance when compared with supervised approaches. Even richer representations were obtained by CLIP (Radford et al., 2021) using a large dataset of captioned images. Here, the task is to match an image representation to its corresponding caption, which causes the representation to encode the information contained in the captions.

A limitation of contrastive learning is that the richness of the representations will be upper-bounded by what is necessary to solve the classification task. This means a large number of negative examples is required to make these tasks sufficiently challenging. Even so, outside of small synthetic examples (Kipf et al., 2020), representations will generally not contain details such as precise object positions, which would be necessary to facilitate real-world interaction. This is particularly apparent for CLIP, where only information relevant for captioning will be encoded, with object positions generally not being among them. As a result, CLIP and its derivative models offer only limited spatial reasoning capabilities.

A recent series of works strives to address this issue in a surprising way: Grill et al. (2020) demonstrated that representations can be learned in an unsupervised way even without using *any* negative examples. Instead, the only objective is that different augmented (positive) views of the same scene should yield the same representation. On the surface, it seems like such a system would collapse during training, as a global optimum is reached when all scenes are mapped to the same, constant representation. However, Grill et al. (2020) show that via a model distillation scheme, in which one of the positive examples is processed by a frozen *teacher* network, such a collapse can be prevented. By combining this approch with modern vision transformers, Caron et al. (2021) were able to demonstrate impressive representation learning performance, improving upon SimCLR. Very recently, further engineering and large scale datasets allowed Oquab et al. (2023) to learn real-world image representations which yield convincing object segmentations and depth maps upon linear probing.

3. Unsupervised Representations from Autoencoding

In this chapter, we start our investigation of representation learning with representations derived from autoencoding tasks. Here, the main loss applied to the latent representation z expresses that it should be possible to reconstruct the input data x as accurately as possible. Typically, such models are based on the variational autoencoder approach discussed in Section 2.4.1. This is the simplest and most universal type of proxy task for representation learning, as it is applicable to any kind of data. However, while unstructured autoencoders, where z is a simple vector and encoder and decoder are e.g. MLPs, can yield compressed and denoised representations (Kingma & Welling, 2014), achieving scene understanding requires additional work. Here, we explore various methods for decomposing images into objects in an unsupervised way. As discussed in Section 1.3, object segmentations are of interest to many downstream tasks, and are also a key indicator that the model has learned a useful abstraction of its observation. We will therefore frequently use the accuracy of predicted segmentations to measure the quality of learned representations.

The principal tool we use for achieving unsupervised object segmentation are structural inductive biases: Losses and architectural components which encourage the model to not only learn a latent representation which can reconstruct the input image, but one that does so in a particular way. As a model designer, one has great freedom in implementing the various components of VAEs left unspecified in Section 2.4.1: The latent z might be a complex data structure, and both the encoder network $q(z|x)$ and the decoder network $p(x|z)$ may contain complex, and perhaps not even learned machinery for e.g. graphics rendering. As long as all components are differentiable, even a model with such complexities may still be trained via the ELBO – and various techniques to handle discrete components exist, as well.

We will now introduce two specific models, AIR (Eslami et al., 2016), and its extension, SuPAIR. SuPAIR was published in Stelzner et al. (2019), and the following two sections contain the contents of that publication in edited form.

3.1. Attend-Infer-Repeat (AIR)

A notable early model for unsupervised object detection of this nature is *Attend-Infer-Repeat* (AIR) (Eslami et al., 2016), which incorporates VAEs as object models within a scene generation process and learns a recurrent neural network (RNN) to dynamically detect multiple objects composed in a scene. Other early examples of structured models are (Johnson et al., 2016), which incorporate VAEs into a latent switching linear dynamical system to infer behavioral patterns from mice depth videos, and SketchRNN (Ha & Eck, 2018), which uses an RNN to infer the trajectory of a pen from given sketches.

None of these models require supervision in the form of observed latent representations. Instead, the nature of these representations is specified through the model structure. To this end, the structure is imbued with available prior knowledge, such as the rules of object interaction, pen stroke rendering, or Markovian assumptions of biological behavior. Other parts, such as the appearance of objects or typical pen trajectories are subject to learning. Exact inference is intractable, either because the global model structure is already intractable itself or because intractable components such neural networks are used.

However, the generality of variational inference enables impressive results in such highly intractable models. Some of the key contributing factors in these techniques are inference networks and amortization (Gershman & Goodman, 2014; Kingma & Welling, 2014), the reparameterization trick (Kingma & Welling, 2014; Titsias & Gredilla-Lázaro, 2014; Schulman et al., 2015), and techniques for reducing the variance in gradient estimates (Mnih & Gregor, 2014).

Unlike previous work, e.g. (Lempitsky & Zisserman, 2010), the Attend-Infer-Repeat (AIR) framework approaches the problem of object detection and scene understanding in an unsupervised way, using a Bayesian approach. Following the VAE framework, it is assumed that a given image x is generated according to some generative process $p(x, z) = p(x \mid z)\, p(z)$. In this framework, scene analysis is cast into standard Bayesian inference, i.e. we condition on a scene x and infer the posterior $p(z \mid x) \propto p(x \mid z)\, p(z)$.

The posterior might be used to derive a single scene description via a MAP solution $\arg\max_z p(z \mid x)$, or the whole posterior might be incorporated into a downstream decision making process.

Specifically, Eslami et al. (2016) introduced the following assumptions. The scene descriptor z is organized in N blocks, i.e. $z = (z^1, \ldots, z^N)$, corresponding to N objects in the scene. Each block $z^i = (z^i_{\text{where}}, z^i_{\text{what}})$ contains a description for its respective object, where z_{where} contains pose parameters (translation and scale) and z_{what} describes object appearance (object class, texture, etc). Since the number of objects in a scene varies, N is also a random variable, taking values between zero and some N_{max}.

Assuming a priori independence among the objects and the descriptor components, the prior of the entire scene description z takes the form $p(z) = p(N) \prod_{i=1}^{N} p(z^i_{\text{where}})\, p(z^i_{\text{what}})$. The number of objects N is straightforwardly modeled via e.g. a categorical or (truncated) geometric distribution. The distribution over pose parameters z^i_{where} can also take a simple form, such as a uniform distribution over a suitable range. In order to describe the appearance of objects, however, a more expressive model is required. The approach taken by Eslami et al. (2016) is to leverage a variational autoencoder (VAE) (Kingma & Welling, 2014) using the Gaussian distributed z^i_{what} as its latent code. z^i_{what} is processed by a neural net, generating an *object draft* y^i, the visual appearance of a single object. Each y^i is then transformed by its corresponding pose parameters z^i_{where} and inserted into a private canvas, denoted as $\tilde{y}^i$. Finally, the pixelwise means of scene x are generated as the sum of all $\tilde{y}^i$ which are present in the scene, i.e., for which $i \leq N$. The final distribution over x is then given by an isotropic Gaussian with these means and fixed variance.

Inference in AIR is addressed by variational inference techniques. Following the compositional structure of the model, an RNN is employed as the inference network, at each step outputting a variational distribution $q(z^i_{\text{where}}, z^i_{\text{what}}, z^i_{\text{pres}})$, conditioned on the input x and the previously inferred object descriptors. Here, the binary variable z^i_{pres} indicates at each inference step whether the i^{th} object is present or if all objects have been found and the inference process should terminate. This effectively parameterizes $q(N)$ as a series of yes/no decisions, also called a stick-breaking process, such that

$$q(N{=}n) = (1 - q(z^{n+1}_{\text{pres}})) \prod_{i=1}^{n} q(z^i_{\text{pres}}). \tag{3.1}$$

Both the model and inference parameters are learned by stochastic optimization of the ELBO. To yield gradient estimates of the ELBO, the reparameterization trick (Kingma & Welling, 2014; Titsias & Gredilla-Lázaro, 2014) is employed where possible. For the

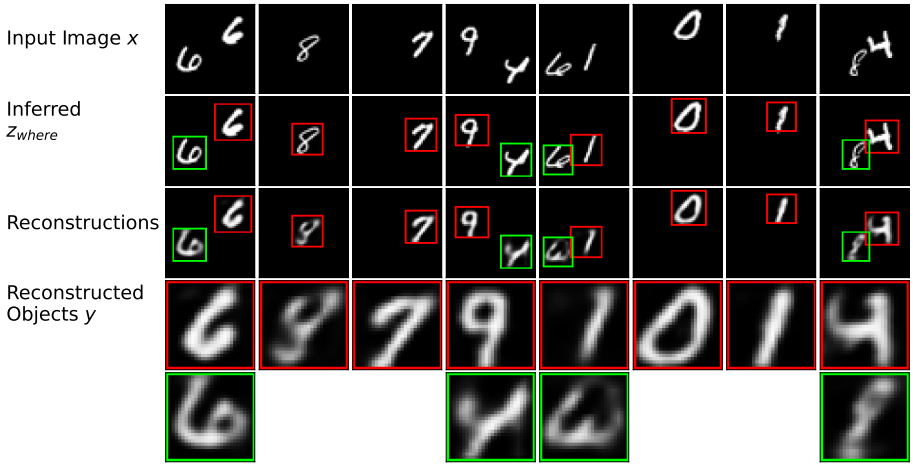

Figure 3.1.: AIR (Eslami et al., 2016) takes an input image x and infers object positions z_{where} in the form of bounding boxes. Their contents y are modeled via variational autoencoders, yielding a reconstruction of the overall scene. This figure was produced using the implementation by Bingham et al. (2018).

discrete z^i_{pres}, score estimators with variance reduction techniques are used (Mnih & Gregor, 2014; Schulman et al., 2015).

The results of Eslami et al. (2016) impressed, because they showed that with the right structural biases, a notion of objects could be learned directly from a dataset of images, without any human supervision. As illustrated by Figure 3.1, experiments showed this effect on MultiMNIST, a dataset of images consisting of up to three handwritten digits. Additional experiments were conducted in a custom 3D environment, but subsequent literature has focused on the easier to reproduce 2D case.

However, learning and inference in AIR – and other structured probabilistic models – is far from perfect. One issue is the non-trivial interplay between the generative model and the inference network, which can cause the generative model to adapt to a too weak inference network (Cremer et al., 2018). On the other hand, improving the quality of the variational approximation can also be detrimental for learning (Rainforth et al., 2018b). The severity of these issues scales with the complexity of the latent variables to be estimated. In the following section, we therefore explore how the number of latent variables in AIR may be reduced by integrating tractable components into the model.

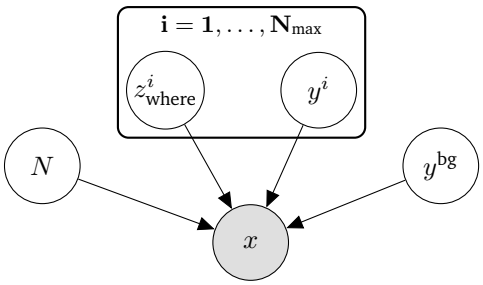

Figure 3.2.: The generative model forming the basis for SuPAIR. The scene x is modeled to be composed of *background* y^{bg} and up to N_{max} objects drafts y^i (denoted in plate notation). *Position* and *scale* of object y^i within the scene is encoded in the latent parameter vector z^i_{where}. The number of present objects is given by the discrete variable N.

3.2. SuPAIR: A Generative Model with Tractable Components

The biggest intractable part of AIR are the object VAEs, since the object images are represented by high dimensional latent vectors z_{what}, compared to the small number of variables $z_{\text{where}}, z_{\text{pres}}$ required for position and presence. Here, we therefore explore replacing the object VAEs with tractable models, drastically reducing the dimensionality of the latent variables to be estimated. To this end, we utilize models which are both tractable yet expressive, namely SPNs (Section 2.4.2). Their ability to efficiently compute exact marginals plays a key role in our derivation of the variational lower bound for our model – called *Sum-Product Attend-Infer-Repeat* (SuPAIR) – and allows us to robustly handle noisy backgrounds and treat object occlusions in a principled manner.

Replacing intractable VAEs with SPNs leads to a dramatic reduction of the inference effort. In particular, our inference network does not need to predict latent object codes nor do we require object reconstructions—rather, we are able to directly assign well-calibrated likelihood scores to proposed scene descriptions. This approach can be understood as a form of *Rao-Blackwellization* and drastically reduces the variance in gradient estimates for the variational objective. As shown in our experiments, SuPAIR yields significant reductions in training time as well as increased robustness compared to the original AIR system. The code for SuPAIR is available online.[1]

[1] `github.com/stelzner/supair`

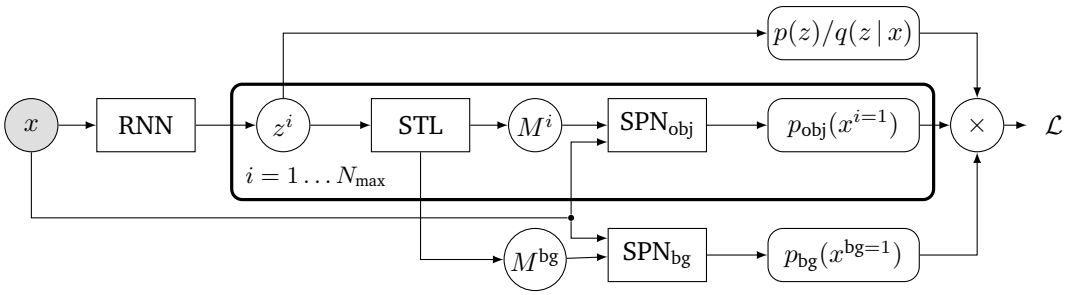

Figure 3.3.: Computation graph for learning in SuPAIR. Given an image x, an RNN infers the number, position and size of objects, encoded in object descriptors z^i. The *spatial transformer layer* (STL) then computes object and background masks (M^i, M^{bg}) which indicate the marginalization domains for the object SPN and the background SPN. Due to our deterministic cut-and-paste interaction model and the tractable marginalization property in SPNs, a Rao-Blackwellized evidence lower bound is obtained from the SPNs' outputs.

3.2.1. Sum-Product Attend-Infer-Repeat

We now develop the Sum-Product AIR (SuPAIR) framework, following the generative model shown in Fig. 3.2. There are three main differences to the original AIR system:

1. We *directly* model the distribution over object drafts y^i with SPNs. Therefore, we do not need to infer latent object codes during learning, instead, we effectively marginalize both the latent SPN variables *and* y^i, which can be seen as a type of *Rao-Blackwellization*, speeding up the training process.

2. We incorporate a background model y^{bg} which allows for the capture of image noise and increases the model's robustness.

3. We use an alternative interaction (scene rendering) model, which plays well with efficient SPN inference.

Let us now devise the SuPAIR model in detail, touching in turn upon the priors, the interaction model, how to marginalize objects and the background, and finally how to perform variational inference. As a whole, this results in the SuPAIR computation graph shown in Fig. 3.3.

Objects, Background, and Scene Priors

The SuPAIR model generates a scene x of size $B \times B$. Each scene contains $0 \leq N \leq N_{\text{max}}$ objects, where the prior over N is modeled as a truncated geometric distribution. Each object i has latent pose parameters z^i_{where}, a 4-tuple representing the coordinates and size of the object's bounding box. The prior over each z^i_{where} is modeled as a uniform distribution with suitable bounds. To prevent highly or even fully occluded objects, we add an unnormalized penalty term modeled as a Gamma distribution over each object's *occlusion ratio*, the ratio of its pixels which are occluded in the scene.

The visual content y^i of object i is generated by a RAT-SPN (Peharz et al., 2019) over an $A \times A$ pixel array. To model the individual pixels, we employ univariate Gaussians at the leaf nodes. We let the objects share the SPN's parameters, i.e. all objects have the same prior distribution, denoted $y^i \sim p_{\text{obj}}(\cdot)$. However, objects could also be easily equipped with private SPNs, and the SPNs could also be conditioned on some context, such as a class variable. While SuPAIR does not explicitly include latent object codes like AIR's z_{what}, if necessary such representations can still be obtained from the object SPN via the procedure proposed by Vergari et al. (2018). Furthermore, and differently from AIR, we assume a background model y^{bg} of the same size as the canvas, i.e. $B \times B$, also represented by a RAT-SPN with Gaussian leaves. We denote this density as $y^{\text{bg}} \sim p_{\text{bg}}(\cdot)$. Background and present objects are then combined into a scene according to an interaction (rendering) model, which is described next.

Cut-and-Paste Interaction Model

To render a scene, we follow a natural *cut-and-paste* approach, as illustrated by Fig. 3.4. First, the background is inserted into the canvas, and then, one after the other, each object is inserted, occluding portions of the background and possibly other previously drawn objects. We assume that objects with smaller indices are in front of objects with larger indices, similar to how graphics engines employ z-buffers.

More formally, let the latent factors of SuPAIR N, y^{bg}, and all y^i, z^i_{where} be given. To avoid cluttered notation, we sometimes use $y^{1\dots N_{\text{max}}}$ for all objects, y for all objects *and* background, and z for all scene descriptors. Now, we define $B \times B$ *object indicator matrices* I^i which are 1 within the transformation window of the i^{th} object, and 0 elsewhere. Furthermore, we recursively define the *mask matrices* $M^i := I^i \prod_{j<i}(1 - I^j)$ and $M^{\text{bg}} := \prod_{i=1}^{N}(1 - I^j)$. Note that $\{M^i\}_{i=1}^{N}$ and M^{bg} encode a partition of the scene,

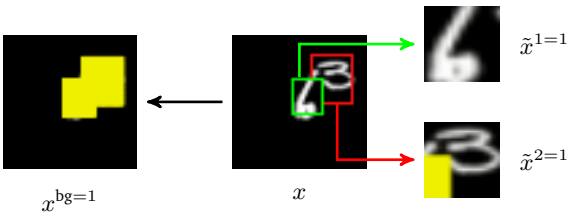

Figure 3.4.: Cut-and-Paste Interaction Model: We assume that objects occlude the background and each other. Consequently, at inference time, we use the scene description z to decompose scenes x into their parts, the (rescaled) objects $\tilde{x}^{i=1}$ and the background $x^{\text{bg}=1}$. Occluded parts of background and objects are unobserved and drawn in yellow. The marginal likelihood of each component can then be evaluated using SPNs.

i.e. they are all binary $B \times B$-matrices and sum up to the constant 1 matrix. In particular, they indicate visible pixels for the objects (if present) and the background. We define the scaled version of the objects $\tilde{y}^i = \tilde{y}^i(y^i, z^i_{\text{where}})$ as the result of transforming y^i according to z^i_{where}, and inserting it into an empty (black) $B \times B$-canvas. Now, the interaction model can be formalized as

$$p(x \mid y, z) = \delta_x \left[M^{\text{bg}} \times y^{\text{bg}} + \sum_{i=1}^{N} M^i \times \tilde{y}^i \right], \tag{3.2}$$

where $\times$ denotes element-wise multiplication, and $\delta_x[\cdot]$ the Dirac delta at x. Note that in this interaction model each pixel in x is deterministically given by either the background y^{bg} or one of the transformed objects $\tilde{y}^i$. This allows us to marginalize the scene components y, which will tremendously speed up the inference and learning process.

Marginalizing Objects and Background

Putting everything together, we arrive at the joint distribution of our model, given as

$$p(x, y, z) = p(x \mid y, z) p_{\text{bg}}(y^{\text{bg}}) p(N) \prod_{i=1}^{N_{\text{max}}} p_{\text{obj}}(y^i) p(z^i_{\text{where}}). \tag{3.3}$$

By conditioning on x, we can obtain a posterior over y and z. However, since we use SPNs as priors for the objects and the background, and since we use the *deterministic* interaction model (3.2), we are able to marginalize y:

$$p(x, z) = \int_{\dots} \int p(x, y^{\text{bg}}, y^{1 \dots N_{\text{max}}}, z) \, \mathrm{d}y^{\text{bg}} \mathrm{d}y^{1 \dots N_{\text{max}}}. \tag{3.4}$$

This is highly beneficial for learning, since we are ultimately interested in the posterior over the scene description z, and not in the latent scene components y. Note that each of the multi-dimensional integrals in (3.4) can be written as a series of single-dimensional integrals.

Marginalizing non-present objects (i.e. where $i > N$) is trivial, since the interaction model (3.2) does not depend on them. Thus, the integration over their appearances y^i can be switched with the product in (3.3). Since their priors $p(y^i)$ are normalized, the integrals over them evaluate to 1, i.e. they are effectively removed from the product.

Next, we discuss how to marginalize y^{bg}. To this end, let the entries in y^{bg} be split into two groups $y^{\text{bg}=0}$ and $y^{\text{bg}=1}$, indicated by $M^{\text{bg}} = 0$ and $M^{\text{bg}} = 1$, respectively. Since the interaction model $p(x \mid y, z)$ does not depend on $y^{\text{bg}=0}$, these pixels are just marginalized from the prior over y^{bg} in (3.3). On the other hand, for $y^{\text{bg}=1}$, we know that for all i, $M^i = 0$ at these pixels, so that (3.2) puts all probability mass on the event $y^{\text{bg}=1} = x^{\text{bg}=1}$, where, akin to the definition above, $x^{\text{bg}=1}$ are those pixels in x where $M^{\text{bg}} = 1$. Consequently, marginalizing over y^{bg} yields $p(x, z) =$

$$p_{\text{bg}}(x^{\text{bg}=1}) \int_{\dots} \int p(x^{\text{bg}=0}, y^{1 \dots N}, z) \, \mathrm{d}y^{1 \dots N}. \tag{3.5}$$

Here, we can readily draw on the remarkable property of SPNs, which allows us to evaluate the marginal $p_{\text{bg}}(x^{\text{bg}=1})$ using a *single network pass*. Moreover, by applying automatic differentiation, we can obtain the gradients required for learning without any extra effort.

We can, in principle, proceed in a similar way as with y^{bg} and eliminate one object y^i after the other, as there is no difference in the functional form for objects and background in (3.2). However, note that while the background y^{bg} enters *directly* in (3.2), each object y^i is incorporated via its *transformation* $\tilde{y}^i(y^i, z^i_{\text{where}})$. Following the same argument as above, the i^{th} object is marginalized from (3.5), by integrating y^i over the event $\tilde{y}^{i=1}(y^i, z^i_{\text{where}}) = x^{i=1}$, where $\tilde{y}^{i=1}$ and $x^{i=1}$ are the pixels indicated by $M^i = 1$. Unfortunately, when standard image transformations such as bilinear

interpolation are employed, the mapping from y^i to $\tilde{y}^i$ is many-to-one, rendering this integral analytically intractable, even for SPNs.

One approach for addressing this problem is to choose a transformation which makes each pixel in y^i either independent of, or uniquely determined by, $\tilde{y}^i$, such as the probabilistic hard downsampling procedure described in the following. We assume that the size $A \times A$ of the objects y^i is chosen such that z^i_{where} only involves downsampling, i.e. the objects are modeled at their maximal resolution. Let $W \times H$ be the size of the transformed object, as determined by z^i_{where}. By interpreting the coefficients of the bilinear transform from size $W \times H$ to $A \times A$ as probabilities, we randomly select a pixel from y^i for each pixel in $x^{i=1}$. Now, given such a hard mapping, $x^{i=1}$ can be evaluated as a marginal of p_{obj} such that the joint (3.5) reduces to $p(x, z) =$

$$p(N)p_{\text{bg}}(x^{\text{bg}=1}) \prod_{i=1}^{N} p_{\text{obj}}(x^{i=1}) \prod_{i=1}^{N_{\text{max}}} p(z^i_{\text{where}}). \tag{3.6}$$

For a fixed downsampling scheme, only one random sub-marginal of the SPN is queried with $x^{i=1}$, and thus "informed" during learning. Ideally, we should marginalize over all possible downsampling schemes in order to get a smooth transformation in expectation. This marginalization can be efficiently incorporated into stochastic variational optimization, by simply drawing a new downsampling scheme in each iteration. This introduces some additional noise into the learning process, but does not introduce bias into the SuPAIR model.

In practice, we choose a more pragmatic approach by using a simple but biased method to approximate probabilistic downsampling. Rather than evaluating the object SPN at a random sub-marginal, we scale the content of the object's bounding box $x^{i=1}$ to the SPN's native dimensions $A \times A$, approximating the inverse of $\tilde{y}^i$. We can then evaluate the SPN on this scaled version of $x^{i=1}$. We also scale the content of M^i to respect marginalized pixels stemming from occlusion. Since this scaling of the input introduces "artificial" dimensions in the model, we renormalize the result by $p \leftarrow p^{W_i H_i / A^2}$. In sum, the approximate version of probabilistic downsampling is computed as $p(x, z) =$

$$p(N)p_{\text{bg}}(x^{\text{bg}=1}) \prod_{i=1}^{N} p_{\text{obj}}(\tilde{x}^{i=1})^{W_i H_i / A^2} \prod_{i=1}^{N_{\text{max}}} p(z^i_{\text{where}}) \tag{3.7}$$

where $\tilde{x}^{i=1}$ denotes the scaled input.

Variational Inference

The joint, obtained from either Eq. (3.6) or Eq. (3.7), can now be used to perform posterior inference over the latent scene description z. To this end, we employ an RNN as an inference network, representing the variational distribution $q_\phi(z \mid x)$, which detects one object in each iteration. We process the input scene x using an LSTM layer with 256 hidden units, the output of which is fed into two fully connected layers with 50 and 9 units, respectively. The 9 outputs are the parameters for SuPAIR's scene descriptors for a single object, namely 8 parameters (means and standard deviations) for the 4-dimensional $q(z^i_{\text{where}})$, modeled as a Gaussian, and one parameter for the Bernoulli $q(z^i_{\text{pres}})$. As in AIR, the latter is used to parameterize $q(N \mid x)$ as defined in (3.1). We summarize all parameters of the inference networks in ϕ and all SPN parameters in θ. The prior parameters of z are kept fixed.

We simultaneously learn the SuPAIR model and the inference network by optimizing the *evidence lower bound* (ELBO)

$$\mathcal{L} = \mathbb{E}_{q_\phi(z \mid x)} \left[\log p_\theta(x, z) - \log q_\phi(z \mid x) \right] \tag{3.8}$$

with respect to θ and ϕ. The ELBO can be estimated using Monte Carlo samples from the variational distribution q, yielding an unbiased but potentially high-variance estimator. The reparameterization trick (Kingma & Welling, 2014; Schulman et al., 2015) is one of the most effective techniques to reduce the gradient's variance, and is straightforwardly applied to z^i_{where}. To ensure differentiability of the likelihood with respect to z^i_{where}, we use bilinear interpolation when computing the mask matrices M. This allows the masks to take non-integer values, slightly blending the scene components at their edges. The main invariant of the cut-and-paste model is still maintained, however, as the mask matrices continue to sum to one. Following AIR, we refer to this interpolation operation as a *spatial transformer layer* (STL), originally proposed as a subcomponent in (Jaderberg et al., 2015).

While the discrete variable N cannot be easily reparameterized, its expectation in (3.8) can be computed *exactly* via enumeration for only a low computational cost. Since our variational distribution factorizes over z, we can write the ELBO as $\mathcal{L} =$

$$\mathbb{E}_{q(z_{\text{where}})} \left[\sum_{n=0}^{N_{\text{max}}} q(n \mid x) \left(\log p(x, z) - \log q(z \mid x) \right) \right]. \tag{3.9}$$

As mentioned above, the outer expectation in (3.9) is handled by stochastic variational inference and the reparameterization of z_{where}. The expectation over N is treated as a

Table 3.1.: Number of parameters for SuPAIR and AIR. Note that AIR does not feature a (parameterized) background model, while SuPAIR does not require data-dependent baselines.

	SuPAIR	**AIR**
Inference network	2,836,477	2,879,488
Object model	286,560	344,884
Background model	90,108	0
Baselines	**0**	2,879,745
Total	3,213,145	6,104,117

sum over the possible numbers of objects. To compute it, we merely require $N_{\text{max}} + 1$ evaluations of the background network, to compute $p_{\text{bg}}(x^{\text{bg}=1})$ for each of the possible occlusion masks. Similarly, the object SPN only needs to be evaluated once for each of the N_{max} potential objects, since objects with higher indices are behind objects with lower indices. Thus, the object related terms in the joint are either not included (object is missing) or the same for each n.

3.2.2. Experiments

In this section we compare SuPAIR to the original AIR system (Eslami et al., 2016), and investigate the following two questions: **(Q1)** Do tractable appearance models lead to faster and more stable learning, i.e. with smaller variance? **(Q2)** Does an explicit background model make SuPAIR more robust to noise than AIR? To this end, we implemented SuPAIR in TensorFlow, making use of the RAT-SPN implementation by Peharz et al. (2019). We have also experimented with the SPN structure formulated by Poon & Domingos (2011) for the image domain, but have not found it to deliver significant improvements in learning speed or accuracy. We therefore report the results obtained using the more generally applicable random structures. All experiments were conducted using a single NVIDIA GeForce GTX 1080 Ti and a AMD Ryzen Threadripper 1950X CPU. Since the original code by Eslami et al. (2016) is not publically available, we made use of the well-documented AIR implementation in Pyro (Bingham et al., 2018) as our baseline, adopting the hyperparameter settings recommended by the authors.

Benchmark Datasets

We conducted experiments on two standard benchmarks for AIR, each with a different set of objects: *Multi-MNIST*, using MNIST-digits as objects, and *Sprites*, a dataset using artificially generated geometric shapes. In both datasets, each scene is a 50×50 grayscale image containing zero, one or two objects, with their positions and scale varied according to uniform distributions. Scenes with excessively overlapping objects are discarded. 20% of each dataset was retained as a test set to evaluate the count accuracy achieved by the inference network.

Hyperparameters and Inductive Bias

Unlike AIR, our model does not make the hard assumption that the background will always be black. It is therefore necessary to provide SuPAIR with some inductive bias expressing what ought and ought not to be considered background. As is common for unsupervised models, we specify this bias by means of hyperparameters.

Since we expect the background to be less visually complex than the objects, we make the background-SPN shallower and narrower, giving it less room to model dependencies. In turn, we set a lower limit for the variance of its Gaussian leaf nodes, allowing it to achieve higher likelihood scores on low variance data, such as black background. We have found this to be a surprisingly subtle yet effective way of guiding the model. While stronger biases could of course be specified, for instance by constraining the means of the Gaussian leaves or even by pretraining the SPNs on manually labelled data, we have found this not to be necessary to achieve good performance. The total number of learnable parameters in SuPAIR is given by Table 3.1 and is comparable to AIR, with the main difference being the lack of a baseline inference network, a consequence of our choice to enumerate N.

(Q1) Counting Objects

Fig. 3.5 depicts the inference results obtained at various stages of training, illustrating that our model learns to correctly count and locate the objects. A comparison of the count accuracies achieved over the course of training is provided in Fig. 3.6, highlighting the main advantage of our approach: training on MNIST is close to an order of magnitude faster when compared to the original AIR system. We have found that executing a

Table 3.2.: Count accuracies and ELBO values achieved after convergence. Note that the ELBO scores are not directly comparable between SuPAIR and AIR, due to the different formulations of the generative model.

	Count Accuracy		**ELBO**	
	SuPAIR	AIR	SuPAIR	AIR
Multi-MNIST	98.3%	94.7%	4923	615
Sprites	99.2%	95.9%	5022	685
Noisy-MNIST	93.8%	0.0%	1047	232
Grid-MNIST	97.5%	0.0%	3564	228

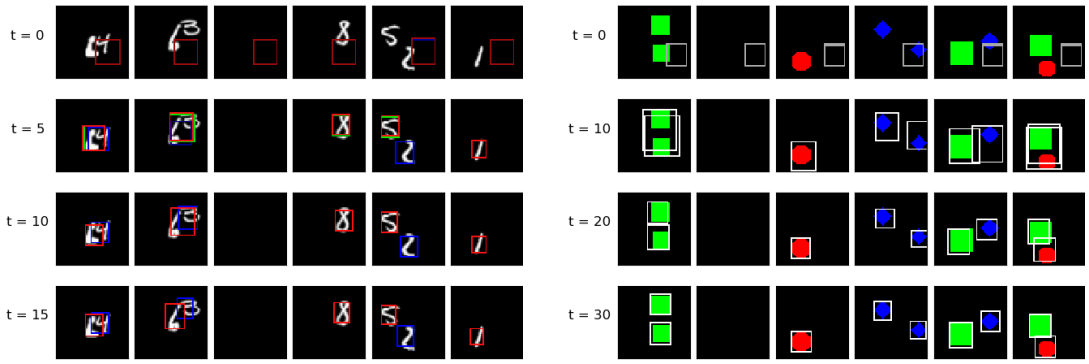

Figure 3.5.: Inference results of SuPAIR on *Multi-MNIST* (left) and *Sprites* (right) after t training epochs. The outlines of the up to three predicted object positions are displayed as bounding boxes. The model reliably learns to count and locate the objects in both datasets.

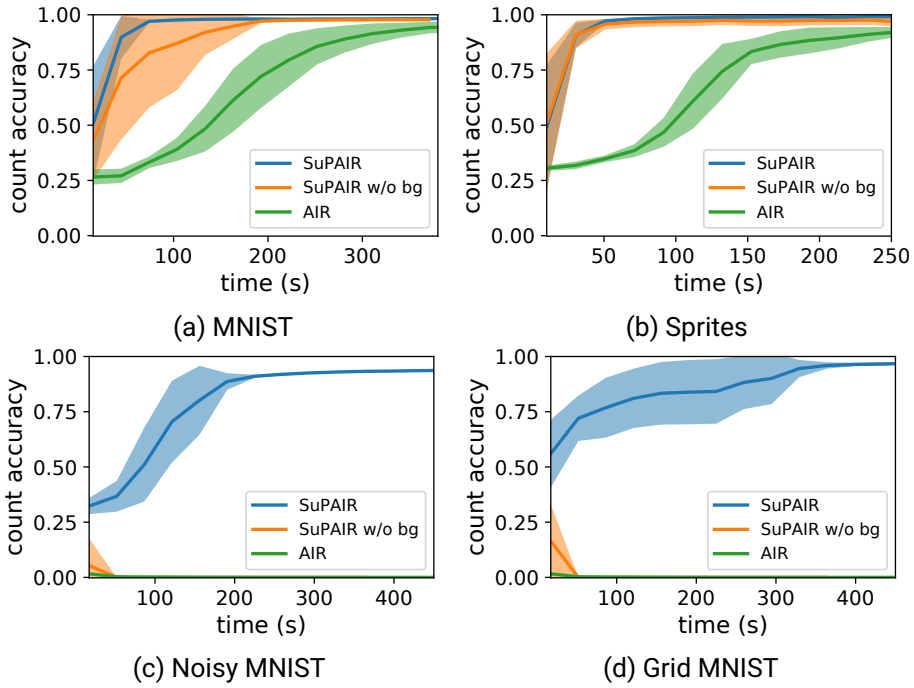

Figure 3.6.: Learning progress of SuPAIR and AIR on the four datasets. As an ablation test, we also report results for a variant of SuPAIR without a learned background model ("w/o bg"). On *Multi-MNIST* and *Sprites*, SuPAIR achieves a high count accuracy almost an order of magnitude faster than AIR.

single training epoch for SuPAIR is about 40% faster than for AIR, likely a consequence of the lack of baseline networks. This also suggests that most of the overall speedup can be attributed to faster statistical convergence as opposed to computational speed. The final accuracies and ELBO values achieved are given by Table 3.2. We note that the convergence speed we observed for AIR compares favorably with the numbers reported in the literature: Eslami et al. (2016) state they achieved convergence after two days of training on a NVIDIA Quadro K4000, while Bingham et al. (2018) report convergence after about 15 minutes on the much more powerful NVIDIA K80. On the sprite dataset, SuPAIR converged even faster, reaching a count accuracy of over 95% in less than a minute.

(Q2) Robustness to Noise

In order to evaluate the robustness of our model, we also trained it on two variants of the Multi-MNIST dataset, each featuring a different type of background, one resembling pure noise, the other structured background. For the first case, we simply add Gaussian noise to the entire scene. For the second, we generate a regular grid by coloring every fifth row and column of pixels gray, starting at a randomly selected offset. The MNIST digits are then overlaid on top.

Fig. 3.7 depicts the inference results of both SuPAIR and AIR after training on these datasets. Our model is still able to locate the digits, albeit with a slightly decreased count accuracy of about 90%. The results of the ablation test reported in Fig. 3.6 indicate that the background model is indeed crucial for this. AIR on the other hand fails in this setting: unsurprisingly, due to its lack of background model, it is forced to allocate at least one object to the entire image in order to attempt to reconstruct the background. More severe, however, is the fact that AIR's variational autoencoder fails to properly capture the distribution over objects. Its reconstructions only render a vague blur in the place of detected objects. This effect significantly degrades the training signal for the RNN, which consequently fails to accurately detect and locate the digits.

3.2.3. Conclusion

Structured probabilistic models such as AIR have achieved impressive results in various applications, mainly fueled by advances in approximate inference. We cannot, however, expect approximate inference to be a silver bullet, in particular for models of increasing complexity. Meanwhile, advances in probabilistic deep learning have shown that tractable models, such as sum-product networks (SPNs), can also be used to faithfully capture high-dimensional distributions. Consequently, building structured probabilistic models which combine the best of both worlds can be a fruitful avenue.

We presented a modification of AIR called SuPAIR, which learns to count and locate scene elements using SPNs as object appearance models. This allows SuPAIR to marginalize object and background models, yielding a well-calibrated scene likelihood. As a result, the SuPAIR inference network does not need to predict latent object codes, drastically reducing the variance of gradient estimates for the variational objective. As shown in our experiments, this property yields a dramatic reduction in training time, higher object detection accuracy, and improved noise robustness compared to the original AIR system.

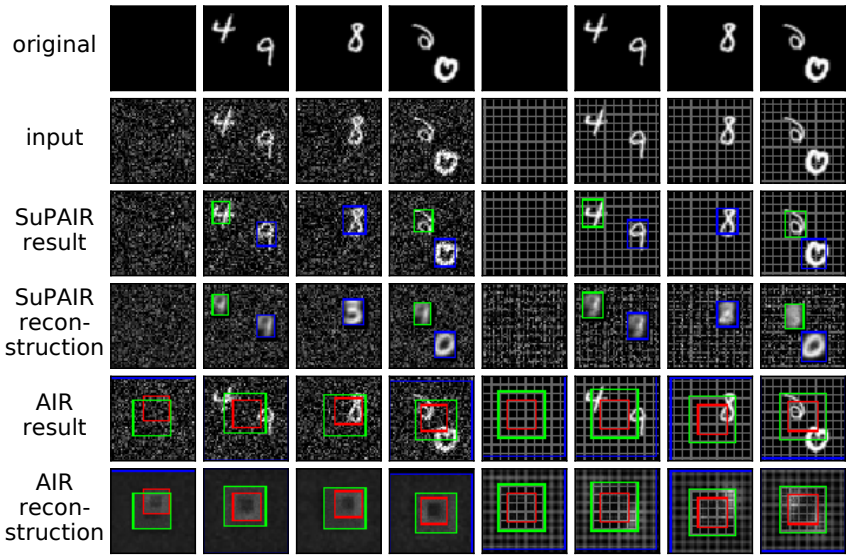

Figure 3.7.: Comparison of the inference results and reconstructions of SuPAIR and AIR on two variants of Multi-MNIST featuring either Gaussian noise (left) or a regular grid background drawn at a random offset (right). Reconstructions for SuPAIR are obtained according to the procedure proposed by Vergari et al. (2018). Note that AIR always predicts $N_{\text{max}} = 3$ objects.

3.3. Recent Improvements in Autoencoding-Based Scene Understanding

Several elements of AIR and SuPAIR have since been improved upon. To put SuPAIR into context, and to lay the groundwork for future discussions making use of these innovations, we highlight three aspects in which recent works differ from SuPAIR's architecture: utilizing attention instead of iterative inference, predicting pixelwise masks instead of bounding boxes, and combining them via a mixture model instead of deterministically.

3.3.1. Inferring and Processing Sets via Attention

A key weakness of SuPAIR and related systems lies in the way the encoder parameterizes q. Employing iterative inference with RNNs comes with a significant performance cost, and gives rise to deep and slow to optimize computational graphs. Even more critically, it necessarily introduces an order among objects in the scene, which recent analysis has identified as an important source of performance deficits (Zhang et al., 2019). Such a network is forced to learn a policy for assigning output slots to objects based on their appearance, position, etc. This necessarily leads to discontinuities when identical objects are moved to switch positions: No matter whether the learned policy orders objects top to bottom, foreground to background, or some other way, at some point there must be a discontinuous boundary at which they switch slots, which hurts performance and model stability.

Recent work has addressed this by employing *permutation equivariant* architectures, i.e. functions f with the property that any permutation π applied to their input set $\mathbf{x} = \{x_1, \ldots, x_n\}$ should permute the output, but not change it otherwise:

$$f\left(\left\{x_{\pi(1)}, \ldots, x_{\pi(n)}\right\}\right) = \left\{f_{\pi(1)}(\mathbf{x}), \ldots, f_{\pi(n)}(\mathbf{x})\right\}. \tag{3.10}$$

Conveniently (and perhaps not coincidentally), the currently most successful neural architecture for processing arbitrary pieces of information, namely *attention* (Vaswani et al., 2017), happens to possess this property. It has therefore become the standard for the permutation equivariant processing of sets (Lee et al., 2019; Zhang et al., 2019). In situations where taking input order into account is desired, such as language processing, it has to be specifically enabled by e.g. concatenating positional encodings $e(i)$ to the elements of the input set x_i, thereby encoding order in the input data as opposed to the

architecture. For a detailed description of (multi-head) attention, we refer the reader to these papers; here, we continue by denoting the operation as $\text{Att}(Q, K, V)$.

Utilizing sets as input to predict non-set values, such as global scene properties, can easily be achieved in an order-invariant way by combining set elements through a commutative operator, e.g. by summing them (Zaheer et al., 2017). Predicting sets from data of different shape is slightly more challenging, but can still be done. Most prominently, Slot Attention (Locatello et al., 2020) achieves this by initializing the output set z randomly, and then processing it equivariantly using attention, conditioned on the input features x. Specifically, updates are computed via $\text{Att}(z, x, x)$, except that attention weights are normalized over the slots, to ensure that slots compete over explaining parts of the input. The input x can be any set of vectors, for instance, the entries in the feature map output by a convolutional image encoder, enriched with positional encodings. Similar to a transformer architecture, this attention step is repeated multiple times and interspersed with additional learned nonlinear updates. For details, we refer the reader to Locatello et al. (2020).

The resulting architecture represents a general module for mapping arbitrary input to an unordered set of outputs of a given size. Due to its generality and suitability to the scene understanding problem, it has seen wide adoption in the field. Similar ideas have also been employed for supervised object detection (Carion et al., 2020).

3.3.2. Predicting Pixelwise Masks

A striking constraint of AIR and SuPAIR is that they frame objects using bounding boxes. This is a rather limiting way to parameterize object poses, and clearly breaks down in real-world images with objects of complicated shapes. In addition, it leads to sparse gradients with respect to the pose parameters, as the spatial transformer layer only yields gradients at the edges of the bounding box. Subsequent works, starting with MONet (Burgess et al., 2019) and IODINE (Greff et al., 2020a), have therefore opted for predicting pixelwise masks. Specifically, they employ a *spatial broadcast decoder* (Watters et al., 2019b) to decode latent vectors z_k into pixelwise masks $m_k \in \mathbb{R}^{h \times w}$. Architecturally, this is accomplished by broadcasting z_k to a map of size $h \times w$, adding positional encodings, and applying a convolutional network. Compared to a standard deconvolutional architecture, this makes it easier to render objects at specific coordinates, a task for which the position invariance imposed by convolutions can be detrimental.

A pixelwise segmentation may be computed by taking the SoftMax over the mask parameters m_k, yielding a categorical distribution over the slot S responsible for a given pixel:

$$P(S = k) = \frac{\exp(m_k)}{\sum_{i=1}^{n} \exp(m_i)}. \tag{3.11}$$

A downside of this approach is that there is no longer a hard architectural constraint forcing objects to be spatially contiguous. Only the convolutional nature of the decoder network yields a soft bias to this effect. A failure mode seen when applying models such as MONet to overly complex datasets is therefore segmenting the image by color, as opposed to by object semantics.

3.3.3. Object-Centric Distributions over Images via Pixelwise Mixtures

Finally, there is the question of how object images x_k should be combined based on this segmentation distribution. Neither AIR's interaction model of summing images, nor SuPAIR's approach of superimposing them are appropriate for an unordered, colored set of object images. A natural choice adopted by MONet and subsequent works is that of a pixelwise mixture over object images. Having access to a pixelwise categorical distribution $P(S = k)$ over slots, they decode each pixel as

$$p(x \mid z) = \sum_k p(S = k \mid z) p(x_k = x \mid z). \tag{3.12}$$

This provides a principled way to combine object images which respects the probabilistic nature of the model as well as order invariance. The resulting likelihood may be used for training, and MAP estimates may be utilized at test time.

3.4. Limitations of Scene Understanding via Autoencoding

Despite substantial work on unsupervised scene understanding models based on autoencoding 2D images, the effective segmentation results seen on synthetic datasets such as Multi-MNIST or CLEVR have mostly not generalized to real-world scenes. The model which showed the most success in this regard is likely GENESIS-V2 (Engelcke et al., 2021), which utilizes a stochastic stick-breaking process to segment simple real-world scenes.

An intuitive explanation for the difficulty of scene understanding in general real-world images is that they contain too many visual confounders which prevent an unsupervised model trained by autoencoding pixels from learning about objects. Lighting conditions, shadows, occlusion, etc., all have drastic impact on the 2D pixels observed by such a model. They are thereby obfuscating the true latents we would most like to infer, namely 3D geometry and semantic segmentations.

One option to address these issues is to add handmade inductive biases to steer the model in the right direction, overriding the effects of the confounders. Examples of this approach can be seen in a variety of works (Mao et al., 2019; Deng et al., 2020b; Lin et al., 2020; Jiang et al., 2020a). These methods introduce biases such as scene graphs, dedicated background slots, and strong priors on object size, to address certain more complex, but still synthetic datasets, such as ATARI games. However, there are several downsides to this approach: Most importantly, handmade biases that work on one such dataset are unlikely to generalize to other data, especially real-world images, which makes the resulting models quite brittle. In addition, it appears difficult to scale this approach to large, real-world data, where more and stronger biases would be needed. In summary, the issue with this approach is that the quality of the unsupervised representations output by such a model is roughly proportional to the strength of the biases which the modeler puts in.

An alterative approach is to seek out different self-supervised objective functions that are capable of scaling to more complicated data by allowing the model to avoid visual confounders through learning and scaling, as opposed to handmade biases. There are several potential avenues to achieve this. One is to devise more sophisticated learning schemes for unsupervised learning on images, such as masked autoencoders (He et al., 2022; Shi et al., 2022), or the ones discussed in Section 2.5. Another is the approach we will follow here: Move on to richer modalities, which allow disentangling the semantic latents from their confounders. In particular, we focus on 2D videos, and 3D multiview data. In both settings, we can leverage a simple observation: Throughout time, and between different viewpoints, the shape and texture of rigid objects will remain the same. In addition, their pose will only change smoothly through time. This introduces additional redundancies, and makes it possible to cut through the confounders discussed above.

3.5. The Role of Structured Models

The above observations may be seen as an instance of Sutton's *bitter lesson* (Sutton, 2019): While imbuing models with hand-designed structural biases frequently leads to short-term improvements, in the long term, these will be outscaled by more general systems which discover the relevant concepts organically through learning. Examples of this trend can already be seen in the developments described in Section 3.3. Having noticed this pattern, it is easy to dismiss research utilizing hand-designed structural constraints as a waste of time. If the models we eventually use in practice have so few hand-designed features, should we just focus all research on them, and forget about the alternatives?

While this is a tempting conclusion, it is important to note that the successful development of large-scale models often depends on results from prior smaller-scale research involving structural biases. Structured models tell us which concepts and biases are powerful enough to unlock qualitatively new model capabilities. This then informs follow-up work on large-scale, general models to choose architectures which facilitate learning about these concepts organically. For example, the research discussed in this chapter has shown that a set-structured latent space in an autoencoder can naturally lead to knowledge being represented in a semantically segmented way. While follow-up work might dispense with implementation details such as confining objects to bounding boxes, it is likely to retain set-structured latents to facilitate this general effect.

We therefore take the view that structured models are first and foremost a tool to analyze model behavior. If a modest number of structured biases are enough to elicit semantically meaningful responses from the model, such as object segmentations, it is likely that a model with fewer or relaxed biases will also feature representations of similar quality, even if its semantic meaning cannot be immediately read out. In such a case, it is frequently still possible to predict e.g. object segmentations by training a decoder on a small number of supervised examples (see, e.g., Sajjadi et al. (2022c)).

For these reasons, the relationship between research on structured and large-scale models can be characterized as synergistic, with the former using limited data and computational resources to identify promising biases and objectives, and the latter leveraging an abundance of resources to leverage them at scale. Further examples of this pattern will appear throughout this thesis.

4. Strengthening and Utilizing Representations through Dynamics Prediction

In the previous chapter, we have seen the ability of autoencoders to discover segmentation of scenes in an unsupervised way, and have also discussed the limitations of these methods. We have identified the setting of autoencoding 2D images as fundamentally limiting, as it does not allow access to many of the clues humans use for scene understanding in the real world.

In this chapter, we explore a natural step forward, namely, the compositional modeling of videos. This not only opens the door to more complex tasks, such as reinforcement learning, but the additional redundancy in such data can also simplify and regularize the object detection problem (Kosiorek et al., 2018). To this end, the notion of temporal consistency may be leveraged as an additional inductive bias, guiding the model to desirable behavior. In situations where interactions between objects are prevalent, understanding and explicitly modeling these interactions in an object-centric state-space is valuable for obtaining good predictive models.

4.1. Modeling Videos as Deep State Space Models

We now outline the general approach for representation learning and scene understanding for video data. Building upon our work for the single image domain, we continue taking a Bayesian approach, and first specify a probabilistic model to represent the data generation process. We denote the individual images in the input video as random variables $\mathbf{x} = (x_0, \dots, x_T)$. Similar to before, we postulate (potentially structured) latent variables $\mathbf{z} = (z_0, \dots, z_T)$ as a low-dimensional representation of the state of the

Figure 4.1.: A schematic depiction of a state-space model. For discrete latents, this is known as a hidden Markov model.

environment. The process of rendering such a state into visual observations $p(x_t|z_t)$ may be modeled as before. However, we would now also like to model the evolution of the system over time, i.e. , the conditional probability $p(z_{t+1}|z_t)$. This gives rise to a state-space model, as depicted in Figure 4.1. The structure of this model expresses the Markov assumption: The entire state of the system at time t should be captured by z_t, such that any future observations x_{t+p} will be independent of past observations x_{t-q} given z_t, for any $p, q > 0$.

As before, while it is easy enough to specify such a model, the challenge lies in learning its parameters, and using it for inference. Variational inference once more provides a conceptual framework to address these challenges: If we can construct a variational distribution $q(\mathbf{z}|\mathbf{x})$ to approximate the true posterior $p(\mathbf{z}|\mathbf{x})$, we can then utilize this for both inference, and to learn the model parameters by optimizing the ELBO. However, constructing such an encoder is somewhat more complicated than before, as both $\mathbf{z}$ and $\mathbf{x}$ are sequences. Processing entire videos at once (e.g. , by concatenating all frames) will quickly run into computational limits as the sequence length increases. At the same time, processing all frames independently by factorizing $q(\mathbf{z}|\mathbf{x}) = \prod_{t=1}^{T} q(z_t|x_t)$ means that we cannot take into account dependencies between frames during inference. This squanders one of the main upsides of video data, namely that the notion of temporal consistency may be used to improve scene understanding.

Compromises between these two extremes can be found by taking inspiration from the literature on *filtering* (Särkkä, 2013). It is concerned with estimating the most recent state of a system $p(z_t|x_{1:T})$. We may observe that the joint distribution factorizes as

follows:

$$p(z_t, x_{1:t}) = \sum_{z_{t-1}} p(z_{t-1}, z_t, x_{1:t}) \tag{4.1}$$

$$= \sum_{z_{t-1}} p(z_{t-1}, x_{1:t-1}) p(z_t | z_{t-1}, x_{1:t-1}) p(x_t | z_t, z_{t-1}, x_{1:t-1}) \tag{4.2}$$

$$= \sum_{z_{t-1}} p(z_{t-1}, x_{1:t-1}) p(z_t | z_{t-1}) p(x_t | z_t) \tag{4.3}$$

Since the latter two terms $p(z_t|z_{t-1})p(x_t|z_t)$ are known from the model definition, this equation suggests a recursive method to compute the desired distribution: Starting from $t = 1$, we can calculate $p(z_t, x_{1:t})$ from $p(z_{t-1}, x_{1:t-1})$ by marginalizing only a single variable (z_{t-1}) at a time. For many classical state-space models, this is tractable. For instance, a linear Gaussian model yields the Kalman filter. A downside of this approach is that the influence of future observations x_t on earlier states z_{t-k} is missed. Methods which do take those into account are called *smoothing*. However, in many applications, we are most interested in keeping a running estimate of the current state of the environment, as observations are made in real time. In such a setting, filtering is sufficient.

How may we now apply this idea in the context of deep variational models? Here, even marginalizing over a single z_{t-1} is intractable, since it is multi-dimensional, and $p(z_t|z_{t-1})$ is modelled by a neural network. However, the basic idea of iteratively estimating $p(z_t|x_{1:t})$ from $p(z_{t-1}|x_{1:t-1})$ is still valid, as it stems from the Markov assumption: All dependencies between past and future observations need to flow through z_t. A natural approach is therefore to factorize the variational distribution similarly, i.e.

$$q(z_{1:t}|x_{1:t}) = q(z_1|x_1) \prod_{t=2}^{T} q(z_t|x_t, z_{t-1}). \tag{4.4}$$

When the neural network representing $q(z_t|x_t, z_{t-1})$ is powerful enough, it can learn to approximate the intractable marginalization in Eq. (4.1).

Many parameterizations of this variational filtering idea are possible and have been proposed. Prominent examples include STORN (Bayer & Osendorfer, 2014), VRNN (Chung et al., 2015), DKF (Krishnan et al., 2015), and PlaNet (Hafner et al., 2019). These works differ in subtle ways regarding the exact conditioning of the variational network: Some process the observations deterministically using an RNN and condition $q(z_t|\cdot)$ on its hidden state h_t, others more closely follow the formulation given in

Eq. (4.4). For an overview over the various conditioning schemes, we refer the reader to Kossen (2020) and Bayer et al. (2021).

Other works have eschewed the filtering approximation in favor of full smoothing posteriors (Krishnan et al., 2017; Babaeizadeh et al., 2018; Gregor et al., 2019). As noted by Bayer et al. (2021), such an approach is most beneficial in highly stochastic environments in which both $p(z_t|z_{t-1})$ and $p(x_t|z_t)$ are multimodal distributions. In the context of visually observed physical interactions, however, both are largely deterministic up to a small amount of noise, and a cheaper filtering approach is typically sufficient.

4.2. Modeling Multi-Object Dynamics

In our introduction to representation learning (Section 2.2), we have argued that we should pragmatically aim for representations which are most useful for the relevant prediction tasks. In the context of dynamics modeling, one of the most important tasks is clearly to predict the evolution of the system over time. Being able to do so is of critical importance for learning an accurate generative model, forecasting future trajectories, and planning in interactive environments. What makes a good representation z_t of the system's state is therefore largely determined by our ability to learn dynamics models $p(z_t|z_{t-1})$.

It is therefore instructive to consider the large body of literature investigating dynamics modeling in a vacuum, i.e. , without the visual element. In this setting, we have access to sequences of ground truth states $z_{1:T}^{1:n}$ for n objects, and would like to learn a prediction model. What type of neural network architecture should be best suited to learn such a function for everyday physics? While this depends on the exact type of system being modelled to some degree, we can notice some commonalities present in many everyday physical interactions. Consider for example elastic collisions, friction, or gravity, as possible interactions between objects. All of these have in common that their effects may be computed by considering pairwise interactions between objects. Additionally, they are all based on a general rule applying to all pairs of objects (e.g. Newton's law of universal gravitation), with the specifics depending on the current state of the objects (e.g. mass, position, etc.).

Similar to what we saw in Section 3.3.1, these properties again motivate using permutation equivariant architectures to process the set of object states. These architectures typically feature a component $I(z^o, z^{o'})$ computing the interaction between a pair of

objects $z^o, z^{o'}$. To compute the total influence on an object, these are then aggregated in a permutation-invariant way, e.g. by summing them. The future state may then be computed using e.g. an MLP f:

$$z_t^o = f\left(z_{t-1}^o, \sum_{o' \in [1,n] \setminus \{o\}} I(z_{t-1}^o, z_{t-1}^{o'})\right). \tag{4.5}$$

This type of architecture closely matches the intuition regrading physical laws outlined above: $I(\cdot,\cdot)$ may model physical laws to determine the forces acting on an object, and f computes the effect these forces have on the state z^o.

The most widely used architecture of roughly this structure is self-attention, where the interaction function is the dot-product, the aggregation method a weighted, normalized sum of features, and the post-processing function f is typically a shallow MLP. This yields a generically powerful predictor that is likely to be effective when sufficient data is available. However, more specialized solutions to physics modeling exist, as well.

Many such models originate from the *graph neural network* (GNN) community (Kipf & Welling, 2017; Zhou et al., 2020). These are networks designed to process graphs, i.e. sets of nodes, with edges connecting them. This may be done similarly to the architecture shown above (Eq. (4.5)), except that we do not compute interactions over all pairs of objects/nodes, but only over those connected by an edge. Influential papers taking this view of physics modeling include (Battaglia et al., 2016; Sanchez-Gonzalez et al., 2018). Formulating an explicit graph structure allows the modeller to impose additional structure on the model to determine which objects should and should not influence each other, which can regularize learning and save computational effort. The graph structure may also be determined dynamically to e.g. only allow interactions between sufficiently close objects. That said, the usual caveats regarding structured models apply (Section 3.5) - especially in end-to-end visual learning tasks, it can be more convenient to simply assume a fully connected structure, and let the interaction model learn which edges are relevant. This question of structure notwithstanding, GNNs have been used successfully in a wide variety of physical modeling tasks, including rigid body physics, deformable objects, fluid dynamics, and particle physics (Li et al., 2019; Sanchez-Gonzalez et al., 2020; Shlomi et al., 2020). Many of these works have compared their methods to unstructured prediction models such as MLPs, and found that segmented, permutation equivariant methods are much more effective (Battaglia et al., 2016; Sanchez-Gonzalez et al., 2018). This is a major motivation for pursuing unsupervised object-centric representation learning: If we can learn an object-centric

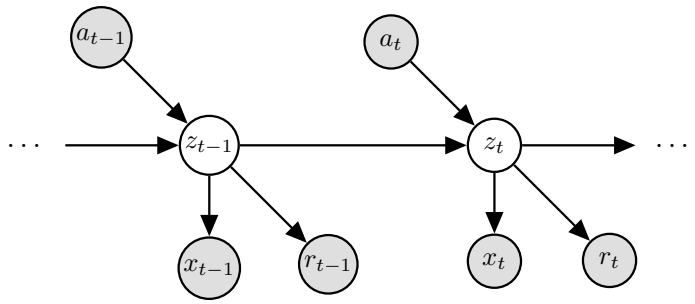

Figure 4.2.: A schematic depiction of a world model. For discrete systems, this is known as a partially observable Markov decision process (POMDP).

representation, this will enable us to build better dynamics models, and therefore better planners for real-world interactions.

4.3. Leveraging Dynamics Models for Reinforcement Learning

Aside from being able to leverage the additional redundancy from video data, the main motivation for modeling dynamics is unlocking the ability to plan in interactive settings. To model this interactivity, we condition the dynamics model $p(z_t|z_{t-1}, a_t)$ on actions a_t, and add a reward distribution $p(r_t|z_t)$ (see Figure 4.2). In classical AI, the resulting structure is studied in the context of discrete systems and referred to as a partially observable Markov decision process (POMDP), whereas the current literature often simply calls it a world model (Ha & Schmidhuber, 2018). The task is now to find a policy $\pi : z \to a$ selecting actions which maximize the cumulative reward $R = \sum_{t=0}^{T} \delta^t r_t$ over a timeframe T with a discount factor $\delta \in (0, 1)$. This is the domain typically considered by (deep) reinforcement learning. Approaches broadly fall into two categories: *model-free* and *model-based* (Sutton & Barto, 2018).

4.3.1. Model-free RL

Model-free RL methods learn policies purely based on a set of sampled experiences (z_{t-1}, z_t, a_t, r_t). Using such a dataset, one may learn a policy $\pi : z_t \to a_t$ directly using *policy gradient* methods (Agarwal et al., 2021) such as proximal policy optimization

(PPO) (Schulman et al., 2017). Alternatively, one may learn a function $Q(z_t, a_t)$ estimating the maximum future score R achievable from a given (state, action) pair assuming optimal future actions. This may then be used in turn to formulate a policy $\pi(z) = \arg\max_a Q(z, a)$ selecting the action with the highest estimated Q-value. Both types of methods have been originally formulated for simple tabular representations of either the policy or the Q-function. However, they have been adapted to more complex settings by instead approximately representing these functions using deep neural networks, yielding the representational power necessary for high-dimensional state spaces. As the distribution of experiences shifts with the policy used, new samples are collected at regular intervals during training using the current policy.

Additional difficulties are introduced by the fact that the visual environments we are interested in are *partially observable*: Contrary to what is assumed by the formulations above, we do not have access to the true state z_t of the environment, but only to visual observations x_t. A frequently used approximate solution to this is to record experiences of the form $(x_{t-k}, \ldots, x_t, a_t, r_t)$, for some $k > 0$, instead. Learned policy or Q-functions may then be conditioned on a small window of prior observations $x_{t-k:t}$ in lieu of the exact state. This is adequate when the true state can be estimated from $x_{t-k:t}$, which is the case for many environments that do not actually contain significant hidden information, and x_t is merely a higher dimensional, more redundant encoding of z_t. The most frequent exception to this property, the fact that object velocities are typically not apparent from a single frame, is addressed by using a small number of frames $k > 0$. Information hidden over long timeframes however may not be captured in this manner. Additionally, it places a significant burden on the RL model, as it not only needs to learn to evaluate the state, but first needs to infer a suitable representation for said state.

Despite these challenges, model-free RL systems have proven very effective at learning highly capable policies in complex environments when supplied with sufficient compute and data. They have been successfully applied to many synthetic environments including ATARI games (Mnih et al., 2015), multi-agent settings featuring tool use (Baker et al., 2019), and modern competitive video games (Vinyals et al., 2019; Berner et al., 2019), though the latter examples did not use visual observations and instead allowed the model direct access to the observable state of the environment in structured form. Real-world use of these methods is hindered by their enormous hunger for data samples: Even solving the simple ATARI games in this manner requires millions of data samples, and for the more sophisticated environments, that number reaches into the billions. Had the games used for training OpenAI Five (Berner et al., 2019) been played sequentially in real time instead of simulated in parallel, they would have taken over 10000 years. The cost of collecting such an enormous amount of data is aggravated by the fact that

many of the most successful methods, such as PPO, are *on-policy* methods. This means that they need to be trained on data collected *using the current policy* π, and samples generally cannot be used multiple times during training, or shared between different runs (Ibarz et al., 2021). Finally, even after this cost has been paid, the result is a model which is entirely task-specific: It can not straight-forwardly be used for other objectives in the same environment. While this is acceptable for the zero-sum games frequently considered as testbeds, it is insufficient for the real world, where we would like to learn a wide variety of skills.

Real-world robotics applications of model-free RL are therefore bottlenecked by the cost of collecting the necessary data, making especially vision based learning difficult (Ibarz et al., 2021). Even using fleets of robots, the number of obervations that can be reasonably obtained is orders of magnitude lower compared to the synthetic examples discussed above (Dasari et al., 2020; Kalashnikov et al., 2021). Many researchers therefore opt to train in simulated environments, and then attempt to transfer the learned model to the real world (Zhao et al., 2020). This so called *sim-to-real* approach is limited to environments which can be effectively simulated, and runs the risk of degraded performance when the simulator is imperfect, as even small inaccuracies of the simulator can be exploited by the reinforcement learner.

4.3.2. Model-based RL

A natural angle to alleviate these issues is to leverage learned models of the environment, such as the ones we have discussed in Section 4.1. The most important benefit such a model provides (also over simulators as discussed above) is the prediction of future trajectories $p(z_{t+1}, r_{t+1}|a_t, x_{1:t})$ from visual observations $x_{1:t}$, which may be used for planning: Even if we have learned no task-specific policy at all, we can construct one by simply asking our model about the consequences of various action sequences, and pick the most beneficial one. Classical algorithms for search and planning such as breadth-first search or Monte-Carlo Tree Search (MCTS) (Coulom, 2007) may be used for this purpose.

Alternatively, the model may be used to generate samples to train a policy, as an alternative to a manually programmed simulator as discussed above. In this case, the training loop consists of three phases: collecting samples in the real world, using those to train the model, and employing the model to improve the policy. This sort of arrangement is frequently referred to as a *Dyna* loop, after an early system of this type (Sutton, 1991). Unlike a fixed simulator, such a loop has the capacity to correct model

inaccuracies: If the policy learns to exploit a quirk of the model that does not translate to the real world, it will subsequently collect data demonstrating that the real world behaves differently, which will in turn be used to correct the model.

Of course, this argument breaks down when the world model is lacking the capacity to accurately model the dynamics of the environment. The challenges inherent to video modeling therefore indirectly also affect model-based RL, which has delayed its adoption compared to model-free alternatives. Recently however, significant progress has been made, which we will discuss in Section 4.5.3.

Inferring efficient representations in the sense introduced in Section 2.2 has several advantages in the context of RL. First, it increases the fidelity of the world model: As we have seen in Section 4.2, segmented representations allow more effective prediction of physical trajectories than unstructured ones, or even worse, raw images. Second, it reduces the cost of planning, as the dynamics model only needs to process a relatively small state description as opposed to high-resolution images. Finally, it simplifies policy learning, as a more compact state encoding will also make it easier to predict actions and rewards.

4.4. STOVE: Jointly Learning about Objects and Their Dynamics

We now present the STOVE model, a structured obect-aware video prediction model which combines the ideas described above. Prior works in this area, such as SQAIR (Kosiorek et al., 2018), DDPAE (Hsieh et al., 2018), R-NEM (van Steenkiste et al., 2018), and COBRA (Watters et al., 2019a) have explored these concepts, but have not demonstrated realistic long-term video predictions on par with supervised approaches to modeling physics. STOVE has been published in Kossen et al. (2020), and the text in this section and the corresponding Appendix A is an edited version of that publication.

With STOVE, we combine image and physics modeling into a single state-space model, which explicitly reasons about object positions and velocities. It is trained end-to-end on pure video data in a self-supervised fashion and learns to detect objects, to model their interactions, and to predict future states and observations. To facilitate learning via variational inference in this model, we provide a novel inference architecture, which reuses the learned generative physics model in the variational distribution. As we will demonstrate, our model generates realistic rollouts over hundreds of time steps,

outperforms other video modeling approaches, and approaches the performance of the supervised baseline which has access to the ground truth object states.

Moving beyond unsupervised learning, we also demonstrate how STOVE can be employed for model-based reinforcement learning (RL). By conditioning state predictions on actions and adding reward predictions to our dynamics predictor, we allow our model to be used for search or planning. Our empirical evidence shows that an actor based on Monte-Carlo tree search (MCTS) (Coulom, 2007) on top of our model is competitive to model-free approaches such as Proximal Policy Optimization (PPO) (Schulman et al., 2017), while only requiring a fraction of the samples.

We proceed by introducing the two main components of STOVE: a structured image model and a dynamics model. We show how to perform joint inference and training, as well as how to extend the model to the RL setting. We then present our experimental evaluation, before touching on further related work and concluding.

4.4.1. Structured Object-Aware Video Modeling

We approach the task of modeling a video with frames $x_0, \ldots, x_T$, from a probabilistic perspective, assuming a sequence of Markovian latent states $z_0, \ldots, z_T$, which decompose into the properties of a fixed number O of objects, i.e. $z_t = (z_t^1, \ldots, z_t^O)$. In the spirit of compositionality, we propose to specify and train such a model by explicitly combining a dynamics prediction model $p(z_{t+1} \mid z_t)$ and a scene model $p(x_t \mid z_t)$. This yields a state-space model, which can be trained on pure video data, using variational inference and an approximate posterior distribution $q(\mathbf{z} \mid \mathbf{x})$. Our model differs from previous work that also follows this methodology, most notably SQAIR and DDPAE, in three major ways:

- We propose a more compact architecture for the variational distribution $q(\mathbf{z} \mid \mathbf{x})$, which reuses the dynamics model $p(z_{t+1} \mid z_t)$, and avoids the costly double recurrence across time and objects which was present in previous work.
- We parameterize the dynamics model using a graph neural network, taking advantage of the decomposed nature of the latent state z_t.
- Instead of treating each z_t^o as an arbitrary latent code, we explicitly reserve the first six entries of this vector for the object's position, size, and velocity, each in x, y direction, and use this information for the dynamics prediction task. We write $z_t^o = (z_{t,\text{pos}}^o, z_{t,\text{size}}^o, z_{t,\text{velo}}^o, z_{t,\text{latent}}^o)$.

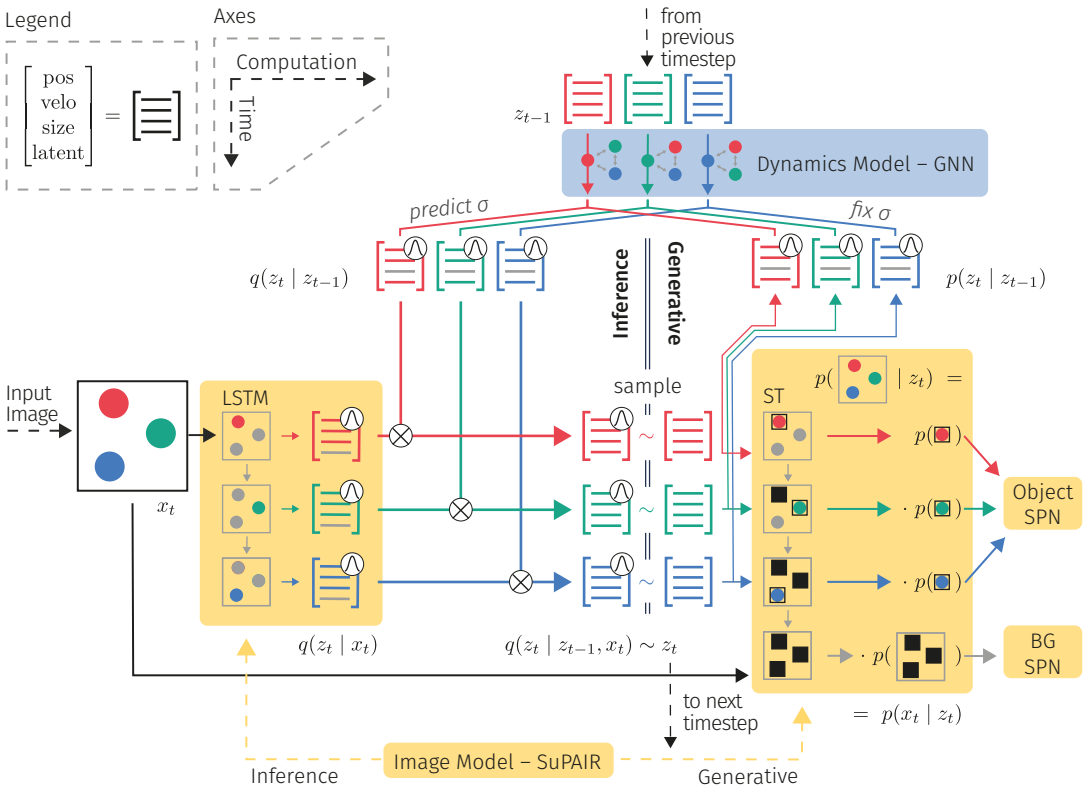

Figure 4.3.: Overview of STOVE's architecture. (Center left) At time t, the input image x_t is processed by an LSTM in order to obtain a proposal distribution over object states $q(z_t \mid x_t)$. (Top) A separate proposal $q(z_t \mid z_{t-1})$ is obtained by propagating the previous state z_{t-1} using the dynamics model. (Center) The multiplication of both proposal distributions yields the final variational distribution $q(z_t \mid z_{t-1}, x_t)$. (Right) We sample z_t from this distribution to evaluate the generative distribution $p(z_t \mid z_{t-1})p(x_t \mid z_t)$, where $p(z_t \mid z_{t-1})$ shares means – but not variances – with $q(z_t \mid z_{t-1})$, and $p(x_t \mid z_t)$ can be obtained by direct evaluation of x_t in the sum-product networks. Not shown is the dependence on x_{t-1} in the inference routine which allows for the inference of velocities.

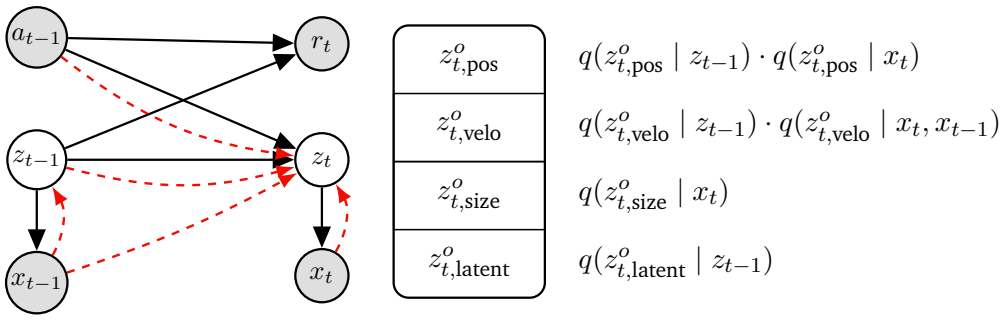

Figure 4.4.: (Left) Depiction of the graphical model underlying STOVE. Black arrows denote the generative mechanism and red arrows the inference procedure. The variational distribution $q(z_t \mid z_{t-1}, x_t, x_{t-1})$ is formed by combining predictions from the dynamics model $p(z_t \mid z_{t-1})$ and the object detection network $q(z_t \mid x_t)$. For the RL domain, our approach is extended by action conditioning and reward prediction. (Right) Components of z^o_t and corresponding variational distributions. Note that the velocities are estimated based on the change in positions between timesteps, inducing a dependency on x_{t-1}.

We begin by briefly introducing the individual components before discussing how they are combined to form our state-space model. Fig. 4.3 visualises the computational flow of STOVE's inference and generative routines, Fig. 4.4 (left) specifies the underlying graphical model.

As our image model $p(x_t \mid z_t) = p(x_t \mid z_{t_{\text{pos}}}, z_{t_{\text{size}}})$, we directly apply SuPAIR as introduced in Section 3.2.1, except that we fix the number of objects, and substitute $z^o_{t,\text{where}} = (z^o_{t,\text{pos}}, z^o_{t,\text{size}})$. Similarly, we use an inference network as in SuPAIR to model $q(z_{t_{\text{where}}} \mid x_t)$. In order to successfully capture complex dynamics, the state transition distribution $p(z_{t+1} \mid z_t) = p(z^1_{t+1}, \dots, z^O_{t+1} \mid z^1_t, \dots, z^O_t)$ needs to be parameterized using a flexible, non-linear estimator. As argued in Section 4.2, graph neural networks are an excellent choice for such a task. Following the GNN approach, we build a dynamics model of the basic form

$$\hat{z}^o_{t+1,\text{pos}}, \hat{z}^o_{t+1,\text{velo}}, \hat{z}^o_{t+1,\text{latent}} = f\left(g(z^o_t) + \sum_{o' \neq o} \alpha(z^o_t, z^{o'}_t) h(z^o_t, z^{o'}_t)\right), \tag{4.6}$$

where f, g, h, α represent functions parameterized by dense neural networks. α is an attention mechanism which allows the network to focus on specific object pairs. Finally,

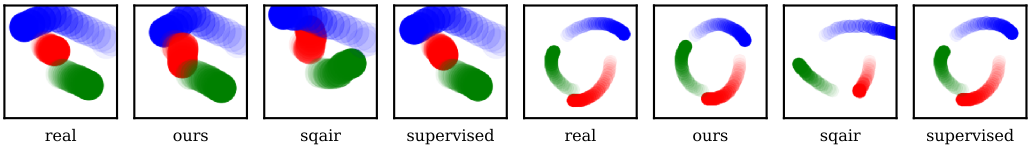

Figure 4.5.: Visualisation of object positions from the real environment and predictions made by our model, SQAIR, and the supervised baseline, for the billiards and gravity environment after the first 8 frames were given. Our model achieves realistic behaviour, outperforms the unsupervised baselines, and approaches the quality of the supervised baseline, despite being fully unsupervised. For full effect, the reader is encouraged to watch animated versions of the sequences in repository github.com/jlko/STOVE.

we assume a constant prior over the object sizes, i. e., $\hat{z}^o_{t+1,\text{size}} = z^o_{t,\text{size}}$. The full state transition distribution is then given by the Gaussian $p(z^o_{t+1} \mid z^o_t) = \mathcal{N}(\hat{z}^o_{t+1}, \sigma)$, using a fixed σ.

4.4.2. Joint State-Space Model

Next, we assemble a single state-space model from the two separate, compositional models for image modeling and physics prediction. The interface between the two component models are the latent positions and velocities. The scene model infers them from images and the physics model propagates them forward in time. Combining the two yields the state-space model $p(\mathbf{x}, \mathbf{z}) = p(z_0)p(x_0 \mid z_0) \prod_t p(z_t \mid z_{t-1})p(x_t \mid z_t)$. To initialize the state, we model $p(z_0, z_1)$ using simple uniform and Gaussian distributions. Details are given in Appendix A.3.3.

Our model is trained on given video sequences $\mathbf{x}$ by maximizing the evidence lower bound (ELBO) $\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z} \mid \mathbf{x})]$. This requires formulating a variational distribution $q(\mathbf{z} \mid \mathbf{x})$ to approximate the true posterior $p(\mathbf{z} \mid \mathbf{x})$. As discussed in Section 4.1, a natural approach is to factorize this distribution over time, i.e. $q(\mathbf{z} \mid \mathbf{x}) = q(z_0 \mid x_0) \prod_t q(z_t \mid z_{t-1}, x_t)$, resembling a Bayesian filter. The distribution $q(z_0 \mid x_0)$ is then readily available using the inference network provided by SuPAIR.

The formulation of $q(z_t \mid z_{t-1}, x_t)$, however, is an important design decision. Previous work, including SQAIR and DDPAE, have chosen to unroll this distribution over objects, introducing a costly double recurrence over time and objects, requiring $T \cdot O$ sequential

recurrence steps in total. This increases the variance of the gradient estimate, slows down training, and hampers scalability. Inspired by Becker-Ehmck et al. (2019), we avoid this cost by *reusing* the dynamics model for the variational distribution. First, we construct the variational distribution $q(z^o_{t,\text{pos}} \mid z^o_{t-1})$ by slightly adjusting the dynamics prediction $p(z^o_{t,\text{pos}} \mid z^o_{t-1})$, using the same mean values but separately predicted standard deviations. Together with an estimate for the *same* object by the object detection network $q(z^o_{t,\text{pos}} \mid x_t)$, we construct a joint estimate by multiplying the two Gaussians and renormalizing, yielding another Gaussian:

$$q(z^o_{t,\text{pos}} \mid z_{t-1}, x_t) \propto q(z^o_{t,\text{pos}} \mid z_{t-1}) \cdot q(z^o_{t,\text{pos}} \mid x_t). \tag{4.7}$$

Intuitively, this results in a distribution which reconciles the two proposals. A double recurrence is avoided since $q(z_t \mid x_t)$ does not depend on previous timesteps and may thus be computed in parallel for all frames. Similarly, $q(z_t \mid z_{t-1})$ may be computed in parallel for all objects, leading to only $T + O$ sequential recurrence steps total. An additional benefit of this approach is that the information learned by the dynamics network is reused for inference — if $q(z_t \mid x_t, z_{t-1})$ were just another neural network, it would have to essentially relearn the environment's dynamics from scratch, resulting in a waste of parameters and training time. A further consequence is that the image likelihood $p(x_t \mid z_t)$ is backpropagated through the dynamics model, which has been shown to be beneficial for efficient training (Karl et al., 2017; Becker-Ehmck et al., 2019). The same procedure is applied to reconcile velocity estimates from the two networks, where for the image model, velocities $z^o_{t,\text{velo}}$ are estimated from position differences between two consecutive timesteps. The object scales $z^o_{t,\text{scale}}$ are inferred solely from the image model. The latent states $z^o_{t,\text{latent}}$ increase the modeling capacity of the dynamics network, are initialised to zero-mean Gaussians, and do not interact with the image model. This then gives the inference procedure for the full latent state $z^o_t = (z^o_{t,\text{pos}}, z^o_{t,\text{size}}, z^o_{t,\text{velo}}, z^o_{t,\text{latent}})$, as illustrated in Fig. 4.4 (right).

Despite its benefits, this technique has thus far only been used in environments with a single object or with known state information. A challenge when applying it in a multi-object video setting is to match up the proposals of the two networks. Since the object detection RNN outputs proposals for object locations in an indeterminate order, it is not immediately clear how to find the corresponding proposals from the dynamics network. We have, however, found that a simple matching procedure results in good performance: For each z_t, we assign the object order that results in the minimal difference of $||z_{t,\text{pos}} - z_{t-1,\text{pos}}||$, where $||\cdot||$ is the Euclidean norm. The resulting Euclidean bipartite matching problem can be solved in cubic time using the classic Hungarian algorithm (Kuhn, 1955).

4.4.3. Conditioning on Actions

To extend STOVE to a reinforcement learning setting, we make two changes, yielding a distribution $p(z_t, r_t \mid z_{t-1}, a_{t-1})$.

First, we condition the dynamics model on actions a_t, enabling a conditional prediction based on both state and action. To keep the model invariant to the order of the input objects, the action information is concatenated to each object state z^o_{t-1} before they are fed into the dynamics model. The model has to learn on its own which of the objects in the scene are influenced by the actions. To facilitate this, we have found it helpful to also concatenate appearance information from the extracted object patches to the object state. While this patch-wise code could, in general, be obtained using some neural feature extractor, we achieved satisfactory performance by simply using the mean values per color channel when given colored input.

The second change to the model is the addition of reward prediction. In many RL environments, rewards depend on the interactions between objects. Therefore, the dynamics prediction architecture, presented in Eq. 4.6, is well suited to also predict rewards. We choose to share the same encoding of object interactions between reward and dynamics prediction and simply apply two different output networks (f in Eq. 4.6) to obtain the dynamics and reward predictions. The total model is again optimized using the ELBO, this time including the reward likelihood $p(r_t \mid z_{t-1}, a_{t-1})$.

4.4.4. Experimental Evidence

In order to evaluate our model, we compare it to baselines in three different settings: First, pure video prediction, where the goal is to predict future frames of a video given previous ones. Second, the prediction of future object positions, which may be relevant for downstream tasks. Third, we extend one of the video datasets to a reinforcement learning task and investigate how our physics model may be utilized for sample-efficient, model-based reinforcement learning. Our PyTorch implementation of STOVE has been released publicly.[1]

[1]The code can be found in the GitHub repository github.com/jlko/STOVE. It also contains animated versions of the videos predicted by our model and the baselines.

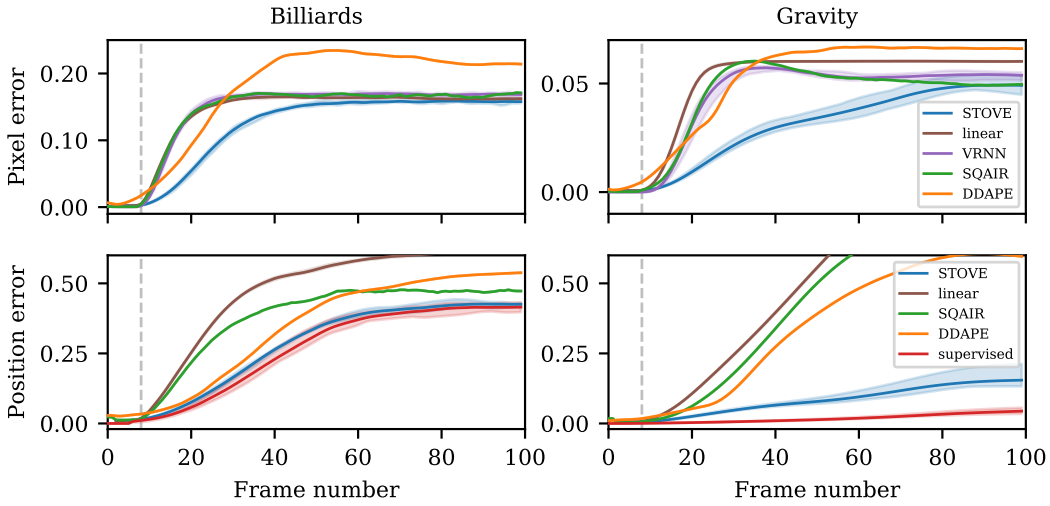

Figure 4.6.: Mean test set performance of our model compared to baselines. Our approach (STOVE) clearly outperforms all unsupervised baselines and is almost indistinguishable from the supervised dynamics model on the billiards task. (Top) Mean squared errors over all pixels in the video prediction setting (the lower, the better). (Bottom) Mean Euclidean distances between predicted and true positions (the lower, the better). All position and pixel values are in $[0, 1]$. In all experiments, the first eight frames are given, all remaining frames are then conditionally generated. The shading indicates the max and min values over multiple training runs with identical hyperparameters.

Video and State Modeling

Inspired by Watters et al. (2017), we consider grayscale videos of objects moving according to physical laws. In particular, we opt for the commonly used bouncing billiards balls dataset, as well as a dataset of gravitationally interacting balls. For further details on the datasets, see Appendix A.4. When trained using a single GTX 1080 Ti, STOVE converges after about 20 hours. As baselines, we compare to VRNNs (Chung et al., 2015), SQAIR (Kosiorek et al., 2018), and DDPAE (Hsieh et al., 2018). To allow for a fair comparison, we fix the number of objects predicted by SQAIR and DDPAE to the correct amount. Furthermore, we compare to a supervised baseline: Here, we consider the ground truth positions and velocities to be fully observed, and train our

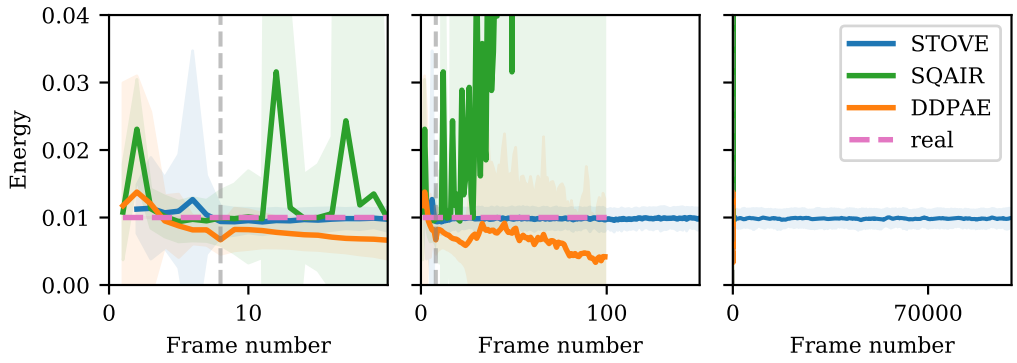

Figure 4.7.: Comparison of the kinetic energies of the rollouts predicted by the models, computed based on position differences between successive states. Only STOVE's predictions reflect the conservation of total kinetic energy in the billiards data set. This is a quantitive measure of the convincing physical behavior in the rollout videos. (Left, center) Averages are over 300 trajectories from the test set. Shaded regions indicate one standard deviation. STOVE correctly predicts trajectories with constant energy, whereas SQAIR and DDPAE quickly diverge. (Right) Rolling average over a single, extremely long-term run. We conjecture that STOVE predicts physical behavior indefinitely.

dynamics model on them, resembling the setting of Battaglia et al. (2016). Since our model needs to infer object states from pixels, this baseline provides an upper bound on the predictive performance we can hope to achieve with our model. In turn, the size of the performance gap between the two is a good indicator of the quality of our state-space model. We also report the results obtained by combining our image model with a simple linear physics model, which linearly extrapolates the objects' trajectories. Since VRNN does not reason about object positions, we only evaluate it on the video prediction task. Similarly, the supervised baseline does not reason about images and is considered for the position prediction task only. For more information on the baselines, see Appendix A.5.

Fig. 4.6 depicts the reconstruction and prediction errors of the various models: Each model is given eight frames of video from the test set as input, which it then reconstructs. Conditioned on this input, the models predict the object positions or resulting video frames for the following 92 timesteps. The predictions are evaluated on ground truth data by computing the mean squared error between pixels and the Euclidean distance

Table 4.1.: Predictive performance of our approach, the baselines, and ablations (lower is better, best unsupervised values are in bold). STOVE outperforms all unsupervised baselines and is almost indistinguishable from the supervised model on the billiards task. The values are computed by summing the prediction errors presented in Fig. 4.6 in the time interval $t \in [9, 18]$, i.e., the first ten predicted timesteps. Standard deviations across multiple training runs are given where available.

	Billiards (pixels)	Billiards (positions)	Gravity (pixels)	Gravity (positions)
STOVE (ours)	$\mathbf{0.240 \pm 0.014}$	$\mathbf{0.418 \pm 0.020}$	$\mathbf{0.040 \pm 0.003}$	$\mathbf{0.142 \pm 0.007}$
VRNN	0.526 ± 0.014	–	0.055 ± 0.012	–
SQAIR	0.591	0.804	0.070	0.194
DDPAE	0.405	0.482	0.120	0.298
Linear	0.844 ± 0.005	1.348 ± 0.015	0.196 ± 0.002	0.493 ± 0.004
Supervised	–	0.232 ± 0.037	–	0.013 ± 0.002
Abl: Double Dynamics	0.262	0.458	0.042	0.154
Abl: No Velocity	0.272	0.460	0.053	0.174
Abl: No Latent	0.338	0.050	0.089	0.235

between positions based on the best available object matching. We outperform all baselines on both the state and the image prediction task by a large margin. Additionally, we perform strikingly close to the supervised model.

For the gravitational data, the prediction task appears easier, as all models achieve lower errors than on the billiards task. However, in this regime of easy prediction, precise access to the object states becomes more important, which is likely the reason why the gap between our approach and the supervised baseline is slightly more pronounced. Despite this, STOVE produces high-quality rollouts and outperforms the unsupervised baselines.

Table 4.1 underlines these results with concrete numbers. We also report results for three ablations of STOVE, which are obtained by (a) training a separate dynamics network for inference with the same graph neural network architecture, instead of sharing weights with the generative model as argued for in section 4.4.2, (b) no longer explicitly modeling velocities z_{velo} in the state, and (c) removing the latent state variables z_{latent}. The ablation study shows that each of these components contributes positively to the performance of STOVE. See Appendix A.6 for a comparison of training curves for the ablations.

Fig. 4.5 illustrates predictions on future object positions made by the models, after

each of them was given eight consecutive frames from the datasets. Visually, we find that STOVE predicts physically plausible sequences over long timeframes. This desirable property is not captured by the rollout error: Due to the chaotic nature of our environments, infinitesimally close initial states diverge quickly and a model which perfectly follows the ground truth states cannot exist. After this divergence has occurred, the rollout error no longer provides any information on the quality of the learned physical behavior. We therefore turn to investigating the total kinetic energy of the predicted billiards trajectories. Since the collisions in the training set are fully elastic and frictional forces are not present, the initial energy should be conserved. Fig. 4.7 shows the kinetic energies of trajectories predicted by STOVE and its baselines, computed based on the position differences between consecutive timesteps. While the energies of SQAIR and DDPAE diverge quickly in less than 100 frames, the mean energies of STOVE's rollouts stay constant and are good estimates of the true energy. We have confirmed that STOVE predicts constant energies – and therefore displays realistic looking behavior – for at least $100\,000$ steps. This is in stark contrast to the baselines, which predict teleporting, stopping, or overlapping objects after less than 100 frames. In the billiards dataset used by us and the literature, the total energy is the same for all sequences in the training set. See Appendix A.2 for a discussion of how STOVE handles diverse energies.

Model-Based Control

To explore the usefulness of STOVE for reinforcement learning, we extend the billiards dataset into a reinforcement learning task. Now, the agent controls one of the balls using nine actions, which correspond to moving in one of the eight (inter)cardinal directions and staying at rest. The goal is to avoid collisions with the other balls, which elastically bounce off of each other, the walls, and the controlled ball. A negative reward of -1 is given whenever the controlled ball collides with one of the others. To allow the models to recognize the object controlled by the agents we now provide it with RGB input in which the balls are colored differently. Starting with a random policy, we iteratively gather observations from the environment, i. e. sequences of images, actions, and rewards. Using these, we train our model as described in Sec. 4.4.3. To obtain a policy based on our world model, we use Monte-Carlo tree search (MCTS), leveraging our model as a simulator for planning. Using this policy, we gather more observations and apply them to refine the world model. As an upper bound on the performance achievable in this manner, we report the results obtained by MCTS when the real environment is used for planning. As a model-free baseline, we consider PPO (Schulman et al., 2017),

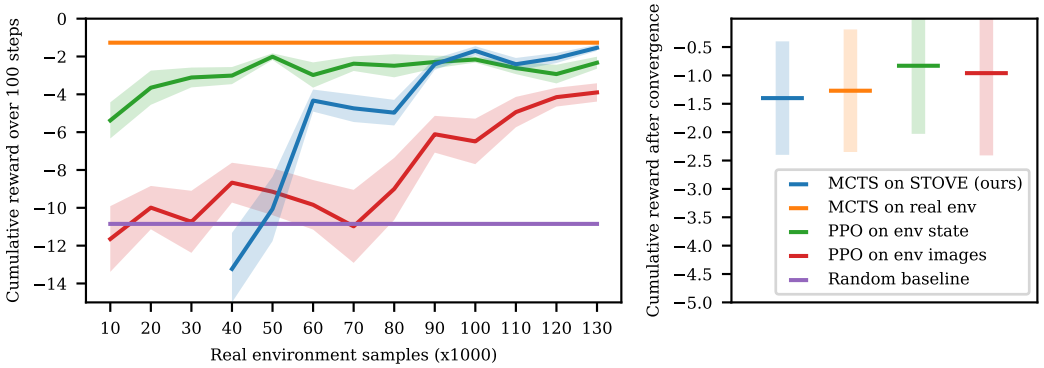

Figure 4.8.: Comparison of all models on sample efficiency and final performance. (Left) Mean cumulative reward over 100 steps on the environment, averaged over 100 environments, using the specified policy. The shaded regions correspond to one-tenth of a standard deviation. In addition to the training curves, two constant baselines are shown, one representing a random policy and one corresponding to the MCTS-based policy when using the real environment as a simulator. (Right) Final performance of all approaches, after training each model to convergence. The shaded region corresponds to one standard deviation.

which is a state-of-the-art algorithm on comparable domains such as Atari games. To explore the effect of the availability of state information, we also run PPO on a version of the environment in which, instead of images, the ground-truth object positions and velocities are observed directly.

Learning curves for each of the agents are given in Fig. 4.8 (left), reported at intervals of $10\,000$ samples taken from the environment, up to a total of $130\,000$. For our model, we collect the first $50\,000$ samples using a random policy to provide an initial training set. After that, the described training loop is used, iterating between collecting $10\,000$ observations using an MCTS-based policy and refining the model using examples sampled from the pool of previously seen observations. After $130\,000$ samples, PPO has not yet seen enough samples to converge, whereas our model quickly learns to meaningfully model the environment and thus produces a better policy at this stage. Even when PPO is trained on ground truth states, MCTS-based on STOVE remains comparable.

After training each model to convergence, the final performance of all approaches is reported in Fig. 4.8 (right). In this case, PPO achieves slightly better results, however it only converges after training for approximately $4\,000\,000$ steps, while our approach only uses $130\,000$ samples. After around $1\,500\,000$ steps, PPO does eventually surpass the performance of STOVE-based MCTS. Additionally, we find that MCTS on STOVE yields almost the same performance as on the real environment, indicating that it can be used to anticipate and avoid collisions accurately.

4.4.5. Related Work

Multiple lines of work with the goal of video modeling or prediction been proposed prior to STOVE. Prominently, supervised modeling of physical interactions from videos has been investigated by Fragkiadaki et al. (2015), who train a model to play billiards with a single ball. Similarly, graph neural networks have been trained in a supervised fashion to predict the dynamics of objects from images (Watters et al., 2017; Sanchez-Gonzalez et al., 2018; Sun et al., 2018; 2019) or ground truth states (Kipf et al., 2018; Wang et al., 2018b; Chang et al., 2017). Janner et al. (2019) show successful planning based on learned interactions, but assume access to image segmentations. Several unsupervised approaches address the problem by fitting the parameters of a physics engine to data (Jaques et al., 2019; Wu et al., 2016; 2015). This necessitates specifying in advance which physical laws govern the observed interactions. In the fully unsupervised setting, mainly unstructured variational approaches have been explored (Babaeizadeh et al., 2018; Chung et al., 2015; Krishnan et al., 2015). However, without the explicit notion of objects, their performance in scenarios with interacting objects remains limited. Nevertheless, unstructured video models have recently been applied to model-based RL and have been shown to improve sample efficiency when used as a simulator for the real environment (Oh et al., 2015; Kaiser et al., 2020).

Only a small number of prior works incorporate objects into unsupervised video models. Xu et al. (2019) and Ehrhardt et al. (2018) take non-probabilistic autoencoding approaches to discovering objects in real-world videos. COBRA (Watters et al., 2019a) represents a model-based RL approach based on MONet, but is restricted to environments with non-interacting objects and only uses one-step search to build its policy. Closest to STOVE were a small number of probabilistic models, namely SQAIR (Kosiorek et al., 2018), R-NEM (van Steenkiste et al., 2018; Greff et al., 2017), and DDPAE (Hsieh et al., 2018). R-NEM learns a mixture model via expectation-maximization unrolled through time and handles interactions between objects in a factorized fashion. However,

it lacks an explicitly structured latent space, and requires noise in the input data to avoid local minima. Both DDPAE and SQAIR extend the AIR approach to work on videos using standard recurrent architectures. As discussed, this introduces a double recurrence over objects and time, which is detrimental for performance. However, SQAIR is capable of handling a varying number of objects, which is not something we considered with STOVE.

4.4.6. Conclusion

We introduced STOVE, a structured, object-aware model for unsupervised video modeling and planning. It combines advances in unsupervised image modeling and physics prediction into a single compositional state-space model. The resulting joint model explicitly reasons about object positions and velocities, and is capable of generating highly accurate video predictions in domains featuring complicated non-linear interactions between objects. As our experimental evaluation shows, it outperforms previous unsupervised approaches and even approaches the performance and visual quality of a supervised model.

Additionally, we presented an extension of the video learning framework to the RL setting. Our experiments demonstrate that our model may be utilized for sample-efficient model-based control in a visual domain, making headway towards a long-standing goal of the model-based RL community. In particular, STOVE yields good performance with more than one order of magnitude fewer samples compared to the model-free baseline, even when paired with a relatively simple planning algorithm like MCTS.

At the same time, STOVE also makes several assumptions for the sake of simplicity. First, we assume a fixed number of objects, which may be avoided by performing dynamic object propagation and discovery like in SQAIR. Second, we have inherited the assumption of rectangular object masks from AIR. Applying a more flexible model such as MONet (Burgess et al., 2019) or GENESIS (Engelcke et al., 2020) may alleviate this, but also poses additional challenges, especially regarding the explicit modeling of movement. Finally, the availability of high-quality learned state-space models enables the use of more sophisticated planning algorithms in visual domains (Chua et al., 2018). In particular, by combining planning with policy and value networks, model-free and model-based RL may be integrated into a comprehensive system (Buckman et al., 2018).

4.5. Recent Developments in Structured Video Modeling

Since the publication of STOVE, a number of developments have further progressed the fields of unsupervised dynamics modeling and model-based RL. Here we give a short overview over these trends.

4.5.1. Attention-Based Prediction

Mirroring what we discussed in Section 3.3.1 for single images, attention has also become the dominant inference mechanism for video modeling. Its ability to incorporate arbitrary pieces of information as inputs is especially useful when the task is to process a sequence of object-centric state-descriptions $\{z_t^o \mid t \in [0, T], o \in [0, n]\}$, such as the ones inferred by the methods discussed in Chapter 3. Ding et al. (2021) have shown that by simply applying self-attention to this set of inputs allows addressing a variety of complex inference tasks, such as visual question answering and object tracking. A key idea here is to add a temporal encoding $f(t)$ to each z_t^o. This allows the model to take into account the temporal order of the input states, while maintaining order invariance with respect to the objects in each frame. This setup also allows the model to implicitly learn to match objects between frames, eliminating the need for Hungarian matching as in STOVE.

4.5.2. Richer Object-based Video Models

A number of works have progressed unsupervised object-based video models towards richer datasets. ViMON (Weis et al., 2021) and OP3 (Veerapaneni et al., 2020) extend the pixelwise masking models MONet and IODINE, which we discussed in Section 3.3.2 to video data. SCALOR (Jiang et al., 2020a) builds upon the ideas of SPAIR (Crawford & Pineau, 2019) to build a spatial transformer model for videos capable of scaling to dozens of objects. However, as noted by Weis et al. (2021), segmentation performance of these models still largely relies on color cues, with models struggling to separate identically colored objects, while also failing on more realistic images involving textures or lighting effects. This suggests that the issues discussed in Section 3.4 remain present for these models, albeit perhaps to a lesser extent.

A way to circumvent these problems and bring object-centric learning to real-world imagery has been demonstrated by Kipf et al. (2022): They show that by first preprocessing images by predicting optical flow, as well as conditioning on object masks for the first frame, one can sufficiently reduce the visual complexity of real-world videos to make them amenable for unsupervised object-based models. Since optical flow predicts the motion of the surface visible at a given pixel, it tends to be largely uniform for solid objects, but disparate for objects following different trajectories. As a result, the ground truth optical flow of a realistic scene, can, when displayed as an RGB image, be almost as simple as the synthetic 2D scenes we have considered thus far. While ground truth optical flow is typically unavailable, there exist prediction models to effectively approximate it, such as Stone et al. (2021). A downside of this approach is that it fails to distinguish static objects, which will not differ in their optical flow. Follow-up work (Elsayed et al., 2022) has therefore explored adding depth (i.e. the distance between the camera and the surface visible at a given pixel) as a supervision target. We further discuss the idea of leveraging 3D information for scene understanding in Chapter 5.

4.5.3. Learning Complex Policies Based on Models

Significant progress has also been made in model-based RL systems, by integrating dynamics models into Dyna-like RL loops. Such agents have been successfully trained on ATARI games (Hafner et al., 2020; 2021), robots (Finn et al., 2016; Wu et al., 2022), and complex video games (Hafner et al., 2023). A key development throughout these works is the improved ability of the world model to learn efficient representations of the environment's state. Early works such as Finn et al. (2016) have simply attempted to learn a video model $p(x_t|x_{t-1}, a_t)$ predicting future frames. In contrast, more recent works such as Hafner et al. (2023) explicitly learn state-space models as introduced in Section 4.1, making it possible to obtain compact representations. Dynamics and reward prediction can then take place without having to process images.

However, we note that the world models employed by these works remain rather basic, often employing a simple vector processed by a recurrent neural network as z_t. We posit that they might benefit from some of the recently developed components discussed above, such as set-structured latents, vision transformers for inference, and attention-based dynamics prediction.

5. 3D Scene Understanding

In the previous chapters, we have discussed how we may construct models for unsupervised scene understanding, and leverage them to act in the surrounding world. However, our experiments have been constrained to synthetic 2D environments, whereas real-world images have remained out of reach. As we have argued in Section 3.4, this limitation stems from the presence of a multitude of visual confounders found in natural images. Consider for instance a tabletop scene with diverse three-dimensional objects on it, such as those depicted in Figure 5.1. We can imagine a concise description of such a scene, specifying object poses, shapes, and texture. However, in the presence of shadows, occlusions, and other visual confounders, such a representation will seem no more plausible to an autoencoding model than one clustering pixels by color.

So how do we as humans arrive at the former so naturally? While the full answer to this question lies outside the realm of computer science, in this chapter, we investigate a particular advantage that humans have over the models we discussed so far: By moving around, we can experience the world around us from multiple viewpoints, giving us insights into the three-dimensional nature of our surroundings. To narrow this gap, we will now consider datasets providing multiple images of each scene. As before in this thesis, we would like to learn the capability of understanding 3D space without relying on human annotations providing supervision as to the true 3D geometry. Multi-view data provides a convenient way to do so: While it is not as abundant on the internet as individual images, it can be collected cheaply at large scale using standard cameras. To help us get started in this direction, we will also assume that we know the *pose* of the camera taking each image, meaning its position and viewing direction. While this makes data gathering somewhat more expensive, it makes it much easier to formulate learning objectives. In Section 5.5.2, we will discuss potential ways to also leverage unposed imagery.

Before we get into methods and representations for 3D learning, let us first discuss which properties we would like our 3D vision system to have. As discussed before, our

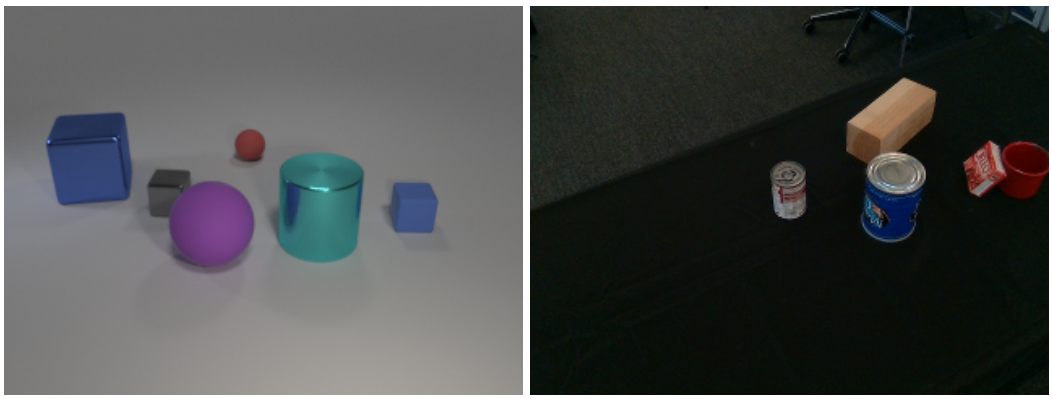

Figure 5.1.: Example images from the synthetic CLEVR (left) (Johnson et al., 2017) and real-world DexYCB (right) (Chao et al., 2021) datasets, two benchmarks which we will utilize in this chapter.

overall goal is to enable robust, real-time interaction with the real world. This implies a variety of design objectives:

1. In order to enable real-time interaction, we need to be able to infer a representation of 3D geometry quickly, without time-consuming offline computations.

2. To enable embodied interaction with the real world, we would ideally like to be able to to do so using only a single input view. Note that humans are capable of this as well: While we do have two eyes, our ability to judge 3D structure is only minorly impeded when one of them is closed, or when we observe a 2D image. The role of multi-view data is therefore to facilitate *learning* about 3D, whereas at test time, we would like to be able to operate on single images.

3. As much as possible, training should be based on data which can be collected at scale, without human intervention. This excludes manual annotations of 3D geometry, and motivates working on multi-view data.

4. In this manner, we would like to infer representations which are not only effective at specifying 3D structure, but are also easy to process for downstream tasks, in a similar way as discussed in previous chapters. This means they should be low-dimensional and segmented by objects.

These requirements give us a clear understanding of what we need to construct in this chapter: An inference network $f : x \to z$, mapping an input image x to a representation

z which is not only an effective scene description as discussed in Chapter 3, but also encodes the 3D geometry of the scene. But how do we represent 3D geometry and make sure that the model learns about it without direct 3D supervision? We will discuss this after a short introduction to the camera model underlying 3D graphics and vision.

5.1. 3D Camera Model

In order to discuss the effects of observing a scene from different viewpoints, we need to briefly formalize the notion of a camera. In this thesis, we will work with the *pinhole camera* model, a simple abstraction of a real-world camera which eliminates the effects of lense distortion and depth of field (i.e., objects at all depths will be in focus), and generally is a good match for our everyday experience with images (Foley et al., 2013). We start with the notion that a *camera sensor* is a grid of points called pixels, which are arranged on a plane, and measure incoming light. In the pinhole camera model, we assume that all light reaching the camera sensor travels through an infinitesimally small hole, which we define as the position of the camera $\mathbf{x}$. This ensures that each point on the camera sensor (i.e., each pixel) will only receive light coming from one direction, namely the direction of the pinhole. As a result, we may associate each pixel with a ray $\mathbf{r}$ indicating the viewing direction of that pixel. We will sometimes write $\mathbf{r}(t) = \mathbf{x} + t\mathbf{r}$ to refer to the point reached when travelling along the ray for a certain distance t. A camera is fully specified by the position of its pinhole, and the rays associated with its pixels. An example of this is illustrated by Figure 5.2.

How can we determine these rays? Typically, one starts by placing the camera in some canonical position, e.g. with the pinhole at the origin, and the center of vision following the negative z-axis. Knowing the dimensions of the camera sensor and its distance from the pinhole (the *focal length*) is enough to place it in this coordinate system, and compute the rays. Placing the camera at a given pose in 3D space then simply amounts to rotating and translating the array of rays as desired, i.e., a linear transform. Of course, in graphics, one is frequently interested in the opposite transformation, namely determining where a given point in 3D space will appear in an image. Thankfully, this inverse process may similarly be expressed via linear projections. For a detailed account of this process, we refer the reader to Hartley & Zisserman (2004).

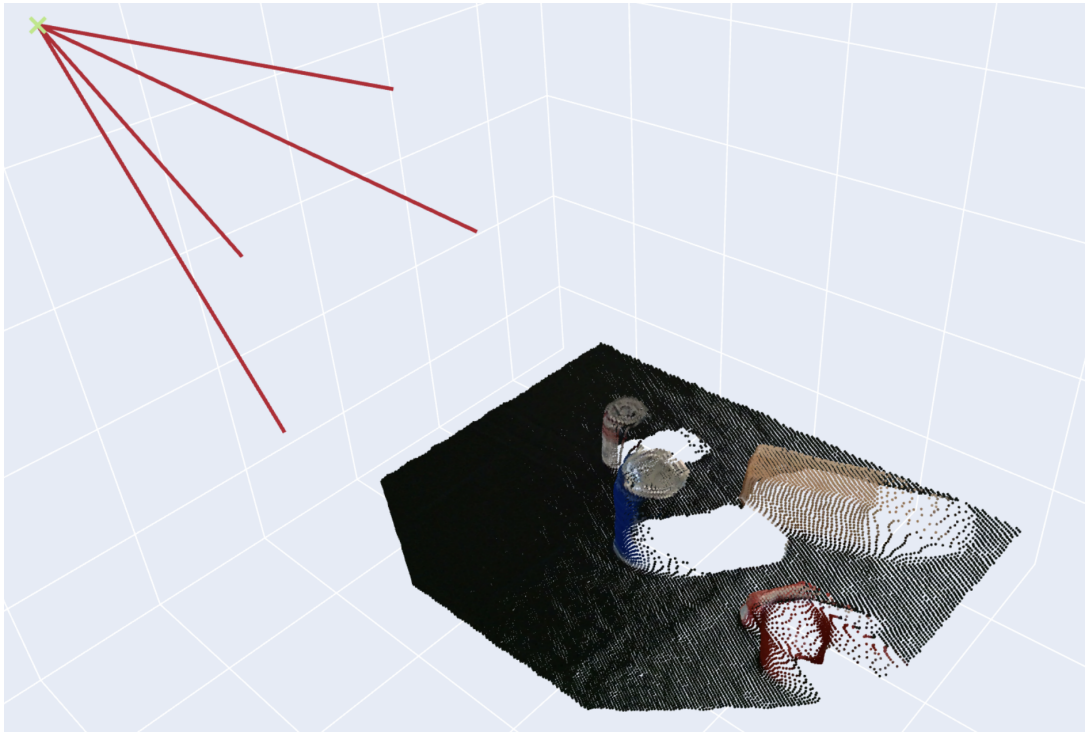

Figure 5.2.: Illustration of the 3D pointcloud obtained by combining the DexYCB input image from Figure 5.1 (right) with the associated camera parameters and depth map. The position of the pinhole camera is marked in green, and its orientation is indicated by the rays associated with the corner pixels (red), illustrating the resulting pyramid of vision. The pointcloud is illustrated in the colors of the associated pixels, and constrained to those points which lie on the tabletop surface. Note the gaps on the back sides of objects resulting from occlusion and the somewhat noisy depth estimates at the edges of objects.

5.2. Approaches for Learning about 3D

No matter how 3D geometry is represented within the model, if we wish to train it, we will need to formulate a loss measuring how well the inferred geometry matches reality. Here, we introduce the three basic methods of doing so: human annotations, 3D sensors, and novel view synthesis.

Human Annotation The most straightforward way to get started with 3D prediction is to manually annotate scenes with 3D information. For instance, one may construct 3D meshes of real-world objects using modeling software, and then annotate scenes with the identity of the objects present, their poses, and the background. This is the approach generally taken in the literature on robot grasping. As a result, many of the most popular benchmark datasets are very small scale, featuring only a few dozen objects, and no more than a thousand scenes (Xiang et al., 2018; Fang et al., 2020; Chao et al., 2021). This enables the training of supervised models regressing on ground truth poses. Many of these systems also leverage known object meshes at test time to refine predictions by e.g. rendering the predicted scene and comparing it to the input image (Li et al., 2018; Zakharov et al., 2019).

At the same time, the small scale of these benchmarks largely precludes success by unsupervised approaches: The ratio between visual complexity and training data redundancy is generally too large to observe the effects we are interested in in this thesis. To nevertheless demonstrate the efficacy of learning methods, many works therefore focus on synthetic 3D datasets such as ShapeNet, a large dataset of 3D models (Chang et al., 2015).

Depth Supervision An alternative source of 3D supervision is depth information, i.e., the distance between the camera and the observed surface for each pixel in an image. Unlike ground-truth meshes, such data may be obtained at scale in the real world with reasonable accuracy using low-cost camera systems (Horaud et al., 2016). For indoor scenes, the average error is typically not larger than a few centimeters, although reflections and glare can cause issues. Alternatively, depth may be estimated from stereo vision or motion, see e.g. (Chang & Chen, 2018; Teed & Deng, 2020).

The most straightforward way to use depth information is to train depth predictors such as Zhao et al. (2022), i.e. models which predict depth information for one or more given RGB images. While these models do not by themselves produce sufficient

scene understanding for downstream tasks, the depth information they provide may be leveraged in a variety of ways: One option is to utilize depth as a training signal to enforce 3D understanding, as shown by Elsayed et al. (2022). We will later show how to use depth information to speed up training of a volume based scene representation. Another usage is to compute point clouds of a scene's surface, as was done for Figure 5.2. We will discuss the usage of point clouds as a representation of a scene's geometry in Section 5.3.2. One new powerful way to leverage depth has recently been demonstrated by Fridman et al. (2023): Using depth information, they simulate the effects of small camera movements, and use the resulting image pairs to inject 3D understanding into a large scale 2D image model.

Novel View Synthesis Finally, 3D information may be gleaned from datasets featuring multiple images for each scene, ideally including camera poses. Methods for doing so have been developed in the field of photogrammetry for over 100 years, starting with manual measurements taken on analog photographs (Albertz, 2007). Recently, software packages such as COLMAP (Schönberger et al., 2016) have enabled robust 3D reconstuctions from high-resolution digital images. While these systems are useful in many contexts, they do not quite fulfill the requirements we outlined at the start of the chapter, since they require significant computation time before yielding results on a scene, always require multiple images, and yield large 3D point clouds as opposed to compact scene representations.

This motivates considering multi-view data in a machine learning setting. The most straightforward way to do this is to regress an arbitrarily chosen view given one or more others, a task known as *novel view synthesis*. The advantage of this approach is that it requires no data beyond posed RGB images, and makes no assumptions regarding the way the model produces the predicted image, as long as it does so in a differentiable manner. Starting with (Eslami et al., 2018), a number of works have demonstrated the usefulness of 3D representations obtained in this manner, and we will discuss many of them in this chapter.

5.3. Representations of 3D Geometry

We now discuss approaches for representing 3D geometry as data to be processed by machine learning models. For each of them, we highlight their pros and cons for

representation learning, particularly their efficiency, the effectiveness of available neural processing components, and the ability to render images from them.

5.3.1. Voxels

One of the simplest representations of 3D geometry are *voxels*. A portmanteau of volume and pixel, the term voxel refers to the idea of segmenting 3D space into regular cubes, and storing occupancy information for each of them. Manipulation of such a structure using neural components is relatively straightforward: We may for instance apply 3D convolutions to a 3D tensor of binary occupancy indicators. If we also have access to ground truth voxels, which is the case for synthetic datasets such as ShapeNet, this yields a relatively straightforward training paradigm, which was used in early 3D reconstruction works (Choy et al., 2016; Wu et al., 2017). It is however also possible to render a given voxel shape in a differentiable way, by projecting voxels into image space. This enables learning 3D inference without direct supervision, as for instance demonstrated by Yan et al. (2016). That work does not consider color or texture information though, which would need to be added to handle more realistic images.

A clear downside of raw voxel representations is their computational cost, which scales cubically with scene resolution. Typically, resolution is no larger than 64^3, which is coarse even for single object scenes. A classical idea to reduce this cost is to use sparse voxel representations, which only store information for non-empty voxels. When combined with octrees for fast indexing in 3D space, this can drastically reduce the memory footprint (Laine & Karras, 2010). Recent 3D reconstruction methods have combined this idea with other representation methods (Liu et al., 2020a; Zhang et al., 2022). Here, each non-empty voxel is associated with some other representation of geometry, such as a neural field. These are then, in effect, glued together into a scene by the underlying voxel structure.

5.3.2. Point Clouds

Another simple and natural representation of 3D geometry are *point clouds*. This simply refers to a set of 3D coordinates representing points on the surface (and sometimes within the interior) of solid objects. Point clouds appear naturally when depth information is available: When we know that a ray $\mathbf{r}$ first encountered a surface at depth d, we can compute the world coordinates of that point as $\mathbf{x} = \mathbf{c} + d\mathbf{r}$.

As point clouds are sets, order-invariant architectures such as transformers are well-suited to processing them. This straightforwardly allows point cloud segmentation and classification (Zaheer et al., 2017; Qi et al., 2017a;b). Point cloud prediction is somewhat complicated by the difficulty of formulating a difference metric between two point clouds S_1, S_2. If both are of the same size, a natural choice is the *earth mover's distance* (EMD) (Achlioptas et al., 2018)

$$d_{\text{EMD}}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \sum_{x \in S_1} ||x - \phi(x)||_2, \tag{5.1}$$

where ϕ is a bijection. The EMD is the minimum total travel distance required to move each point from S_1 to a corresponding point in S_2. The downside of this loss is that finding the globally optimal ϕ requires solving a matching problem, typically via the Hungarian algorithm. This takes (as least using the most common implementation) cubic time, and is not well suited to GPU acceleration, limiting the maximum size of point clouds. An alternative approach is the Chamfer loss, which replaces the global optimization with a greedy per-point approach:

$$d_{\text{CH}}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} ||x - y||_2^2 + \sum_{x \in S_2} \min_{y \in S_1} ||x - y||_2^2. \tag{5.2}$$

While this loss is much cheaper to compute, it can unfortunately introduce local minima, in which e.g. multiple points in S_1 are clumped around a single point in S_2. Both losses can be extended to sets of unequal size, typically by introducing a penalty term for the size differential, and allowing imperfect matchings when computing the EMD (Zhang et al., 2019; Kosiorek et al., 2020).

Notably, neither loss offers a straightforward way to formulate a probability distribution over point clouds. As a result, probabilistic models for point cloud prediction have employed implicit generative models, namely GANs (Achlioptas et al., 2018; Valsesia et al., 2019; Shu et al., 2019; Stelzner et al., 2020). Here, the loss is provided by a discriminator network trained to distinguish generated point clouds from ground truth ones.

While handling point clouds is possible, they are still far from ideal for the purpose of unsupervised representation learning. First, we generally require 3D supervision to obtain ground truth supervision, as there is no straightforward way to render an image from a point cloud. Second, handling point clouds is rather expensive, as the number of points gathered from object surfaces will grow roughly quadratically with the size of a scene. Since attention also scales quadratically with the number of entities,

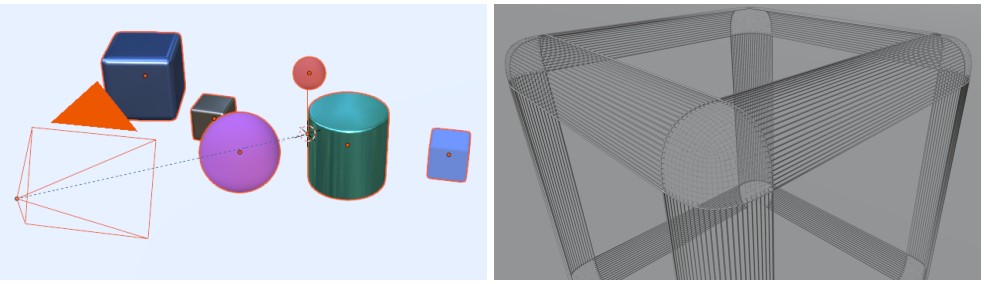

Figure 5.3.: (Left) Illustration of the mesh representation used to produce the synthetic CLEVR image depicted in Figure 5.1 (left), via the 3D modeling software Blender. (Right) Closeup of the wireframe model for the dark blue cube. Note that each of the cube's flat faces can be represented by a single polygon, whereas hundreds are used to model the rounded corners and edges.

the total cost is roughly quartic. Some methods have sought to reduce this cost by applying hierarchical attention (Qi et al., 2017b), or using inducing point methods (Lee et al., 2019; Stelzner et al., 2020). This is effective to some degree, but also introduces hard-to-tune hyperparameters.

5.3.3. Meshes

Meshes are the most widely used representation of 3D geometry in computing, forming the basis of most modern graphics engines (Foley et al., 2013). A mesh (V, F) consists of a set of 3D *vertices* $V \in \mathbb{R}^{n\times 3}$, and a set of polygonal *faces* F specifying visible surfaces. Each $f \in F$ is defined by a sequence of coplanar vertices $f = (v_1, \ldots, v_k), v_i \in V$. Frequently, without loss of generality, rendering is simplified by restricting F to only contain convex faces, quadrilaterals, or even purely triangles. We will only consider triangles going forward. An example is given in Figure 5.3.

Fast rendering of a mesh is typically accomplished using a process called *rasterization*, for which pseudocode is given in Algorithm 1. By first projecting the vertices of a given triangle f into image space, we can determine the set of pixels occupied by the resulting 2D triangle. For each of those pixels (i, j), we can then determine the intersection point P in world coordinates between that pixel's ray R and f. If P is closer to the camera than what was previously rendered at (i, j), we know that f lies in the foreground

Data: World-to-image projection matrix M, Camera position O, Mesh (V, F)
Result: Image $X \in \mathbb{R}^{h \times w \times 3}$, Depth map $D \in \mathbb{R}^{h \times w}$
for $i \in [1, h], j \in [1, w]$ **do**
  Initialize $D[i, j] = \infty$
end
for $f \in F$ **do**
  $f' \leftarrow (Mv_1, Mv_2, Mv_3)$
  for *Pixel* (i, j) *within* f' **do**
    Let r be the ray through (i, j)
    Let P be the intersection of r and f
    Let $d = \text{dist}(P, O)$
    if $d < D[i, j]$ **then**
      $D[i, j] = d$
      $X[i, j] = \text{color}(f, R)$
    end
  end
end

Algorithm 1: Coarse pseudocode of mesh rendering via rasterization, following Foley et al. (2013).

at that pixel, and that we should draw it here. To keep track of the depth values of previously drawn pixels, we use an array D called the depth buffer. The color value assigned when drawing a triangle f can depend on a variety of factors, such as texture, the viewing angle of R, or lighting conditions.

While this paints an extremely coarse picture of the functionality of a graphics engine, it nevertheless allows us to make a variety of observations relevant to our goals. First, this rendering procedure is very fast, since the inner loop over the pixels in the triangle may be vastly accelerated using hardware support and a variety of optimizations beyond the scope of this thesis. The computational cost is therefore roughly linear in the number of triangles. This conveniently means that unlike for the methods discussed previously, cost scales with the actual geometric complexity of a scene, as opposed to its size, since large, flat-sided objects may be represented using a small number of triangles. Figure 5.3 (right) illustrates this idea. Additionally, triangles can be processed in parallel, as long as the depth buffer is accessed atomically. These features together allow modern graphics engines to draw millions of polygons on the screen at 60 frames per second or higher. Secondly, we can observe that drawing a triangle is inherently

a discrete operation: We only draw a triangle f at a given pixel (i, j) if we determine that it is visible and in the foreground. This has consequences for machine learning models operating on meshes, which we will discuss next.

The initial challenge in building ML models to reason over meshes is to construct neural components which can handle them as inputs or outputs. On the input side, we can make use of the fact that a mesh is essentially a graph, and employ graph neural networks as discussed in Section 4.2. However, care must be taken to ensure that the geometric properties of the mesh are properly reflected, e.g. the fact that faces can have vastly different sizes. Hanocka et al. (2019) provide a method of this type capable of mesh classification and segmentation. Mesh prediction is somewhat more complicated, as it necessitates generating cohesive meshes from scratch. Wang et al. (2018a) have accomplished mesh prediction from a single input image by starting with a generic ellipsoid stucture, which is deformed by a GNN conditioned on the input image. The downside of this is that the mesh's graph structure remains static, and cannot be adjusted to match the given images. More recently, Nash et al. (2020) have proposed a more general solution by autoregressively predicting mesh vertices and faces using a transformer decoder. Both solutions are somewhat expensive, as attention incurs a quadratic cost over the number of vertices, and therefore in their current form limited to single objects with no more than a few thousand vertices.

In the context of unsupervised representation learning however, there is a more important limitation: Methods of this type are trained on datasets of human-created meshes, optimizing the similarity of the generated mesh and the ground-truth mesh. Training directly on images would require differentiating through the rendering procedure, i.e. computing a gradient $\partial\hat{x}/\partial z$ between the generated image $\hat{x}$ and the mesh z. As we saw above however, rasterization involves multiple discrete steps, meaning that this gradient will be extremely sparse, and the optimization full of hard-to-escape local minima. As a way out of this, some works have proposed differentiable mesh renderers based on a soft rasterization procedure, during which each triangle *bleeds* into the surrounding pixels, yielding nonzero gradients (Loper & Black, 2014; Kato et al., 2018; Liu et al., 2022). However, a difficult tradeoff remains in the tuning of the softness parameter: If rendering is too soft, the resulting image will be blurry, if it is too hard, there will be no sufficient gradient signal with respect to distant triangles, and the model is likely to get stuck in local optima. Perhaps as a result of this, these methods have only been successfully applied to single objects.

5.3.4. Neural Fields

A set of methods which have seen rising adoption in recent years are *neural fields*. Building upon the observation that the strength of neural networks lies in their ability to represent continuous functions as opposed to manipulating intricate discrete objects, a neural field f represents geometry by mapping each point in 3D space x to information about local geometry.

Occupancy Fields

In the simplest case, the output of f is simply a binary *occupancy* value, indicating if there is geometry present at x or not. Mescheder et al. (2019) have demonstrated how to leverage this idea for effective 3D reconstructions: They process an input image to obtain a latent shape variable $\mathbf{z}$, which is then used to condition an occupancy network $f(\mathbf{z}, \mathbf{x})$, simply implemented as an MLP with residual connections. The system is trained on ShapeNet by randomly sampling points $\mathbf{x}$ from the scene, and using corresponding ground truth occupancy information as supervision.

Compared to the other representations of geometry we have seen, neural fields have several advantages: First, as they represent shapes as continuous functions, they are independent of scene size and resolution, allowing them to represent both arbitrarily large scenes and minute details. Second, since the latent shape variable z is a latent with no particular structure, the model may learn to allocate representation capacity to whichever parts of a scene require it, irrespective of any particular geometric features. Third, occupancy fields are fast to train and evaluate, as the occupancy value for each point is computed independently. Finally, to facilitate downstream applications, the resulting shape representation may be compiled into other formats, such as meshes or voxels, by evaluating f at appropriately chosen samples. This can for instance be accomplished by using the marching cubes algorithm (Lorensen & Cline, 1987) to find an approximate isosurface $\{\mathbf{x} \in \mathbb{R}^3 \mid f(\mathbf{z}, \mathbf{x}) = \tau\}$, and connecting points sampled from the resulting set into a mesh (Mescheder et al., 2019).

Signed Distance Functions

While these features have made neural fields already useful, training them still required 3D supervision. This was changed by a series of works on the differentiable rendering

Data: SDF f, camera position $\mathbf{c}$, camera ray $\mathbf{r}$, maximum rendering depth t_{max}, minimum distance to surface ϵ.
Result: Point x at which $\mathbf{r}$ first intersects a surface, or $\emptyset$, if there is no such intersection.
Initialize $t = 0$
for $i = 0$ *to* max_iter **do**
  $\mathbf{x} = \mathbf{c} + t\mathbf{r}$
  $\delta = f(\mathbf{x})$
  $t = t + \delta$
end
return $\mathbf{x}$

Algorithm 2: Differentiable SDF rendering following the sphere tracing algorithm (Hart, 1996; Sitzmann et al., 2019)

of *signed distance functions* (SDFs) (Sitzmann et al., 2019; Jiang et al., 2020b; Liu et al., 2020b). An SDF is a function f mapping 3D points $\mathbf{x}$ not just to an occupancy indicator, but to the Euclidean distance to the nearest surface. Additionally, points *within* a solid object are mapped to a negative value. This information allows finding the point at which a ray $\mathbf{r}$ first intersects a surface relatively efficiently using the classic *sphere tracing* algorithm (Hart, 1996).

This algorithm follows a *ray marching* approach: Starting from the camera location $\mathbf{c}$, we follow the ray until we encounter a surface. Since the SDF's value $\delta > 0$ at a given location $\mathbf{x}$ lower bounds the distance to the nearest surface, we use it as the step size of our search. The classical version of the algorithm (Hart, 1996) contains a termination condition $\delta < \epsilon$. This is omitted in the neural variant to ensure parallelizability, differentiability, and robustness to erroneous SDFs in the early stages of training. The ability to pinpoint a 3D location responsible for the appearance of a certain pixel makes it possible to render an image by introducing a *color field* $g(\mathbf{x})$ to predict the observed color when looking at a given location. Sitzmann et al. (2019) and following works have shown that such a system can be trained jointly end-to-end for 3D reconstruction purely via novel-view synthesis.

Scaling these successes beyond single object scenes such as those from ShapeNet has proven challenging, however. One issue is that the color field is being only sparsely evaluated at the intersection points returned by the sphere tracing procedure. This can lead the model to converge to undesired optima early in training. High-quality results for complex scenes have only been demonstrated by Vicini et al. (2022), at the cost

of hourslong offline optimization on each scene, which is too slow for representation learning.

Neural Radiance Fields

A different approach to differentiable rendering of neural fields, called *Neural Radiance Fields* (NeRF), was provided by Mildenhall et al. (2020). Parts of our description of it in this subsection has been published as a pre-print in (Stelzner et al., 2021). NeRFs represent the geometry and appearance of a scene as a neural network $f : (\mathbf{x}, \mathbf{r}) \rightarrow (\mathbf{c}, \sigma)$, which maps world coordinates $\mathbf{x}$ and viewing direction $\mathbf{r}$ to a color value $\mathbf{c}$ and a density value $\sigma \geq 0$. To guarantee coherent geometry, the architecture is chosen such that the density σ is independent of the viewing direction $\mathbf{r}$. To simplify notation, we refer to the individual outputs of $f(\cdot)$ as $\sigma(\mathbf{x})$ and $\mathbf{c}(\mathbf{x}, \mathbf{d})$.

Rendering NeRFs is accomplished using classic ray marching techniques for *volume rendering* (Blinn, 1982; Kajiya & von Herzen, 1984). These simulate the effects of light traversing a volume of matter, such as a cloud. The density of matter σ at a given point $\mathbf{x}$ determines the fraction of light that is scattered as it traverses that point. This gives rise to an inhomogeneous spatial Poisson process, with the rate of scattering events governed by σ. Within this framework, standard equations for Poisson processes (Møller & Waagepetersen, 2003) allow us to compute the color observed for a given ray $\mathbf{r}$. While most machine learning works simply operate with the resulting equations, here we will give a brief derivation of them. This will later allow us to formulate a novel loss for depth supervision.

Consider a ray $\mathbf{r}(t) = \mathbf{x_0} + \mathbf{d}t$ of light arriving at a pinhole camera at position $\mathbf{x_0}$ along direction $\mathbf{d}$. The probability that light originating at point $\mathbf{r}(t)$ will not scatter and reach the camera unimpeded (transmittance) is equal to the probability that no events occur in the spatial Poisson process

$$T(t) = \exp\left(-\int_0^t \sigma(\mathbf{r}(t'))dt'\right). \tag{5.3}$$

NeRF does not model light sources explicitly, but assumes that lighting conditions are expressed by the color value of each point. As a result, the light emitted at the point $\mathbf{r}(t)$ in the direction of the camera $-\mathbf{d}$ has the color $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$, and its intensity is proportional to the density of particles $\sigma(\mathbf{r}(t))$ at that point. Consequently, the amount of light *reaching* the camera from $\mathbf{r}(t)$ is proportional to

$$p(t) = \sigma(\mathbf{r}(t))T(t). \tag{5.4}$$

In fact, under mild assumptions, $\mathrm{p}(t)$ is exactly equal to the distribution of possible depths t at which the observed colors originated[1]. We provide the full derivation in Appendix B.1.1.

We can now obtain the color observed for ray $\mathbf{r}$ as the expected color value under the depth distribution $p(t)$:

$$\hat{C}(\mathbf{r}) = \mathbb{E}_{t\sim \mathrm{p}(\cdot)}[\mathbf{c}(\mathbf{r}(t), \mathbf{d})] = \int_0^\infty \mathrm{p}(t) c(\mathbf{r}(t), \mathbf{d}) dt\,. \tag{5.5}$$

Typically, one chooses a maximum render distance t_{far} as the upper bound for the integration interval. This leaves the probability $\mathrm{p}(t > t_{\text{far}}) = T(t_{\text{far}})$ that light from beyond t_{far} is missed. To account for that, Eq. (5.5) is renormalized by dividing by $T(t_{\text{far}})$.

Computing $\hat{C}(\mathbf{r})$ requires approximating the two integrals in Eqs. (5.3) and (5.5) via sampling. To reduce the number of required samples, Mildenhall et al. (2020) introduced a two-step sampling scheme: First, coarse samples are drawn from a set of evenly-spaced bins along the ray, which provides an initial estimate of $p(t)$. A second, more fine-grained set of samples is then drawn from that distribution. Both sets of samples are then used together to compute $\hat{C}(\mathbf{r})$. This rendering procedure, which is illustrated in Figure 5.4, is fully differentiable and capable of producing photorealistic images. These are exactly the features we were looking for to enable unsupervised 3D representation learning on natural images. Consequently, NeRFs are trained by simply minimizing the L2 loss between the rendered colors $\hat{C}(\mathbf{r})$ and the colors of the training image $C(\mathbf{r})$, i.e. $\mathcal{L}_{\text{NeRF}}(\mathbf{r}) = \|\hat{C}(\mathbf{r}) - C(\mathbf{r})\|_2^2$. Rendering and training NeRFs is expensive, however, as approximating Eq. (5.5) requires evaluating f many times per pixel (256 in Mildenhall et al. (2020)). Initial work (Mildenhall et al., 2020; Martin-Brualla et al., 2021; Barron et al., 2021) has therefore focussed on fitting NeRFs to single scenes via hours of offline training. To make NeRFs useable for reinforcement learning, it is necessary to amortize this process into an inference network, similar to what we have seen for other representations. Recent work has done just that: Kosiorek et al. (2021) provide a VAE where the decoder parameterizes a NeRF conditioned on the latent representation z. Later work has identified inference workload as a key issue: Extracting the full geometry and appearance of a scene in a single inference pass is

[1] Note that the transmittance is symmetric, and that both occlusion and radiance depend on the density $\sigma(\cdot)$. Therefore, if a light source is present at $\mathbf{x_0}$, the distribution of points along $\mathbf{r}$ it illuminates is also given by Eq. (5.4). It can be convenient to think of ray marching as shooting rays of light from the camera into the scene instead of the other way around.

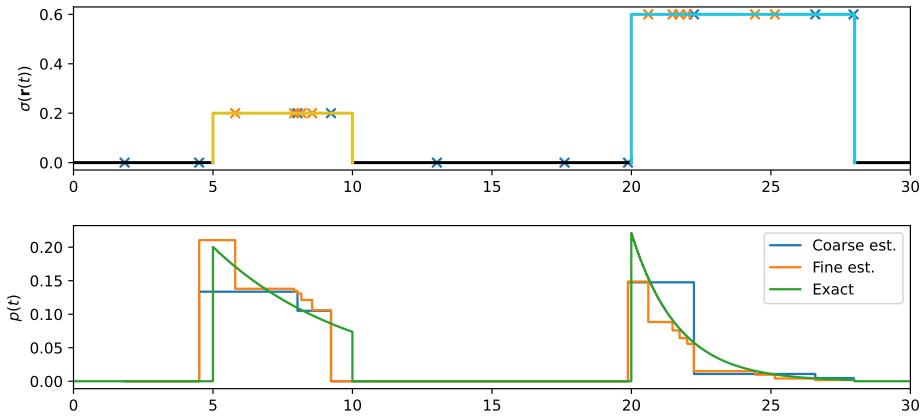

Figure 5.4.: An illustration of NeRF's ray casting procedure: (Top) A ray $\mathbf{r}$ encounters two volumes, each with constant density. The color of the plot indicates the color values output by the scene function $c(\mathbf{r}(t), \mathbf{d})$. The scene function is sampled in two passes: An initial coarse pass (blue X's) draws values for t from evenly sized bins, whereas a second, finer pass (orange X's) collects samples based on the initial estimate of the depth distribution. (Bottom) The true depth distribution (green) is approximated in two passes. The initial coarse estimate (blue) is refined using the additional fine samples (orange). The color value of the pixel corresponding to $\mathbf{r}$ is determined by mixing the color values of the two objects according to $p(t)$, yielding ●.

challenging. Yu et al. (2021a); Trevithick & Yang (2020) have therefore proposed also conditioning the NeRF decoder on those pixels from the input images corresponding to the rays pointed at the query point $\mathbf{x}$. This reduces the inference workload, while ensuring that the scene's geometry is still compactly represented in $\mathbf{z}$. Nevertheless, inference capacity remains a major issue, and explains the large gap between scene complexity that can be handled by amortized NeRF variance, compared to scene-specific NeRFs.

Another important feature that has become standard for neural fields is to apply *positional encodings* $e(\mathbf{x})$ based on Fourier features to the input coordinates $\mathbf{x}$. As discussed in Section 3.3.1, these have originally been introduced to facilitate order-dependent reasoning when applying attention to natural language. In the context of neural fields, they allow f to easily represent functions of a large frequency spectrum, which is critical to model both sharp details and large scenes.

5.3.5. Geometry-Free Representations

Finally, there is the simplest option: Avoid predicting geometry explicitly altogether, and simply predict novel views as images. A VAE model of this type was first presented by Eslami et al. (2018), demonstrating the ability to reconstruct synthetic environments in this way, and showing that the learned representations are useful for simulated robotics tasks. Sitzmann et al. (2021) have renewed this idea by building upon recent work on neural fields, introducing a network directly mapping rays to observed colors. A system capable of high-quality results on real-world data was provided by Rombach et al. (2021b), who utilized discrete autoregressive transformers to sample high quality novel views. This comes at the cost of expensive rendering, and means that each novel view generation involves sampling, resulting in mismatching views of the same scene. Very recently a series of works on *scene representation transformers* has scaled up the idea of predicting colors for rays directly using modern transformer components, enabling use on real-world data (Sajjadi et al., 2022c). Continuations of this work have also enabled object-centric representations (Sajjadi et al., 2022a) and training on unposed data (Sajjadi et al., 2022b).

A downside of the geometry-free approach is that there is no guarantee of 3D consistency: There are no structural constraints preventing the model from predicting incompatible images when queried on different viewpoints for the same scene. This means that the ability to output consistent imagery has to be fully aquired via learning. A consequence of this is that we generally cannot expect geometry-free methods to generalize to

camera poses not present in the training set. While the methods we have investigated so far come with well-defined rendering procedures which work from all viewpoints, geometry-free methods are simply trained neural systems, which will respond to out-of-domain inputs with unpredictable behavior. This makes geometry-free methods most applicable to domains with densely sampled camera poses, and precludes data with a small number of fixed cameras, which is for instance common for current robotics datasets. It also places some restrictions on minumum dataset size, as the training signal needs to be rich enough to learn a robust decoder for rendering.

5.4. ObSuRF: Decomposing 3D Scenes into Objects via Unsupervised Volume Segmentation

Having seen the success of NeRF-based representations for 3D reconstruction, we now ask whether we can leverage such representations for latent-variable modeling, such that the latent variables capture both semantic and structural information, without human supervision. Such representations may then be used for downstream reasoning tasks.

In keeping with our goals outlined at the start of the chapter, we focus on inferring unsupervised object-based representations of multi-object scenes in real time. In this section, we therefore present *ObSuRF* a model which learns to decompose scenes consisting of multiple *Objects* into a *Superposition of Radiance Fields*. We first encode the input image with a slot-based encoder similar to Locatello et al. (2020), but use the resulting set of latent codes to condition NeRFs (Mildenhall et al., 2020; Kosiorek et al., 2021) instead of directly generating 2D images. For training the model in 3D, we provide three RGB-D views of each scene, and optimize the model to match the observed depths and colors. To do so, we reframe NeRF's volumetric rendering as a Poisson process and derive a novel training objective, which allows for more efficient training when depth information is available as supervision. After confirming that the resulting model is capable of segmenting 2D images as well as or better than previous approaches, we test it on two new 3D benchmarks: A 3D version of CLEVR (Johnson et al., 2017) featuring multiple viewpoints, camera positions, and depth information, and MultiShapeNet, a novel multiobject dataset in which the objects are shapes from the ShapeNet dataset (Chang et al., 2015). This contents of this section have been published as a pre-print as Stelzner et al. (2021). In addition, both benchmarks and the code for our model has been provided at `https://github.com/stelzner/obsurf/`.

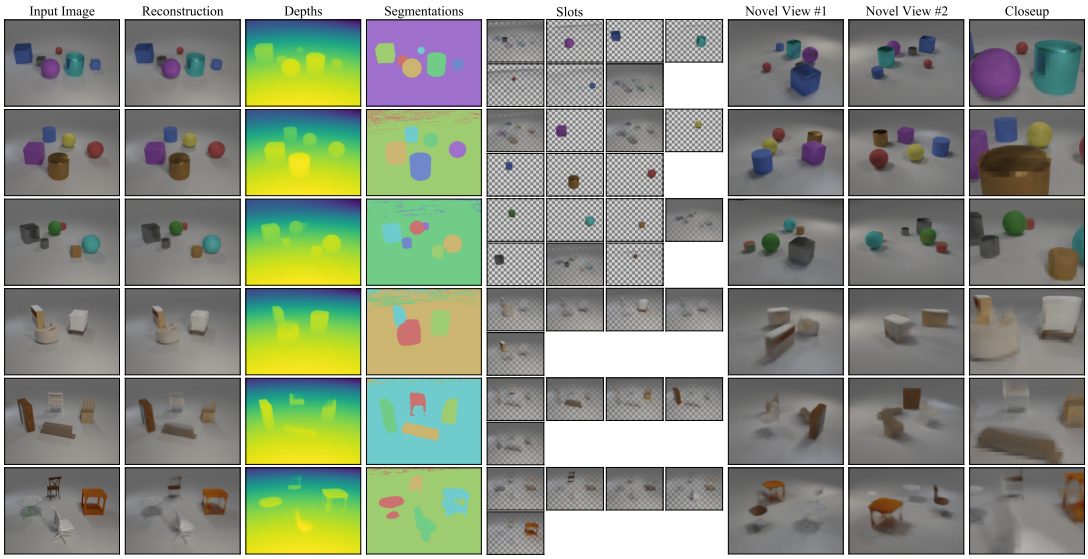

Figure 5.5.: ObSuRF uses a single image (left) to infer a set of NeRFs (slots) representing different objects. ObSuRF accurately models the geometry and appearance of CLEVR-3D scenes (top), and decomposes them into objects. It also produces details which are not visible in the input image, such as the full shape of partially occluded objects, or the shadows on the backsides of objects. This allows rendering the scene from arbitrary angles (right). The MultiShapeNet dataset (bottom) features a much larger variety of more complicated objects, such that ObSuRF cannot capture all of its details. However, it still learns to segment the scene into volumes corresponding to the objects, and to reproduce their dimensions, color, and position. For a full demo, please watch the supplementary videos at `https://sites.google.com/view/obsurf/`.

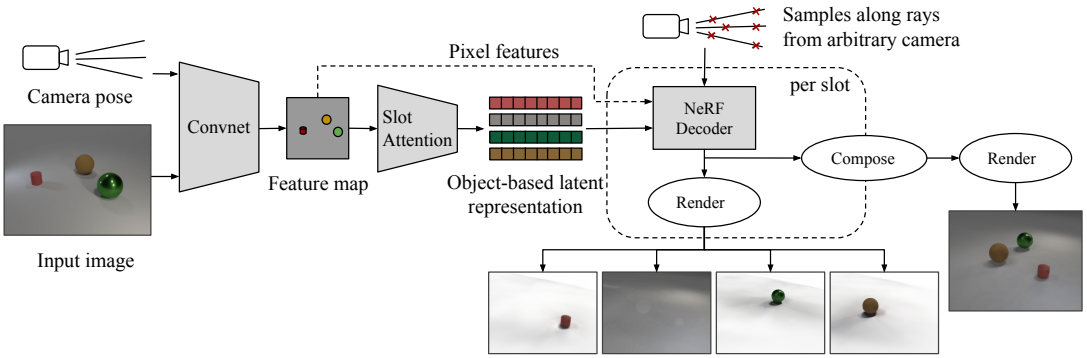

Figure 5.6.: ObSuRF architecture. The encoder is given an input image with the corresponding camera pose, and infers an object-based latent representation consisting of slots. These slots are used independently to condition a shared NeRF decoder. The volumes represented by the resulting scene functions may be rendered individually by querying them along rays coming from an arbitrary camera. Alternatively, the volumes may be composed in order to render the full scene.

5.4.1. Methods

ObSuRF encodes an input image into multiple NeRFs, each representing the 3D geometry and appearance of a separate object in the scene. We derive a principled way of composing n NeRFs into a single, albeit more expensive (by a factor of n), scene function. To decrease this cost, we show how to make NeRF integration cheaper when ground-truth depth is available at training time.

Learning to Encode Scenes as Sets of NeRFs

We now describe the full encoder-decoder architecture of ObSuRF (see Figure 5.6), which can *infer* scene representations from a single view of that scene—for *any* scene from a large dataset. ObSuRF then turns that representation into a set of NeRFs, which allows rendering views from arbitrary viewpoints, providing volumetric segmentation of the scene.

The encoder network f_{enc} infers a set of latent codes $z_1, \ldots, z_n$ (called slots) from the input image and the associated camera pose. Each slot represents a separate object, the

background, or is left empty. By using a slot z_i to condition an instance of the decoder network f_{dec}, we obtain the component NeRF $f_i(\cdot, \cdot) = f_{\text{dec}}(\cdot, \cdot; z_i)$. In practice, we set the number of slots n to one plus the maximum number of objects per scene in a dataset to account for the background.

Encoder. Our encoder combines recent ideas on set prediction (Locatello et al., 2020; Kosiorek et al., 2020). We concatenate the pixels of the input image with the camera position $\mathbf{x_0}$, and a positional encoding (Mildenhall et al., 2020) of the direction $\mathbf{d}$ of the corresponding rays. This is encoded into a (potentially smaller) feature map $\mathbf{y}$. We initialize the slots z_i by sampling from a Gaussian distribution with learnable parameters. Following Locatello et al. (2020), we apply a few iterations of cross-attention between the slots and the elements of the feature map, interleaved with self-attention between the slots (similar to Kosiorek et al. (2020)). These two phases allow the slots to take responsibility for explaining parts of the input, and facilitate better coordination between the slots, respectively. The number of iterations is fixed, and all parameters are shared between iterations. The resulting slots form the latent representation of the scene. We note that it would be straightforward to use multiple views as input to the encoder, but we have not tried that in this work. See Appendix B.2.1 for further details.

Decoder. The decoder (Figure 5.7) largely follows the MLP architecture of Mildenhall et al. (2020): We pass the Fourier-encoded inputs $\mathbf{x}$ through a series of fully-connected layers, eventually outputting the density $\sigma(\mathbf{x})$ and a hidden code $\mathbf{h}$. To condition the MLP on a latent code z_i, we use the code to shift and scale the activations at each hidden layer, a technique which resembles AIN by Dumoulin et al. (2017); Brock et al. (2019). The color value $\mathbf{c}(\mathbf{x}, \mathbf{d})$ is predicted using two additional layers from $\mathbf{h}$, $\mathbf{d}$, and a feature retrieved from the encoder's feature map $\mathbf{y}$. This feature is selected by projecting the query point $\mathbf{x}$ onto the plane of the input view's camera. This conditioning follows Yu et al. (2021a), except that we only use it to predict color, in order to not compromise the idea of the slots encoding the segmented geometry. We explored using z_i to predict an explicit linear coordinate transform between the object and the world space (Niemeyer & Geiger, 2020; Elich et al., 2021), but have not found this to be beneficial for performance. We also note that we deliberately choose to decode each slot z_i independently. This means we can treat objects as independent volumes, and in particular we can render one object at a time. Computing interactions between slots during decoding using e.g. attention (as proposed by Kosiorek et al. (2021)) would

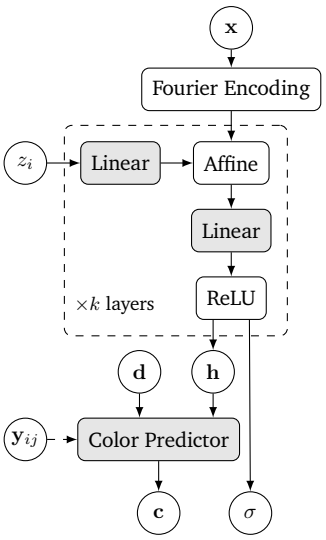

Figure 5.7.: ObSuRF's decoder is a type of conditional NeRF scene function. In each of k conditional blocks, we use a slot-specific latent code z_i to parametrize an affine transformation applied to activations coming from the block above. Color prediction uses the features $\mathbf{h}$, the viewing direction $\mathbf{d}$ and a projected feature $\mathbf{y}_{ij}$. Learnable components are shaded.

likely improve the model's ability to output complex visual features such as reflections, but would give up these benefits. Further details are described in Appendix B.2.2.

Composing NeRFs

Inputting the slots $z_i, \ldots, z_n$ into the decoder network yields a set of independent NeRFs $f_1, \ldots, f_n$, each with densities σ_i and colors $\mathbf{c}_i$. We now show how to compose these NeRFs into a single scene function. We arrive at the same equations as Niemeyer & Geiger (2020); Guo et al. (2020); Ost et al. (2021), but derive them by treating the composition of NeRFs as the *superposition* (Møller & Waagepetersen, 2003) of independent Poisson point processes, yielding a probabilistic interpretation of the algorithm.

Specifically, we assume that while we move along the ray $\mathbf{r}$, each of the n Poisson processes has a chance of triggering an event independently. The total accumulated

transmittance $T(t)$ from $\mathbf{r}(t)$ to $\mathbf{x_0}$—the probability of encountering no events along the way—should therefore be equal to the product of the transmittances $T_i(t)$ of each process:

$$T(t) = \prod_{i=1}^{n} T_i(t) = \exp\left(-\int_0^t \sum_{i=1}^{n} \sigma_i(\mathbf{r}(t))dt'\right). \tag{5.6}$$

This is equivalent to another Poisson process with density $\sigma(\mathbf{x}) = \sum_i \sigma_i(\mathbf{x})$.

To compute the color value, we additionally need to determine to what degree each of the component NeRFs is responsible for the incoming light. Following Eq. (5.4), the probability that NeRF i is responsible for the light reaching the camera from depth t is $\mathrm{p}(t, i) = \sigma_i(\mathbf{r}(t))T(t)$. Similarly to Eq. (5.5), we compute the pixel color by marginalizing over both depth t and component i, yielding

$$\hat{C}(\mathbf{r}) = \mathbb{E}_{t,i\sim\mathrm{p}(\cdot)}[\mathbf{c}_i(\mathbf{r}(t), \mathbf{d})] \tag{5.7}$$

$$= \int_0^\infty \sum_{i=1}^{n} \mathrm{p}(t, i)c_i(\mathbf{r}(t), \mathbf{d})dt. \tag{5.8}$$

It can also be useful to marginalize only one of the two variables. Marginalizing i yields the depth distribution $\mathrm{p}(t) = \sum_i \mathrm{p}(t, i)$ which we use to render scenes via hierarchical sampling as in (Mildenhall et al., 2020). By marginalizing t, one obtains a categorical distribution over the components $\mathrm{p}(i) = \int_t \mathrm{p}(t, i)$, which we use to draw segmentation masks. Finally, in order to compute the color loss derived above, we use the expected color $\mathbf{c}(\mathbf{x}, \mathbf{d}) = \sum_i \mathbf{c}_i(\mathbf{x}, \mathbf{d})\sigma_i(\mathbf{x})/\sigma(\mathbf{x})$.

Efficient Optimization Objective for RGB-D Data

We now turn to the optimization objectives we use to train ObSuRF. Even though Mildenhall et al. (2020) use a hierarchical sampling scheme, training NeRFs requires many function evaluations for each ray and is therefore extremely expensive. Moreover, as we show in Appendix B.1.2, this sampling produces a biased estimate for $\hat{C}(\mathbf{r})$, and, consequently $\mathcal{L}_{\text{NeRF}}$. This is caused by the nested integration in Eq. (5.5): If too few samples are collected, there is a significant chance that thin, high-density volumes are missed entirely, even though they would dominate the color term $\hat{C}(\mathbf{r})$ if it were to be evaluated analytically (see e.g. Rainforth et al. (2018a) for a thorough treatment of such issues).

To avoid this computational cost during training, we use depth supervision by training on RGB-D data, i.e. images for which the distance t between the camera and the visible surfaces is known. Instead of integrating over t, this allows us to directly maximize the depth log-likelihood $\log \mathrm{p}(t)$ in Eq. (5.4) for the known values of t. While we still need to approximate the inner integral, an unbiased estimate can be obtained with uniform random samples from $\mathrm{q}(\cdot) = \mathrm{Uniform}(0, t_{\text{far}})$ as $\log \mathrm{p}(t) = \log \sigma(\mathbf{r}(t)) - t\mathbb{E}_{t' \sim \mathrm{q}(\cdot)} \left[\sigma(\mathbf{r}(t'))\right]$. Since $\sigma(\mathbf{r}(t'))$ is likely to be large near the surface t and close to zero for $t' \ll t$, the variance of this estimator can be very large—especially at the beginning of training. To reduce the variance, we importance sample t' from a proposal $q'(\cdot)$ with a higher density near the end of the integration range,

$$\log \mathrm{p}(t) = \log \sigma(\mathbf{r}(t)) - \mathbb{E}_{t' \sim q'(\cdot)} \left[\sigma(\mathbf{r}(t'))/q'(t')\right] . \tag{5.9}$$

In practice, we set $q'(\cdot)$ to be an even mixture of the uniform distributions from 0 to $0.98t$ and from $0.98t$ to t, i.e. , we take 50% of the samples from the last 2% of the ray's length.

We fit the geometry of the scene by maximizing the depth log-likelihood of Eq. (5.9). Staying with our probabilistic view, we frame color fitting as maximizing the log-likelihood under a Gaussian distribution. Namely, since we know the point $\mathbf{r}(t)$ at which the incoming light originated, we can evaluate the color likelihood as $\mathrm{p}(C \mid \mathbf{r}(t), \mathbf{d}) = \mathcal{N}(C \mid \mathbf{c}(\mathbf{r}(t), \mathbf{d}), \sigma_C^2)$ with fixed standard deviation σ_C. Overall, evaluating the joint log-likelihood $\log \mathrm{p}(t, C)$ requires only two NeRF evaluations: At the surface $\mathbf{r}(t)$ and at a point $\mathbf{r}(t')$ between the camera and the surface. In practice we take the surface sample at $\mathbf{r}(t + \epsilon)$ with $\epsilon \sim \mathrm{Uniform}(0, \delta)$. This encourages the model to learn volumes with at least depth δ instead of extremely thin, hard-to-render surfaces.

Unsupervised End-to-End Training

ObSuRF is trained on a minibatch of scenes at every training iteration. For every scene, we use one image (and its associated camera pose, but without depth) as input to the encoder. This yields the latent state $\{z_i\}_{i=1}^n$. We then average the loss on a random subset of rays sampled from the available RGB-D views of that scene. Finally, we average across the minibatch.

One advantage of 3D representations is that they allow us to explicitly express the prior knowledge that objects should not overlap. This is not possible in 2D, where one cannot distinguish between occluding and intersecting objects. We enforce this prior by

Model	Sprites (bin)	Sprites	CLEVR
ObSuRF	$\mathbf{74.4 \pm 1.8}$	$\mathbf{92.4 \pm 1.3}$	98.3 ± 0.8
sel. runs	74.4 ± 1.8	93.1 ± 0.3	99.0 ± 0.0
SlotAtt.	69.4 ± 0.9	91.3 ± 0.3	$\mathbf{98.8 \pm 0.3}$
IODINE	64.8 ± 17.2	76.7 ± 5.6	$\mathbf{98.8 \pm 0.0}$
MONet	-	90.4 ± 0.8	96.2 ± 0.6
R-NEM	68.5 ± 1.7	-	-

Table 5.1.: Average foreground ARI on 2D datasets (in %, mean $\pm$ standard deviation across 5 runs, the higher the better), compared with values reported in the literature (Locatello et al., 2020; Greff et al., 2020a; Burgess et al., 2019; van Steenkiste et al., 2018). Best values are in bold. For one run on Sprites and 2 runs on CLEVR, ObSuRF learns to segment background from foreground. While desirable, this slightly reduces the foreground ARI scores as extracting exact object outlines is more difficult. To quantify this, we also report results with these runs excluded (*sel. runs*).

adding the *overlap loss* $\mathcal{L}_O(\mathbf{r}) = \sum_i \sigma_i(\mathbf{r}(t)) - \max_i \sigma_i(\mathbf{r}(t))$, optimizing the overall loss $\mathcal{L} = -\log p(t, C) + k_O \mathcal{L}_O(\mathbf{r})$. Experimentally, we find that $\mathcal{L}_O$ can prevent the model from learning object geometry at all when present from the beginning. We therefore start with $k_O = 0$ and slowly increase its value in the initial phases of training. In turn, we find that we do not need the learning rate warm-up of Locatello et al. (2020). We describe all hyperparameters used in Appendix B.4.

5.4.2. Experimental Evaluation

ObSuRF is designed for unsupervised volumetric segmentation of multi-object 3D scenes. As there were no published baselines for this setting, we start by evaluating it on 2D images instead of 3D scenes. This allows us to gauge how inductive biases present in a NeRF-based decoder affect unsupervised segmentation. We also check how a slotted representation affects reconstruction quality (compared to a monolithic baseline). We then move on to the 3D setting, where we showcase our model on our novel 3D segmentation benchmarks; we also compare the reconstruction quality to the closest available baseline, NeRF-VAE (Kosiorek et al., 2021), and provide ablation tests with regard to our RGB-D training method.

Metrics

We evaluate segmentation quality in 2D image space by comparing produced segmentations to ground truth using the *Adjusted Rand Index* (ARI, Rand (1971); Hubert & Arabie (1985)). The ARI measures clustering similarity and is normalized such that random segmentations result in a score of 0, and perfect segmentations in a score of 1. In line with Locatello et al. (2020), we not only evaluate the full ARI, but also the ARI computed on the foreground pixels (Fg-ARI; according to the ground truth). Note that achieving high Fg-ARI scores is much easier when the model is not attempting to segment the background, i.e. , to also get a high ARI score: This is because ignoring the background allows the model to segment the objects using rough outlines instead of sharp masks. To measure the visual quality of our models' reconstructions, we report the mean squared error (MSE) between the rendered images and the corresponding test images. To test the quality of learned geometry in the 3D setting we also measure the MSE between the depths obtained via NeRF rendering and the ground-truth depths in the foreground (Fg-Depth-MSE). We exclude the background as a concession to the NeRF-VAE baseline, as estimating the distance of the background in the kind of data we use can be difficult from visual cues alone.

Unsupervised 2D Object Segmentation

We evaluate our model on three unsupervised image segmentation benchmarks from the multi-object datasets repository (Kabra et al., 2019): CLEVR, Multi-dSprites, and binarized Multi-dSprites. We compare against four prior models as state-of-the-art baselines: SlotAttention (Locatello et al., 2020), IODINE (Greff et al., 2020a), MONet (Burgess et al., 2019), and R-NEM (van Steenkiste et al., 2018). We match the experimental protocol established by Locatello et al. (2020): On dSprites, we train on the first 60k samples, on CLEVR, we select the first 70k scenes for training, and filter out all scenes with more than 6 objects. Following prior work, we test on the first 320 scenes of each validation set (Locatello et al., 2020; Greff et al., 2020a) and process CLEVR images by center-cropping and resizing to 64×64.

To adapt our encoder to the 2D case, we apply the positional encoding introduced by Locatello et al. (2020), instead of providing camera position and ray directions. To reconstruct 2D images, we query the decoder on a fixed grid of 2D points, and again drop the conditioning on viewing direction. During training, we add a small amount of Gaussian noise to this grid to avoid overfitting to its exact position. Similar

to previous methods (Locatello et al., 2020; Greff et al., 2020a), we combine the 2D images which the decoder produces for each slot by interpreting its density outputs σ_i as the weights of a mixture, i.e. , we use the formula $\hat{C}(\mathbf{r}) = \sum_i c_i(\mathbf{r})\sigma_i/\sigma$. As we show in Appendix B.1.3, this is equivalent to shooting a ray through a superposition of volumes with constant densities σ_i and infinite depth. We train the model by optimizing the color reconstruction loss.

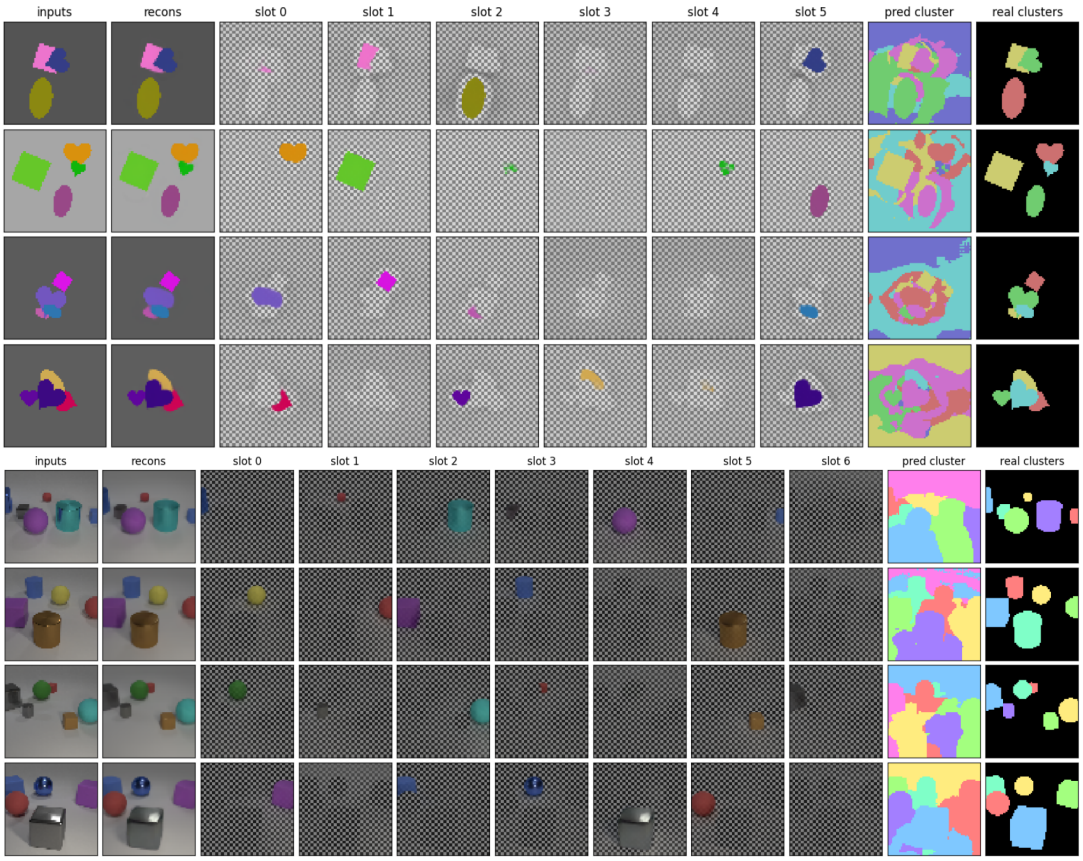

Figure 5.8.: Visualizations of ObSuRF's output given samples from the validation set of Sprites (top) and CLEVR2D (bottom). Slot images are rendered in front of a checkerboard background to illustrate their alpha values.

In Table 5.1, we compare the foreground ARI scores achieved by our model with those from the literature. Visualizations are given in Figure 5.8. We find that our model performs significantly better on the Sprites data, especially the challenging binary

variant, indicating that our decoder architecture imparts useful spatial priors. On CLEVR, our model largely matches the already nearly perfect results obtained by prior models. We note that like (Locatello et al., 2020), our model occasionally learns to segment the background in a separate slot, which is generally desirable but slightly reduces the Fg-ARI score as discussed above.

To quantify the impact of learning representations which are segmented into slots, we additionally construct a baseline which uses only a single slot, called NeRF-AE. To keep the total computational capacity comparable to our model, we set the size of this single latent vector to be equal to the total size of the slots ObSuRF is using. We also double the size of the hidden layers in both the encoder and the decoder. Still, as we show in Table 5.2, our model consistently achieves much lower reconstruction errors. This confirms previous results indicating that representing multi-object scenes using a set of permutation equivariant elements is not just helpful for segmentation, but also for prediction quality (Kosiorek et al., 2021; Locatello et al., 2020).

Training ObSuRF on these 2D datasets takes about 34 hours on a single V100 GPU. This is roughly 4 times faster than SlotAttention (133 hours on one V100 (Locatello et al., 2020)) and almost 40 times more efficient than IODINE (One week on 8 V100s (Greff et al., 2020a)). We attribute the efficiency gains compared to SlotAttention to our decoder architecture, which processes each pixel individually via a small MLP instead of using convolutional layers.

Unsupervised 3D Object Segmentation

To test ObSuRF's capability of learning volumetric object segmentations in 3D, we assemble two novel benchmarks. First, we adapt the CLEVR dataset (Johnson et al., 2017) from (Kabra et al., 2019) by rendering each scene from three viewpoints and also collecting depth values. The initial viewpoint is the same as in the original dataset,

Model	Sprites (bin)	Sprites	CLEVR
ObSuRF	1.34 ± 0.15	0.64 ± 0.11	0.41 ± 0.06
NeRF-AE	4.82	2.60	1.75

Table 5.2.: Testset reconstruction MSEs $\times 10^3$ of our model compared to the NeRF-AE ablation, which uses only a single latent vector.

Model	CLEVR-3D				MultiShapeNet			
	C-MSE	D-MSE	Fg-ARI	ARI	C-MSE	D-MSE	Fg-ARI	ARI
ObSuRF	**0.31**	**0.07**	96.9	92.0	**0.60**	**1.28**	94.7	61.1
$-\mathcal{L}_O$	0.60	0.25	94.0	1.4	0.82	3.32	**94.9**	16.9
− px. c.	0.78	0.10	95.7	**94.6**	1.81	3.44	81.4	**64.1**
− px. c., $\mathcal{L}_O$	0.80	0.12	85.5	4.83	1.78	3.40	94.4	16.5
+ NeRF l.	0.69	0.25	97.8	18.0	2.08	4.00	58.75	3.1
+ D-NeRF l.	1.11	0.15	**97.9**	15.7	4.7	8.95	12.8	0.2
NeRF-VAE	4.7	1.03	-	-	5.5	110.3	-	-

Table 5.3.: Quantitative results on the 3D datasets. C-MSE scores are color MSE $\times 10^3$; D-MSE are depth MSE values computed on the foreground only. px. c. is pixel conditioning. ARI scores are in percent, best values in bold. $\mathcal{L}_O$ is the overlap loss.

the others are obtained by rotating the camera by $120°/240°$ around the z-axis. As in the 2D case, we restrict ourselves to scenes with at most 6 objects in them. Second, we construct MultiShapeNet, a much more challenging dataset which is structurally similar to CLEVR. However, instead of simple geometric shapes, each scene is populated by 2-4 objects from the ShapeNetV2 (Chang et al., 2015) 3D model dataset. The resulting MultiShapeNet dataset contains $11\,733$ unique shapes. We describe both datasets in detail in Appendix B.3. In contrast to the 2D case, we utilize the full 320×240 resolution for both datasets. Due to our decoder architecture, we do not need to explicitly generate full-size images during training, which makes moving to higher resolution more feasible.

When testing our model, we provide a single RGB view of a scene from the validation set to the encoder. We can then render the scene from arbitrary viewpoints by estimating $\hat{C}(\mathbf{r})$ (Eq. (5.8)) via hierarchical sampling as in (Mildenhall et al., 2020). We set the maximum length t_{far} of each ray to 40 or the point at which it intersects $z = -0.1$, whichever is smaller. This way, we avoid querying the model underneath the ground plane, where it has not been trained and where we cannot expect sensible output. We use a rendering obtained from the viewpoint of the input to compute the reconstruction MSE. In order to obtain depth maps and compute the depth-MSE, we use the available samples to estimate the expected value of the depth distribution $\mathbb{E}_{p(t)}[t] = \int_0^{t_{\text{far}}} t\, \mathrm{p}(t) dt$. We draw segmentation masks by computing $\arg\max \mathrm{p}(i)$. Finally, we also render individual slots by applying the same techniques to the individual NeRFs, following Eqs. (5.4) and (5.5). In order to highlight which space they do and do not occupy, we

use the probability $\mathrm{p}(t \leq t_{\text{far}})$ as an alpha channel, and draw the slots in front of a checkerboard background.

We report results for the full ObSuRF model, which uses both the overlap loss $\mathcal{L}_O$ and pixel conditioning, and a number of ablations. These do not use the overlap loss ($-\mathcal{L}_O$), or pixel conditioning ($-$ px. c.). Additionally, we report results for ObSuRF trained with vanilla NeRF rendering ($+$ NeRF l.) and with NeRF rendering plus depth loss of (Deng et al., 2021) ($+$ D-NeRF l.). Details on these ablations are given in Appendix B.2.5. The slot sizes are 128 for CLEVR-3D and 256 for MultiShapeNet. As a baseline, we compare to NeRF-VAE which, similarly to our model, is an autoencoder where the NeRF decoder is conditioned on a latent extracted from a single input image. However, it is trained without depth information, and does not segment the scene. Instead, it maximizes the evidence lower-bound (ELBO), comprised of a reconstruction term and a regularizing KL-divergence term. It also uses much larger latent representations of size $8 \times 8 \times 128$, and an attention-based decoder. While the differences in architecture make training times impossible to directly compare, we estimate that due to our RGB-D based loss, ObSuRF required 24 times fewer evaluations of its scene function than NeRF-VAE over the course of training.

Figure 5.5 and Table 5.3 contain qualitative and quantitative results, respectively. We find that ObSuRF learns to segment the scenes into objects, and to accurately reconstruct their position, dimensions, and color. Due to the smaller variety in object shapes, the geometry tends to be sharper on CLEVR-3D when compared to MultiShapeNet. As expected, pixel conditioning increases the accuracy of predicted colors and geometry. We observe that similarly to the 2D models, our model does not learn to segment the background into its own slot without $\mathcal{L}_O$. With this additional loss term however, it learns to also segment the background, leading to much higher ARI scores. On CLEVR3D, both versions of NeRF training yields similar results to our loss, whereas on MultiShapeNet, they fail to produce accurate segmentations or geometry, with depth supervision (Deng et al., 2021) actually performing worse. Compared to NeRF-VAE, we find that our model achieves much lower reconstruction errors on both datasets.

In the more challenging MultiShapeNet dataset, we also find that our training method converges much faster, as shown by the learning curves in Figure 5.9. There, we compare ObSuRF (without $\mathcal{L}_O$) and the NeRF based ablations in Figure 5.9. While the models behave comparably on CLEVR3D, ObSuRF converges much faster and to a much better segmentation result on Shapenet. Training time on the x-axis is measured in scene function evaluations, as these make up at least 99% of the computational cost of training each model.

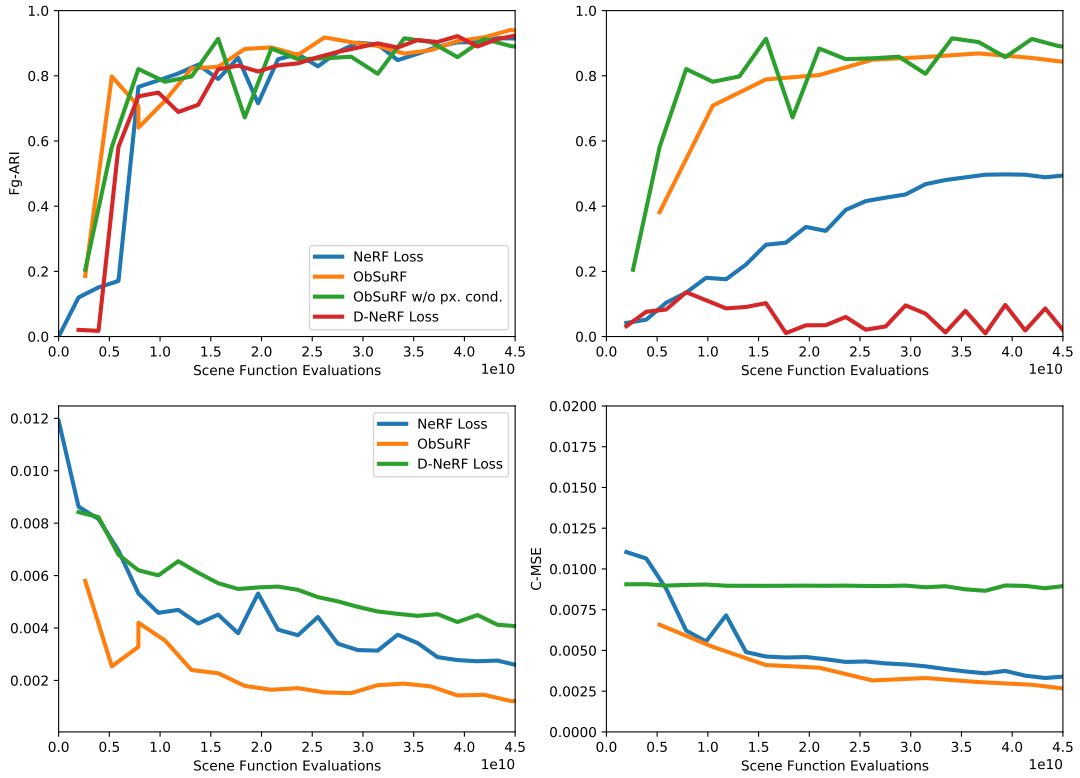

Figure 5.9.: Learning curves of ObSuRF and its ablations measuring Fg-ARI (top) and C-MSE (bottom) when training on CLEVR3D (left) and Multishapenet (right). For better comparability with NeRF-based training, all ObSuRF models are trained without the overlap loss $\mathcal{L}_O$.

Model	Size	Shape	Color	Material
ObSuRF	99.8%	97.1%	13.2%	52.1%
w/o px. cond.	99.7%	94.1%	94.1%	83.5%

Table 5.4.: CLEVR3D validation set accuracy of a 3-layer MLP predicting ground truth object labels based on ObSuRF's latents. 1000 labelled scenes were used for training the classifier, ObSuRF remains unsupervised. Slots and labels were matched based on segmentation mask overlap. On Shapenet, the shape prediction accuracy is 89.5% for ObSuRF, and 88.3% for ObSuRF without pixel conditioning.

Finally, we investigate the utility of ObSuRF's unsupervised slot representations for downstream tasks. To do so, we use the representation to predict the objects' ground truth properties. We utilize a three layer ReLU MLP with hidden size 128, and train it using 1000 scenes from the training set with ground truth annotations. Slots are matched to objects based on the maximum segmentation mask overlap. We report validation set results in Table 5.4, finding that our models achieve high accuracy on the geometric features (shape, size) on both CLEVR3D and MultiShapenet (see caption). We also find that accurately predicting features concerning surface appearance (color, material) based on the slots is only possible when pixel conditioning is *not* used. This highlights how important information bottlenecks are for the functioning of the model: Pixel conditioning allows ObSuRF to directly predict colors using information from the feature map, obviating the need to encode them in the slots.

Exploratory Real-World Results

To explore the viability of using ObSuRF on real-world data, we apply it to the DexYCB dataset (Chao et al., 2021), building upon work by Nematpur (2022). We eliminate the dynamic aspect of the dataset by only utilizing the first frame of each video clip. Additionally, we exclude the cameras facing the experimenter, as they do not provide a clear view of the scene. This results in a dataset of 1000 scenes with 4 views each. Following the evaluation setup **S0** from Chao et al. (2021), we withhold 20% of scenes per experimenter for testing and validation, and use the rest for training.

On this data, we train an ObSuRF model with pixel conditioning for 1 million iterations. We introduce the overlap loss between iterations 120k and 180k. The model achieves a

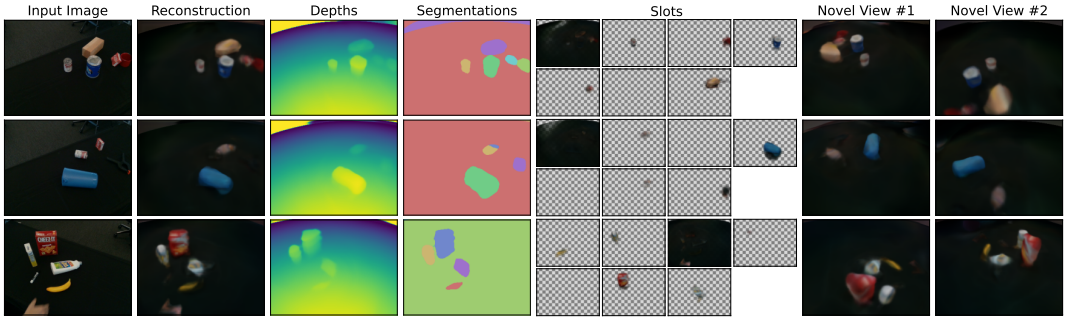

Figure 5.10.: Qualitative results of ObSuRF on the real-world DexYCB dataset (Chao et al., 2021). Objects are generally correctly segmented, but finer geometric details are lost.

foreground ARI of 52.8% and an ARI score of 52.7%. Qualitative results are shown in Figure 5.10: The model generally segments the objects correctly, but misses smaller geometric details. This highlights the limits of deterministic losses for novel view synthesis, which will lead to blurry results when faced with the uncertainty and noise present in real-world data.

5.4.3. Related Work

Aside from the unsupervised segmentation models introduced earlier in this thesis, ObSuRF builds upon prior work on implicit representations of 3D geometry, as well as early works on 3D segmentation.

Depth Supervision for NeRF. As we argued in Section 5.3.4, NeRF (Mildenhall et al., 2020) is well suited for representation learning because it can be trained via novel view synthesis, but suffers from high computational cost to do so, due to the use of ray marching. For ObSuRF, we opt for a pragmatic middle ground between full 3D supervision and costly ray marching: We assume access to views of the training scenes enriched with depth information (RGB-D), and show how this allows for training NeRFs without expensive raymarching.

Using depth supervision to improve NeRF training has concurrently been proposed by Deng et al. (2021), without however avoiding ray marching and the corresponding performance costs. An alternative approach relies on multiview-stereo to infer depth as

in Lin et al. (2021). This does not need depth and speeds up test-time rendering, but unlike ObSuRF, it relies on access to a large number of observed views.

Unsupervised 3D Segmentation Most prior work on unsupervised 3D segmentation has focused on segmenting single shapes into parts, and is trained on ground truth geometry. BAE-NET (Chen et al., 2019) reconstructs voxel inputs as a union of occupancy functions. CvxNet (Deng et al., 2020a) and BSP-Net (Chen et al., 2020b) represent shapes as a union of convex polytopes, with the latter capable of segmenting them into meaningful parts. Similarly, UCSG-Net (Kania et al., 2020) combines SDF primitives via boolean logic operations, yielding rigid but interpretable shape descriptions.

Only a small number of publications attempt to segment full scenes into objects. Ost et al. (2021) train a model to segment a complex real-world video into objects, but doing so requires a full scene graph, including manually annotated tracking information such as object positions. In GIRAFFE (Niemeyer & Geiger, 2020) object-centric representations emerge when a set of NeRFs conditioned on latent codes is trained as a GAN. However, the focus lies on image *synthesis* and not *inference*. Consequently, it is not straightforward to obtain representations for a given set of input images. BlockGAN (Nguyen-Phuoc et al., 2020) achieves similar results, but uses latent-space perspective projections and CNNs instead of NeRFs. RELATE (Ehrhardt et al., 2020) builds on this to allow the manipulation of scenes. Guo et al. (2020) show how to improve visual quality when rendering novel arrangements of objects by learning scattering functions, but do not infer such representations from given multi-object scenes. Our method is unsupervised, and admits efficient inference with a single forward-pass of the encoder. Elich et al. (2021) present a model which encodes a single input image into a set of deep SDFs representing the objects in the scene. In contrast to our work, theirs requires pretraining on ground-truth shapes and operates on simpler scenes without shadows or reflections. Closest to our work is a system concurrently developed by Yu et al. (2021b), which also learns to segment scenes into a set of NeRFs using a slot based encoder. However, they train using NeRF rendering and mitigate its computational cost by adjusting the rendering resolution.

Some recent models have also used geometry-free novel view synthesis to train image-space segmentation models (Nanbo et al., 2020; Chen et al., 2021; Kabra et al., 2021). Since multi-view consistency is not enforced in the model structure, these models tend to exhibit some visual artifacts when queried from novel viewpoints, and do not yield 3D segmentation volumes directly.

5.4.4. Conclusion

We have presented ObSuRF, a model that segments 3D scenes into objects represented as NeRFs. We have shown that it learns to infer object positions, dimensions, and appearance on two challenging and novel 3D modeling benchmarks. Importantly, it does so using purely observational data, without requiring supervision on the identity, geometry or position of individual objects.

Since ObSuRF learns to infer compact object representations that *have to* be well informed of their geometry (due to the model structure), using these representations may be beneficial for downstream tasks including robot grasping, dynamics modeling, and visual question answering. A key step in this direction is to improve inference capabilities and robustness to noisy depth supervision, which will enable working with real-world data. Of equal importance is the creation of large-scale real-world multi-object datasets.

5.5. Current Issues in 3D Representation Learning

We now turn to two current research questions in 3D representation learning: How to model uncertainty, and how to incorporate unposed training data. Taken together, these represent the most significant challenges on the path towards large scale 3D scene understanding.

5.5.1. Modelling Uncertainty for Novel View Synthesis

A key issue we have so far skipped over is handling uncertainty during novel view synthesis. Clearly, this task features a fair amount of inherent, aleatoric uncertainty, as not every corner of the scene may be observed directly from the input images, and multiple possible completions are possible. In a deterministic system such as ObSuRF, we cannot model this uncertainty, and the model is instead incentivized to predict the mean of the multimodal distribution of possible scene extensions, which will lead to blurry reconstructions. This issue gets increasingly pronounced as more complex scenes are considered: Not only are there more unobserved areas, but the model might also lack the inference and representation capacity to capture all fine details. Consider for instance a scene containing a plant, such as the ones considered in the original, scene-specific NeRF paper (Mildenhall et al., 2020). A deterministic model will only be

able to yield non-blurry reconstructions if it can capture the poses of all leaves, which is infeasible for amortized models in the near-term. A probabilistic model on the other hand would be able to learn a representation for the pose and type of plant, and sample a plausible instantiation of that representation in the decoder. While the resulting image might not match the exact input image, it will as least be a plausible image of that plant. The challenge now is to formulate a probabilistic model with sufficient capacity to capture this distribution, and to find a way to train it.

A natural place to look for a solution are inpainting methods for 2D images. They are faced with the same problem of having to model a complicated distribution over possible extensions of a given partial image. The most successful approaches in this space leverage autoregressive (Esser et al., 2021) or, more recently diffusion models (Rombach et al., 2021a; Ramesh et al., 2021) to represent these distributions. The issue with translating these successes to the 3D setting is that they rely on ground truth observations from the learned distribution, i.e. , the full image for an inpainting task [2]. If we directly apply these ideas to 3D, they will allow us to sample novel views $p(\mathbf{x}'|\mathbf{x})$, but not *3D scene representations* $p(\mathbf{z}|\mathbf{x})$, as we generally have no ground truth information for the latter. Rombach et al. (2021b) have demonstrated this using autoregressive transformers: While they are able to synthesize high-quality novel views, these views are not internally consistent, as sampling takes place for each generated image. Watson et al. (2023) have trained a diffusion model under a similar setup, and sought to mitigate this problem by conditioning the model on *every* known image of a scene, including the previously generated ones. This improves 3D consistency, but comes at an increased computational cost, and does not yield a compact 3D scene representation. NerfDiff (Gu et al., 2023) also uses a conditional diffusion model to synthesize high-quality images, but conditions it on the NeRF renders corresponding to the queried view, as opposed to an unstructured scene description, to improve 3D consistency. In addition, it includes a fine-tuning scheme to improve the consistency of a set of generated views post-hoc.

To avoid these consistency issues altogether, an attractive alternative approach is to learn a powerful distribution over scene representations $p(\mathbf{z}|\mathbf{x})$, and use a near-deterministic rendering approach for $p(\mathbf{x}'|\mathbf{z})$, ensuring 3D consistency. But what probabilistic model should be used for $p(\mathbf{z}|\mathbf{x})$? A VAE, as proposed by Kosiorek et al. (2021), is simplest to implement, but remains far from the image quality of 2D diffusion models. A GAN

[2] Note that, while Rombach et al. (2021a) perform diffusion in a latent space, this space is carefully engineered to approximately map one-to-one to observed images via perceptual image compression methods. Such a relationship can clearly not exist between 3D scene representations and 2D images.

would work in principle, but would require a complex discriminator $f(\mathbf{x}, \mathbf{x}')$ capable of determining if $\mathbf{x}, \mathbf{x}'$ are from the same scene, while also coming with a notorious set of training instabilities. Perhaps as a result, GANs for 3D synthesis have generally focussed on unconditional scene synthesis as opposed to inferring scene representations from given images (Niemeyer & Geiger, 2020). As discussed above, autoregressive and diffusion models generally require ground truth observations for training. Some recent works have therefore opted for a two-stage training approach (Bautista et al., 2022; Wang et al., 2022): First, for the training images $\mathbf{x}$, suitable 3D representations are determined using an autodecoding approach, optimizing

$$\max_{\theta, \mathbf{z}} p_\theta(\mathbf{x}|\mathbf{z}). \tag{5.10}$$

In a second step, these representations $\mathbf{z}$ are used as ground truth to train a conditional diffusion model $p(\mathbf{x}|\mathbf{z})$ for inference. While this approach produces high-quality results, it is much less convenient compared to end-to-end training: The latent representations $\mathbf{z}$ need to be individually optimized for each training example and stored offline. Additionally, since $\mathbf{z}$ is optimized independently from the inference procedure, it might not be best suited for inference.

Overall, finding a method that captures the complex latent distribution over scene descriptions and can also be trained in a convenient way remains one of the largest challenges in 3D representation learning.

5.5.2. Leveraging Unposed Training Data

One of the assumptions we made at the beginning of the chapter was that we have access to camera poses for all our training images. This severly constrains the amount of available data. While some large, real-world datasets of this type exist, e.g. the Google Streetview data used by (Sajjadi et al., 2022c), they tend to be constrained to certain domains, are frequently proprietary, and are still orders of magnitude smaller when compared to the 2D image datasets used to train large 2D models (Schuhmann et al., 2022).

Building similarly large 3D models therefore depends on removing the requirement of pose availability, thereby unlocking new sources of data. Of relatively minor importance are the relative poses of the input images: Sajjadi et al. (2022c) showed that lacking this information only leads to a minor loss of performance. More problematic is the pose of the novel view used as the prediction target, relative to the input images. Without this,

it is not clear what image the model should even generate, and difficult to formulate a meaningful training objective. Sajjadi et al. (2022b) recently demonstrated a potential way out of this: They introduced a pose estimator network $g(\mathbf{z}, \mathbf{x}')$ to predict the relative pose between the input views, and the target image $\mathbf{x}'$. The scene representation is then rendered from that pose, and the whole system is trained to match the ground truth novel view. Importantly, the only component of the system that gets to *see* the target view is the pose predictor, and it can only communicate this information through its predicted pose. This means that $\mathbf{z}$ still needs to contain a full scene representation, which is only projected down to a particular view after it receives the pose from the pose estimator. In this manner, it is possible to bootstrap a novel view synthesis model from unposed data. The largest source of unposed multiview data available on the Internet today is videos. Unfortunately, they pose another set of challenges, namely the fact that the contents of the scene generally do not stay constant across frames. Unlocking the potential of this data source by combining 3D models with a dynamics components will therefore be one of the major goals of future research.

An alternative line of research seeks to extract 3D understanding from large scale 2D image models, without relying on multiview data at all. Poole et al. (2022) have shown that a NeRF representation can be optimized to match a textual prompt by rendering it into images, and maximizing the likelihood score a text-to-image model assigns to those. An extension to video, represented by a 4D NeRF conditioned on time and space, was provided by Singer et al. (2023). Additionally, Melas-Kyriazi et al. (2023) have provided a view-conditioned variant. All of these works are of limited relevance to representation learning, as they rely on length optimizations for each scene. For a detailed overview of these ideas, we refer the reader to Kosiorek (2023).

6. Conclusion

In this chapter, we will conclude the thesis by reviewing our contributions, formulating lessons learned, and highlighting directions of future work.

6.1. Lessons Learned

In addition to the technical results presented in each chapter, we can also notice some larger themes and trends that become apparent throughout this work. Here, we attempt to formulate them as a series of leassons learned.

Set-structured representations naturally yield semantically meaningful segmentations. A central technique we have used in this thesis is to build autoencoder models with set structured latent representations. In all three modalities considered, we have shown examples of such models capable of learning representations segmented into objects in an unsupervised way, purely based on visual observations. This supports the idea that predictive coding is a powerful objective for constructing meaningful high-level representations. We were also able to identify certain properties that models should have to facilitate this goal. First, we have consistently found segmented representations to be more effective representations of visual scenes than monolithic vectors, whether for ordinary images, dynamics prediction, or 3D modeling. Second, we have found it important for the elements of our representation to be processed in an order-equivariant manner, to account for the exchangeability of objects. Finally, we have seen the power of probabilistic modeling to express our uncertainty about the contents of the observed scene.

Additional modalities are key for overcoming visual confounders. We have also discussed the main failure mode of unsupervised scene understanding models: Due to the visual confounders prevalent in natural images, the most efficient representation of such an image will more closely resemble a classical image compression method's clustering of pixels, than the high-level scene description we desire (Section 3.4). This poses a limit to fully unsupervised learning on 2D images, and has led us to explore other modalities, in which the signal-to-noise ratio between confounders and meaningful observations is more favorable. In a video setting, we may make use of temporal consistency to more closely delineate objects, whereas 3D observations give us clues as to the 3D geometry of a scene. Importantly, both of those modalities do not just provide us with richer data, but also with new unsupervised training objectives, namely dynamics prediction and novel view synthesis. Compared to a simple 2D autoencoding objective, these suffer from fewer undersirable local optima: On both objectives, high scores can only be achieved with a representation that capture a scene's objects and geometry.

Explicitly enforcing hard segmentations comes at a cost... Throughout much of this thesis, we have evaluated the quality of learned representations by individually decoding its components, and checking whether they match the objects present in the scene. This *hard* segmentation approach comes with significant costs: Most obviously, decoding is more computationally expensive, as the components are processed independently, and some computation might be redundant. Additionally, pushing the model to learn a hard segmentation often requires additional hand-designed loss terms, such as the overlap loss for ObSuRF (Section 5.4.1). Alternative models which have eschewed the goal of producing hard segmentations, and have purely focussed on successfully explaining the given data, have, all else being equal, generally been more scalable, and more general. An example of this is the comparison between ObSuRF and SRT (Sajjadi et al., 2022c). SRT also uses a set-structured latent representation, but does not decode its components independently. Instead, they interact through attention within the decoder to produce the final output. This gives the model the ability to use more efficient representations: Each element of the set representation does not need to represent a specific object, but can simply be an arbitrary piece of knowledge contributing the overall representation. As a result, SRT may be trained purely using novel view synthesis, and generalizes to more complex data. The authors have also shown that it can be used to obtain what one might call *soft* segmentations: Since the representation implicitly knows about objects, one can easily train a decoder using a small amount of ground truth data to predict scene segmentations.

...but provides guarantees and insights into representations. Despite these downsides, there are several reasons why investigating models with explicit structure is fruitful nontheless. First, they give us insights into the power of various biases, allowing us to answer questions on e.g. the importance of temporal consistency for object tracking. Second, hard segmentations are useful for scene manipulation and editing: Especially in the context of 3D models, the ability to move, delete, and insert objects is sought after by many practicioners. Third, they give us clues into which architectures are promising for larger-scale representation learning: As we have discussed in Section 3.5, a successful unsupervised object segmentation model demonstrates that the inference architecture used is powerful enough to infer good representations, and the learning objective is meaningful enough that it yields a desirable optimum. This means that a slightly relaxed version of this model will likely still yield good representations, just without explicit semantics, which then highlights an avenue for more large-scale and general representation learning.

6.2. Future Work

Given these lessons, we now discuss directions for future research. While we have already hinted at some technical avenues in the conclusions of the previous chapters, here we take a more high-level view.

Building large scene models. The successes of large models for language and images in the past few years have shown the power of simple but challenging learning objectives combined with massive datasets. It can be hypothesized that the objectives discussed in this thesis, novel view synthesis and dynamics prediction, would be similarly effective when combined with sufficiently large data. We have discussed the main technical challenges to doing so in Section 5.5: On the architecture side, better ways to handle uncertainty in 3D are required. The question then becomes where to obtain sufficiently large data. This induces a tradeoff between cost of obtainment and modeling difficulty: Posed multiview data is costly to collect but easiest to model, whereas unposed data is easier to acquire but comes with additional challenges. On the far end of the opposite spectrum is unposed video data, which is abundant on the Internet, but is much more difficult to model due to the presence of dynamics.

At this point, it is unclear which of these sources will first allow the creation of such models. Conceivably, certain proprietary multiview datasets, like those collected from

fleets of cars equipped with sensors for self-driving, could massively accelerate large scene models. Alternatively, their development might also follow the trajectory seen for language and image models, where the sheer scale of data available on the open Internet proved most effective.

Integrating 3D and Dynamics Finding effective ways to combine 3D and dynamics modeling is critical to unlocking video as a source of data. A challenge in doing so is that this type of data softens the notion of 3D consistency: The change between two consecutive frames of video may be attributed to a change of viewpoint, or a change in the scene, or both. This means we cannot guarantee multi-view consistency between frames, which makes it difficult to employ strict geometric models such as NeRF. On the other hand, an advantage compared to unposed multiview data lies in the fact that both camera and object trajectories in a video are largely smooth and continuous. But even this will not always hold true, as real-world videos may contain cuts or special effects. These issues highlight that enforcing hard, structural constraints will become increasingly difficult, and motivates finding ways for models to learn about them organically. Potential ways to do this were demonstrated by Sajjadi et al. (2022b), using geometry-free view synthesis and pose prediction.

Interaction and Embodiment One aspect we have only touched on tangentially in this thesis is the question of learning to interact with the environment. In Chapter 4, we have discussed how learned models of the environment may be leveraged for efficient reinforcement learning, allowing us to learn policies to achieve certain objectives. However, we should be careful to not leave this as an afterthought, as there are a number of interactions between the representations we use to think about the world, and the ways we act in it. In this thesis, we have largely assumed that learning about the pose and geometry of objects is a useful prerequesite for acting in it. While this is probably a fine assumption, the desired level of detail will depend on the required tasks. Thus, tasks will to some degree inform what representations we want to infer.

Additionally, interactability can also change the way we obtain our representation of the world: We can imagine a variety of behaviors central to capturing good representations, such as an agent purposefully modifying their own viewpoint to capture more aspects of a scene, or experimenting with objects to better estimate their characteristics. These considerations highlight that some important effects might only become apparent once we train actual agents to act in the real world. Until then, our notion of effective

representations will, for better or worse, be guided by human-designed biases and tasks.

7. Selected Papers and Contributions

- **Karl Stelzner**, Robert Peharz, Kristian Kersting. Faster Attend-Infer-Repeat with Tractable Probabilistic Models. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019.

  I formulated the idea for this paper, implemented the method, and conducted the experiments. The text was written jointly by all three authors. Robert Peharz contributed some of the formalisms for communicating the method in a precise way.

- Jannik Kossen (*), **Karl Stelzner (*)**, Marcel Hussing, Claas Voelcker, Kristian Kersting. Structured Object-Aware Physics Prediction for Video Modeling and Planning. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 2020. (*) equal contribution.

  This paper resulted from a collaboration between me and my students Jannick Kossen, Marcel Hussing, and Claas Voelcker, whose Master's theses I was supervising at the time. I formulated the idea for the paper and core techniques such as the variational distribution and the object matching procedure. Building on my implementations of SuPAIR and interaction networks, Jannick Kossen implemented the STOVE model and ran most of the experiments. Marcel Hussing and Claas Voelcker contributed the reinforcement learning experiments. I wrote most of the paper's text and conducted some of the baseline experiments, Jannick Kossen created many of the figures, such as Figure 4.3. He also introduced the idea for the energy conservation experiments (Figure 4.7). Due to our close collaboration throughout this project, we agreed to share first authorship and to add the equal contribution notice.

- **Karl Stelzner**, Kristian Kersting, Adam R. Kosiorek. Decomposing 3D Scenes into Objects via Unsupervised Volume Segmentation. arXiv preprint *arXiv:2104.01148*.

The idea for this paper was the product of discussions between Adam Kosiorek and me. I implemented the code, conducted the experiments, and produced the figures. The text was written jointly by all three authors.

- **Karl Stelzner**, Kristian Kersting, Adam R. Kosiorek. Generative Adversarial Set Transformers. In *Working Notes of the ICML 2020 Workshop on Object-Oriented Learning (OOL)*. 2020.

  The idea for this paper comes from Adam Kosiorek. I implemented the model and ran the experiments. The text was written jointly by all three authors.

- Xiaoting Shao, Alejandro Molina, Antonio Vergari, **Karl Stelzner**, Robert Perharz, Thomas Liebig, Kristian Kersting. Conditional Sum-Product Networks: Imposing Structure on Deep Probabilistic Architectures. In *Proceedings of the 10th International Conference on Probabilistic Graphical Models (PGM)*. 2020.

  I was also involved in the writing and discussions for this paper, particularly in the conception of the neural CSPN idea. I also conducted the neural CSPN experiments.

- Robert Peharz, Antonio Vergari, **Karl Stelzner**, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, Zoubin Ghahramani. Random Sum-Product Networks: A Simple but Effective Approach to Probabilistic Deep Learning. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*. 2019.

  I was involved in the discussions and writing for this paper and did some of the data preparation.

- Robert Peharz, Steven Lang, Antonio Vergari, **Karl Stelzner**, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, Zoubin Ghahramani. Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*. 2020.

  I was involved in the discussions and writing for this paper and conducted some of the baseline experiments.

Bibliography

Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L. J. Learning representations and generative models for 3d point clouds. In *Proceedings of ICML*, 2018.

Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *The Journal of Machine Learning Research*, 22(1):4431–4506, 2021.

Albertz, J. A look back - 140 years of "photogrammetry" - some remarks on the history of photogrammetry. *Photogrammetric Engineering and Remote Sensing*, 73:504–506, 05 2007.

Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R. H., and Levine, S. Stochastic variational video prediction. In *Proceedings of ICLR*, 2018.

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

Barron, J. T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., and Srinivasan, P. P. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of ICCV*, 2021.

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D., and Kavukcuoglu, K. Interaction networks for learning about objects, relations and physics. In *Proceedings of NeurIPS*, 2016.

Bautista, M. A., Guo, P., Abnar, S., Talbott, W., Toshev, A., Chen, Z., Dinh, L., Zhai, S., Goh, H., Ulbricht, D., Dehghan, A., and Susskind, J. Gaudi: A neural architect for immersive 3d scene generation. In *Proceedings of NeurIPS*, 2022.

Bayer, J. and Osendorfer, C. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

Bayer, J., Soelch, M., Mirchev, A., Kayalibay, B., and van der Smagt, P. Mind the gap when conditioning amortised inference in sequential latent-variable models. In *Proceedings of ICLR*, 2021.

Becker-Ehmck, P., Peters, J., and Van Der Smagt, P. Switching linear dynamics for variational bayes filtering. In *Proceedings of ICML*, 2019.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828, 2013.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. URL `https://openai.com/five/`.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Blinn, J. F. Light reflection functions for simulation of clouds and dusty surfaces. In *Proceedings of SIGGRAPH*, 1982.

Brock, A., Donahue, J., and Simonyan, K. Large scale GAN training for high fidelity natural image synthesis. In *Proceedings of ICLR*, 2019.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Proceedings of NeurIPS*, 2020.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., and Zhang, Y. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Proceedings of NeurIPS*, pp. 8224–8234, 2018.

Burda, Y., Grosse, R. B., and Salakhutdinov, R. Importance weighted autoencoders. In *Proceedings of ICLR*, 2016. URL http://arxiv.org/abs/1509.00519.

Burgess, C. P., Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M., and Lerchner, A. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *Proceedings of ECCV*, 2020.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. Emerging properties in self-supervised vision transformers. In *Proceedings of ICCV*, 2021.

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. ShapeNet: An Information-Rich 3D Model Repository. In *arXiv preprint arXiv:1512.03012*, 2015.

Chang, J.-R. and Chen, Y.-S. Pyramid stereo matching network. In *Proceedings of CVPR*, 2018.

Chang, M., Ullman, T., Torralba, A., and Tenenbaum, J. B. A compositional object-based approach to learning physical dynamics. In *Proceedings of ICLR*, 2017.

Chao, Y.-W., Yang, W., Xiang, Y., Molchanov, P., Handa, A., Tremblay, J., Narang, Y. S., Van Wyk, K., Iqbal, U., Birchfield, S., Kautz, J., and Fox, D. DexYCB: A benchmark for capturing hand grasping of objects. In *Proceedings of CVPR*, 2021.

Chen, C., Deng, F., and Ahn, S. Roots: Object-centric representation and rendering of 3d scenes. In *arXiv preprint arXiv:2006.06130*, 2021.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *Proceedings of ICML*, 2020a.

Chen, Z., Yin, K., Fisher, M., Chaudhuri, S., and Zhang, H. Bae-net: Branched autoencoder for shape co-segmentation. In *Proceedings of ICCV*, 2019.

Chen, Z., Tagliasacchi, A., and Zhang, H. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of CVPR*, 2020b.

Choi, Y., Vergari, A., and den Broeck, G. V. Probabilistic circuits: A unifying framework for tractable probabilistic models, 2020. URL http://starai.cs.ucla.edu/papers/ProbCirc20.pdf.

Choy, C. B., Xu, D., Gwak, J., Chen, K., and Savarese, S. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proceedings of ECCV*, 2016.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Proceedings of NeurIPS*, pp. 4754–4765, 2018.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. A recurrent latent variable model for sequential data. In *Proceedings of NeurIPS*, 2015.

Clark, A. Whatever next? predictive brains, situated agents, and the future of cognitive science. *The Behavioral and brain sciences*, 36(3):181–204, 2013.

Coulom, R. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games*, pp. 72–83. Springer Berlin Heidelberg, 2007.

Crawford, E. and Pineau, J. Spatially invariant unsupervised object detection with convolutional neural networks. In *Proceedings of AAAI*, 2019.

Cremer, C., Li, X., and Duvenaud, D. Inference suboptimality in variational autoencoders. In *Proceedings of ICML*, volume 80, pp. 1078–1086, 2018.

Dasari, S., Ebert, F., Tian, S., Nair, S., Bucher, B., Schmeckpeper, K., Singh, S., Levine, S., and Finn, C. Robonet: Large-scale multi-robot learning. In *Proceedings of CoRL*, 2020.

Deng, B., Genova, K., Yazdani, S., Bouaziz, S., Hinton, G., and Tagliasacchi, A. Cvxnet: Learnable convex decomposition. In *Proceedings of CVPR*, 2020a.

Deng, F., Zhi, Z., Lee, D., and Ahn, S. Generative scene graph networks. In *Proceedings of ICLR*, 2020b.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Proceedings of CVPR*, 2009.

Deng, K., Liu, A., Zhu, J.-Y., and Ramanan, D. Depth-supervised nerf: Fewer views and faster training for free. In *arXiv preprint arXiv:2107.02791*, 2021.

Dhariwal, P. and Nichol, A. Diffusion models beat gans on image synthesis. In *Proceedings of NeurIPS*, 2021.

Ding, D., Hill, F., Santoro, A., Reynolds, M., and Botvinick, M. Attention over learned object embeddings enables complex visual reasoning. In *Proceedings of NeurIPS*, 2021.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of ICLR*, 2021.

Dumoulin, V., Shlens, J., and Kudlur, M. A learned representation for artistic style. In *Proceedings of ICLR*, 2017.

Ehrhardt, S., Monszpart, A., Mitra, N., and Vedaldi, A. Unsupervised intuitive physics from visual observations. In *Proceedings of ACCV*, pp. 700–716, 2018.

Ehrhardt, S., Groth, O., Monszpart, A., Engelcke, M., Posner, I., J. Mitra, N., and Vedaldi, A. Relate: Physically plausible multi-object scene synthesis using structured latent spaces. In *Proceedings of NeurIPS*, 2020.

Elich, C., Oswald, M. R., Pollefeys, M., and Stueckler, J. Semi-supervised learning of multi-object 3d scene representations. In *arXiv preprint arXiv:2010.04030*, 2021.

Elsayed, G. F., Mahendran, A., van Steenkiste, S., Greff, K., Mozer, M. C., and Kipf, T. SAVi++: Towards end-to-end object-centric learning from real-world videos. In *Proceedings of NeurIPS*, 2022.

Engelcke, M., Kosiorek, A. R., Jones, O. P., and Posner, I. Genesis: Generative scene inference and sampling with object-centric latent representations. In *Proceedings of ICLR*, 2020.

Engelcke, M., Parker Jones, O., and Posner, I. GENESIS-V2: Inferring Unordered Object Representations without Iterative Refinement. *Proceedings of NeurIPS*, 2021.

Eslami, S. A., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Hinton, G. E., et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Proceedings of NeurIPS*, pp. 3225–3233, 2016.

Eslami, S. A., Jimenez Rezende, D., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018.

Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis. In *Proceedings of CVPR*, 2021.

Fang, H.-S., Wang, C., Gou, M., and Lu, C. Graspnet-1billion: A large-scale benchmark for general object grasping. In *Proceedings of CVPR*, 2020.

Finn, C., Goodfellow, I., and Levine, S. Unsupervised learning for physical interaction through video prediction. *Proceedings of NeurIPS*, 29, 2016.

Foley, J. D., Van, F. D., Van Dam, A., Feiner, S. K., and Hughes, J. F. *Computer graphics: principles and practice*. Addison-Wesley Professional, 2013.

Fragkiadaki, K., Agrawal, P., Levine, S., and Malik, J. Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*, 2015.

Fridman, R., Abecasis, A., Kasten, Y., and Dekel, T. Scenescape: Text-driven consistent scene generation. *arXiv preprint arXiv:2302.01133*, 2023.

Gershman, S. J. and Goodman, N. D. Amortized inference in probabilistic reasoning. In *Proceedings of CogSci*, 2014.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Proceedings of NeurIPS*, pp. 2672–2680, 2014.

Greff, K., van Steenkiste, S., and Schmidhuber, J. Neural expectation maximization. In *Proceedings of NeurIPS*, pp. 6691–6701, 2017.

Greff, K., Kaufman, R. L., Kabra, R., Watters, N., Burgess, C., Zoran, D., Matthey, L., Botvinick, M., and Lerchner, A. Multi-object representation learning with iterative variational inference. In *Proceedings of ICML*, 2020a.

Greff, K., van Steenkiste, S., and Schmidhuber, J. On the binding problem in artificial neural networks. *arXiv preprint arXiv:2012.05208*, 2020b.

Gregor, K., Papamakarios, G., Besse, F., Buesing, L., and Weber, T. Temporal difference variational auto-encoder. In *Proceedings of ICLR*, 2019.

Grenander, U. *Lectures in Pattern Theory: Vol. 2 Pattern Analysis*. Springer-Verlag, 1976.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., et al. Bootstrap your own latent-a new approach to self-supervised learning. *Proceedings of NeurIPS*, 2020.

Gu, J., Trevithick, A., Lin, K.-E., Susskind, J., Theobalt, C., Liu, L., and Ramamoorthi, R. Nerfdiff: Single-image view synthesis with nerf-guided distillation from 3d-aware diffusion. *arXiv preprint arXiv:2302.10109*, 2023.

Guo, M., Fathi, A., Wu, J., and Funkhouser, T. Object-centric neural scene rendering. In *arXiv preprint arXiv:2012.08503*, 2020.

Ha, D. and Eck, D. A neural representation of sketch drawings. In *Proceedings of ICLR*, 2018.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Proceedings of NeurIPS*, 2018.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *Proceedings of ICML*, pp. 2555–2565, 2019.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. In *Proceedings of ICLR*, 2020.

Hafner, D., Lillicrap, T. P., Norouzi, M., and Ba, J. Mastering atari with discrete world models. In *Proceedings of ICLR*, 2021.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse domains through world models. *arXiv preprint 2301.04104*, 2023.

Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., and Cohen-Or, D. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

Hart, J. C. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527–545, 1996.

Hartley, R. I. and Zisserman, A. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of CVPR*, 2016.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. Masked autoencoders are scalable vision learners. In *Proceedings of CVPR*, 2022.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Proceedings of NeurIPS*, 2020.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Horaud, R., Hansard, M., Evangelidis, G., and Clément, M. An overview of depth cameras and range scanners based on time-of-flight technologies. *Machine Vision and Applications*, 27, 2016.

Hsieh, J.-T., Liu, B., Huang, D.-A., Fei-Fei, L. F., and Niebles, J. C. Learning to decompose and disentangle representations for video prediction. In *Proceedings of NeurIPS*, pp. 517–526, 2018.

Hubert, L. and Arabie, P. Comparing partitions. *Journal of Classification*, 1985.

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. Spatial transformer networks. In *Proceedings of NeurIPS*, pp. 2017–2025, 2015.

Janner, M., Levine, S., Freeman, W. T., Tenenbaum, J. B., Finn, C., and Wu, J. Reasoning about physical interactions with object-centric models. In *Proceedings of ICLR*, 2019.

Jaques, M., Burke, M., and Hospedales, T. Physics-as-inverse-graphics: Joint unsupervised learning of objects and physics from video. *arXiv preprint arXiv:1905.11169*, 2019.

Jiang, J., Janghorbani, S., De Melo, G., and Ahn, S. Scalor: Generative world models with scalable object representations. In *Proceedigns of ICLR*, 2020a.

Jiang, Y., Ji, D., Han, Z., and Zwicker, M. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *Proceedings of CVPR*, 2020b.

Johnson, J., Hariharan, B., Van Der Maaten, L., Fei-Fei, L., Lawrence Zitnick, C., and Girshick, R. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of CVPR*, 2017.

Johnson, M., Duvenaud, D. K., Wiltschko, A., Adams, R. P., and Datta, S. R. Composing graphical models with neural networks for structured representations and fast inference. In *Proceedings of NeurIPS*, pp. 2946–2954, 2016.

Kabra, R., Burgess, C., Matthey, L., Kaufman, R. L., Greff, K., Reynolds, M., and Lerchner, A. Multi-object datasets. `https://github.com/deepmind/multi_object_datasets/`, 2019.

Kabra, R., Zoran, D., Erdogan, G., Matthey, L., Creswell, A., Botvinick, M., Lerchner, A., and Burgess, C. P. SIMONe: View-invariant, temporally-abstracted object representations via unsupervised video decomposition. In *Proceedings of NeurIPS*, 2021.

Kaiser, Ł., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. Model based reinforcement learning for atari. In *Proceedings of ICLR*, 2020.

Kajiya, J. and von Herzen, B. Ray tracing volume densities. In *Proceedings of SIGGRAPH*, 1984.

Kalashnikov, D., Varley, J., Chebotar, Y., Swanson, B., Jonschkowski, R., Finn, C., Levine, S., and Hausman, K. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

Kania, K., Zięba, M., and Kajdanowicz, T. Ucsg-net – unsupervised discovering of constructive solid geometry tree. In *Proceedings of NeurIPS*, 2020.

Karl, M., Soelch, M., Bayer, J., and van der Smagt, P. Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data. In *Proceedings of ICLR*, 2017.

Karras, T., Aittala, M., Laine, S., Härkönen, E., Hellsten, J., Lehtinen, J., and Aila, T. Alias-free generative adversarial networks. In *Proceedings of NeurIPS*, 2021.

Kato, H., Ushiku, Y., and Harada, T. Neural 3d mesh renderer. In *Proceedings of CVPR*, 2018.

Kingma, D., Salimans, T., Poole, B., and Ho, J. Variational diffusion models. In *Proceedings of NeurIPS*, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of ICLR*, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *Proceedings of ICLR*, 2014.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. In *Proceedings of ICML*, 2018.

Kipf, T., van der Pol, E., and Welling, M. Contrastive learning of structured world models. In *Proceedings of ICLR*, 2020.

Kipf, T., Elsayed, G. F., Mahendran, A., Stone, A., Sabour, S., Heigold, G., Jonschkowski, R., Dosovitskiy, A., and Greff, K. Conditional Object-Centric Learning from Video. In *Proceedings of ICLR*, 2022.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *Proceedings of ICLR*, 2017.

Kleeberger, K., Bormann, R., Kraus, W., and Huber, M. F. A survey on learning-based robotic grasping. *Current Robotics Reports*, 1:239–249, 2020.

Kornmeier, J. and Bach, M. Ambiguous figures – what happens in the brain when perception changes but not the stimulus. *Frontiers in Human Neuroscience*, 6, 2012.

Kosiorek, A., Kim, H., Teh, Y. W., and Posner, I. Sequential attend, infer, repeat: Generative modelling of moving objects. In *Proceedings of NeurIPS*, pp. 8606–8616, 2018.

Kosiorek, A. R. Geometry in text-to-image diffusion models, 2023. URL `https://akosiorek.github.io/geometry_in_image_diffusion/`.

Kosiorek, A. R., Kim, H., and Rezende, D. J. Conditional set generation with transformers. *arXiv preprint arXiv:2006.16841*, 2020.

Kosiorek, A. R., Strathmann, H., Zoran, D., Moreno, P., Schneider, R., Mokrá, S., and Rezende, D. J. Nerf-vae: A geometry aware 3d scene generative model. In *arXiv preprint arXiv:2104.00587*, 2021.

Kossen, J. Modelling videos of physically interacting objects. *Master Thesis (Heidelberg University)*, 2020.

Kossen, J., Stelzner, K., Hussing, M., Voelcker, C., and Kersting, K. Structured object-aware physics prediction for video modeling and planning. In *Proceedings of ICLR*, 2020.

Krishnan, R., Shalit, U., and Sontag, D. Structured inference networks for nonlinear state space models. In *Proceedings of AAAI*, 2017.

Krishnan, R. G., Shalit, U., and Sontag, D. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Proceedings of NeurIPS*, 2012.

Kuhn, H. W. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Laine, S. and Karras, T. Efficient sparse voxel octrees. In *Proceedings of SIGGRAPH*, 2010.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

LeCun, Y., Bengio, Y., et al. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 1995.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521:436–44, 05 2015.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of ICML*, 2019.

Lempitsky, V. and Zisserman, A. Learning to count objects in images. In *Proceedings of NeurIPS*, pp. 1324–1332, 2010.

Li, Y., Wang, G., Ji, X., Xiang, Y., and Fox, D. Deepim: Deep iterative matching for 6d pose estimation. In *Proceedings of the ECCV*, 2018.

Li, Y., Wu, J., Tedrake, R., Tenenbaum, J. B., and Torralba, A. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *Proceedings of ICLR*, 2019.

Lin, H., Peng, S., Xu, Z., Bao, H., and Zhou, X. Efficient neural radiance fields with learned depth-guided sampling. *arXiv preprint arXiv:2112.01517*, 2021.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *Proceedings of ECCV*, 2014.

Lin, Z., Wu, Y.-F., Peri, S. V., Sun, W., Singh, G., Deng, F., Jiang, J., and Ahn, S. Space: Unsupervised object-oriented scene representation via spatial attention and decomposition. In *Proceedings of ICLR*, 2020.

Liu, A., Zhang, H., and den Broeck, G. V. Scaling up probabilistic circuits by latent variable distillation. In *Proceedings of ICLR*, 2023.

Liu, L., Gu, J., Zaw Lin, K., Chua, T.-S., and Theobalt, C. Neural sparse voxel fields. *Proceedings of NeurIPS*, 33:15651–15663, 2020a.

Liu, S., Zhang, Y., Peng, S., Shi, B., Pollefeys, M., and Cui, Z. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *Proceedings of CVPR*, 2020b.

Liu, S., Li, T., Chen, W., and Li, H. A general differentiable mesh renderer for image-based 3d reasoning. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2022.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention. In *Proceedings of NeurIPS*, 2020.

Loper, M. M. and Black, M. J. Opendr: An approximate differentiable renderer. In *Proceedings of ECCV*, 2014.

Lorensen, W. E. and Cline, H. E. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

MacKay, D. J. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 354(1):73–80, 1995.

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *Proceedings of ICLR*, 2019.

Martin-Brualla, R., Radwan, N., Sajjadi, M. S. M., Barron, J. T., Dosovitskiy, A., and Duckworth, D. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of CVPR*, 2021.

Melas-Kyriazi, L., Rupprecht, C., Laina, I., and Vedaldi, A. Realfusion: 360° reconstruction of any object from a single image. *arXiv preprint arXiv:2302.10663*, 2023.

Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of CVPR*, 2019.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of ECCV*, 2020.

Mnih, A. and Gregor, K. Neural variational inference and learning in belief networks. In *Proceedings of ICML*, 2014.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Molina, A., Vergari, A., Di Mauro, N., Natarajan, S., Esposito, F., and Kersting, K. Mixed sum-product networks: A deep architecture for hybrid domains. In *Proceedings of AAAI*, 2018.

Murphy, K. P. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. URL `http://probml.github.io/book2`.

Møller, J. and Waagepetersen, R. P. *Statistical inference and simulation for spatial point processes*. CRC Press, 2003.

Nanbo, L., Eastwood, C., and Fisher, R. B. Learning object-centric representations of multi-object scenes from multiple views. In *Proceedings of NeurIPS*, 2020.

Nash, C., Ganin, Y., Eslami, S. A., and Battaglia, P. Polygen: An autoregressive generative model of 3d meshes. In *Proceedings of ICML*, 2020.

Nematpur, I. Inferring high-fidelity object-oriented scene functions using pixel conditioning. *Master Thesis (TU Darmstadt)*, 2022.

Nguyen-Phuoc, T., Richardt, C., Mai, L., Yang, Y.-L., and Mitra, N. Blockgan: Learning 3d object-aware scene representations from unlabelled images. In *Proceedings of NeurIPS*, 2020.

Niemeyer, M. and Geiger, A. Giraffe: Representing scenes as compositional generative neural feature fields. In *arXiv preprint arXiv:2011.12100*, 2020.

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. Action-conditional video prediction using deep networks in atari games. In *Proceedings of NeurIPS*, pp. 2845–2853, 2015.

Oquab, M., Darcet, T., Moutakanni, T., Vo, H. V., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., Howes, R., Huang, P.-Y., Xu, H., Sharma, V., Li, S.-W., Galuba, W., Rabbat, M., Assran, M., Ballas, N., Synnaeve, G., Misra, I., Jegou, H., Mairal, J., Labatut, P., Joulin, A., and Bojanowski, P. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Ost, J., Mannan, F., Thuerey, N., Knodt, J., and Heide, F. Neural scene graphs for dynamic scenes. In *arXiv preprint arXiv: 2011.10379*, 2021.

Peharz, R., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Kersting, K., and Ghahramani, Z. Probabilistic deep learning using random sum-product networks. In *Proceedings of UAI*, 2019.

Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van den Broeck, G., Kersting, K., and Ghahramani, Z. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *Proceedings of ICML*, 2020.

Poole, B., Jain, A., Barron, J. T., and Mildenhall, B. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

Poon, H. and Domingos, P. Sum-product networks: A new deep architecture. In *Proceedings of UAI*, pp. 337–346, 2011.

Porter, T. and Duff, T. Compositing digital images. In *Proceedings of SIGGRAPH*, 1984.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of CVPR*, 2017a.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Proceedings of NeurIPS*, 30, 2017b.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *Proceedings of ICML*, 2021.

Rainforth, T., Cornish, R., Yang, H., and Warrington, A. On nesting monte carlo estimators. In *Proceedings of ICML*, 2018a.

Rainforth, T., Kosiorek, A., Le, T. A., Maddison, C., Igl, M., Wood, F., and Teh, Y. W. Tighter variational bounds are not necessarily better. In *Proceedings of ICML*, volume 80, pp. 4277–4285, 2018b.

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Rand, W. M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 1971.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of ICLR*, 2014.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. *arXiv preprint arXiv:2112.10752*, 2021a.

Rombach, R., Esser, P., and Ommer, B. Geometry-free view synthesis: Transformers and no 3d priors. *Proceedings of ICCV*, 2021b.

Sajjadi, M. S. M., Duckworth, D., Mahendran, A., van Steenkiste, S., Pavetić, F., Lučić, M., Guibas, L. J., Greff, K., and Kipf, T. Object Scene Representation Transformer. In *Proceedings of NeurIPS*, 2022a.

Sajjadi, M. S. M., Mahendran, A., Kipf, T., Pot, E., Duckworth, D., Lučić, M., and Greff, K. Rust: Latent neural scene representations from unposed imagery. *arXiv preprint arXiv:2211.14306*, 2022b.

Sajjadi, M. S. M., Meyer, H., Pot, E., Bergmann, U., Greff, K., Radwan, N., Vora, S., Lucic, M., Duckworth, D., Dosovitskiy, A., Uszkoreit, J., Funkhouser, T., and Tagliasacchi, A. Scene Representation Transformer: Geometry-Free Novel View Synthesis Through Set-Latent Scene Representations. In *Proceedings of CVPR*, 2022c.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. Graph networks as learnable physics engines for inference and control. In *Proceedings of ICML*, 2018.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *Proceedings of ICML*, 2020.

Särkkä, S. *Bayesian filtering and smoothing*. Number 3. Cambridge university press, 2013.

Schönberger, J. L., Zheng, E., Pollefeys, M., and Frahm, J.-M. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

Schuhmann, C., Beaumont, R., Vencu, R., Gordon, C., Wightman, R., Cherti, M., Coombes, T., Katta, A., Mullis, C., Wortsman, M., Schramowski, P., Kundurthy, S., Crowson, K., Schmidt, L., Kaczmarczyk, R., and Jitsev, J. Laion-5b: An open large-scale dataset for training next generation image-text models. In *Proceedings of NeuIPS, Track on Datasets and Benchmarks*, 2022.

Schulman, J., Heess, N., Weber, T., and Abbeel, P. Gradient estimation using stochastic computation graphs. In *Proceedings of NeurIPS*, pp. 3528–3536, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sevilla, J., Heim, L., Ho, A., Besiroglu, T., Hobbhahn, M., and Villalobos, P. Compute trends across three eras of machine learning. *arXiv preprint arXiv:2202.05924*, 2022.

Shi, Y., Siddharth, N., Torr, P. H., and Kosiorek, A. R. Adversarial masking for self-supervised learning. In *Proceedings of ICML*, 2022.

Shlomi, J., Battaglia, P., and Vlimant, J.-R. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.

Shu, D. W., Park, S. W., and Kwon, J. 3d point cloud generative adversarial network based on tree structured graph convolutions. In *Proceedings of ICCV*, 2019.

Singer, U., Sheynin, S., Polyak, A., Ashual, O., Makarov, I., Kokkinos, F., Goyal, N., Vedaldi, A., Parikh, D., Johnson, J., and Taigman, Y. Text-to-4d dynamic scene generation. *arXiv preprint arXiv:2301.11280*, 2023.

Sitzmann, V., Zollhöfer, M., and Wetzstein, G. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Proceedings of NeurIPS*, 2019.

Sitzmann, V., Rezchikov, S., Freeman, B., Tenenbaum, J., and Durand, F. Light field networks: Neural scene representations with single-evaluation rendering. *Proceedings of NeurIPS*, 2021.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. In *Proceedings of NeurIPS*, 2019.

Stelzner, K., Peharz, R., and Kersting, K. Faster attend-infer-repeat with tractable probabilistic models. In *Proceedings of ICML*, 2019.

Stelzner, K., Kersting, K., and Kosiorek, A. R. Generative adversarial set transformers. In *Workshop on Object-Oriented Learning at ICML*, 2020.

Stelzner, K., Kersting, K., and Kosiorek, A. R. Decomposing 3d scenes into objects via unsupervised volume segmentation. *arXiv preprint arXiv:2104.01148*, 2021.

Stone, A., Maurer, D., Ayvaci, A., Angelova, A., and Jonschkowski, R. Smurf: Self-teaching multi-frame unsupervised raft with full-image warping. In *Proceedings of CVPR*, 2021.

Sun, C., Shrivastava, A., Vondrick, C., Murphy, K., Sukthankar, R., and Schmid, C. Actor-centric relation network. In *Proceedings of ECCV*, 2018.

Sun, C., Shrivastava, A., Vondrick, C., Sukthankar, R., Murphy, K., and Schmid, C. Relational action forecasting. In *Proceedings of CVPR*, 2019.

Sutton, R. The bitter lesson, 2019. URL `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`.

Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT Press, 2018.

Teed, Z. and Deng, J. Deepv2d: Video to depth with differentiable structure from motion. In *Proceedings of ICLR*, 2020.

Titsias, M. K. and Gredilla-Lázaro, M. Doubly stochastic variational Bayes for non-conjugate inference. In *Proceedings of ICML*, pp. 1971–1979, 2014.

Trevithick, A. and Yang, B. Grf: Learning a general radiance field for 3d scene representation and rendering. In *arXiv preprint arXiv:2010.04595*, 2020.

Valsesia, D., Fracastoro, G., and Magli, E. Learning localized generative models for 3d point clouds via graph convolution. In *Proceedings of ICLR*, 2019.

van Steenkiste, S., Chang, M., Greff, K., and Schmidhuber, J. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. In *Proceedings of ICLR*, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Proceedings of NeurIPS*, 2017.

Veerapaneni, R., Co-Reyes, J. D., Chang, M., Janner, M., Finn, C., Wu, J., Tenenbaum, J., and Levine, S. Entity abstraction in visual model-based reinforcement learning. In *Proceedings of CORL*, 2020.

Vergari, A., Peharz, R., Di Mauro, N., Molina, A., Kersting, K., and Esposito, F. Sum-product autoencoding: Encoding and decoding representations using sum-product networks. In *Proceedings of AAAI*, 2018.

Vicini, D., Speierer, S., and Jakob, W. Differentiable signed distance function rendering. *ACM Transactions on Graphics (TOG)*, 41(4):1–18, 2022.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Wang, N., Zhang, Y., Li, Z., Fu, Y., Liu, W., and Jiang, Y.-G. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of ECCV*, pp. 52–67, 2018a.

Wang, T., Liao, R., Ba, J., and Fidler, S. Nervenet: Learning structured policy with graph neural networks. In *Proceedings of ICLR*, 2018b.

Wang, T., Zhang, B., Zhang, T., Gu, S., Bao, J., Baltrusaitis, T., Shen, J., Chen, D., Wen, F., Chen, Q., and Guo, B. Rodin: A generative model for sculpting 3d digital avatars using diffusion. *arXiv preprint arXiv:2212.06135*, 2022.

Watson, D., Chan, W., Brualla, R. M., Ho, J., Tagliasacchi, A., and Norouzi, M. Novel view synthesis with diffusion models. In *Proceedings of ICLR*, 2023.

Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A. Visual interaction networks: Learning a physics simulator from video. In *Proceedings of NeurIPS*, pp. 4539–4547, 2017.

Watters, N., Matthey, L., Bosnjak, M., Burgess, C. P., and Lerchner, A. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*, 2019a.

Watters, N., Matthey, L., Burgess, C. P., and Lerchner, A. Spatial broadcast decoder: A simple architecture for disentangled representations in VAEs. In *Proceedings of ICLR Workshop on Learning from Limited Labeled Data*, 2019b.

Weis, M. A., Chitta, K., Sharma, Y., Brendel, W., Bethge, M., Geiger, A., and Ecker, A. S. Benchmarking unsupervised object representations for video sequences. *The Journal of Machine Learning Research*, 22(1):8253–8313, 2021.

Wu, J., Yildirim, I., Lim, J. J., Freeman, B., and Tenenbaum, J. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Proceedings of NeurIPS*, pp. 127–135, 2015.

Wu, J., Lim, J. J., Zhang, H., Tenenbaum, J. B., and Freeman, W. T. Physics 101: Learning physical object properties from unlabeled videos. In *Proceedings of BMVC*, 2016.

Wu, J., Wang, Y., Xue, T., Sun, X., Freeman, B., and Tenenbaum, J. Marrnet: 3d shape reconstruction via 2.5 d sketches. *Proceedings of NeurIPS*, 2017.

Wu, P., Escontrela, A., Hafner, D., Abbeel, P., and Goldberg, K. Daydreamer: World models for physical robot learning. In *Proceedings of CoRL*, 2022.

Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *Proceedings of RSS*, 2018.

Xu, Z., Liu, Z., Sun, C., Murphy, K., Freeman, W. T., Tenenbaum, J. B., and Wu, J. Modeling parts, structure, and system dynamics via predictive learning. In *Proceedings of ICLR*, 2019.

Yan, X., Yang, J., Yumer, E., Guo, Y., and Lee, H. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. *Proceedings of NeurIPS*, 29, 2016.

Yu, A., Ye, V., Tancik, M., and Kanazawa, A. pixelNeRF: Neural radiance fields from one or few images. In *Proceedings of CVPR*, 2021a.

Yu, H.-X., Guibas, L. J., and Wu, J. Unsupervised discovery of object radiance fields. In *arXiv preprint arXiv:2107.07905*, 2021b.

Yurtsever, E., Lambert, J., Carballo, A., and Takeda, K. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Proceedings of NeurIPS*, 2017.

Zakharov, S., Shugurov, I., and Ilic, S. Dpod: 6d pose object detector and refiner. In *Proceedings of ICCV*, 2019.

Zhang, X., Bi, S., Sunkavalli, K., Su, H., and Xu, Z. Nerfusion: Fusing radiance fields for large-scale scene reconstruction. In *Proceedings of CVPR*, 2022.

Zhang, Y., Hare, J., and Prugel-Bennett, A. Deep set prediction networks. *Proceedings of NeurIPS*, 2019.

Zhao, C., Zhang, Y., Poggi, M., Tosi, F., Guo, X., Zhu, Z., Huang, G., Tang, Y., and Mattoccia, S. Monovit: Self-supervised monocular depth estimation with a vision transformer. In *Proceedings of the International Conference on 3D Vision*, 2022.

Zhao, W., Queralta, J. P., and Westerlund, T. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pp. 737–744. IEEE, 2020.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

A. Details on STOVE

Here, we provide details on the STOVE model introduced in 4.4. The material presented here has been published in Kossen et al. (2020).

A.1. Object Reconstructions on Sprites Data

Here, we explore STOVE's ability to handle heterogeneous objects. SuPAIR does not to infer a latent description of the objects' appearances. Nevertheless, object reconstructions can be obtained by using a variant of approximate MPE (most probable explanation) in the sum-product networks as proposed by Vergari et al. (2018).

To demonstrate the capabilities of our image model, we train our model on a variant of the gravity data in which the round balls were replaced by a random selection of four different sprites of the same size. Fig. A.1 shows the reconstructions obtained from SuPAIR when trained on these more complex object shapes.

A.2. Generalization to Different Energy Levels

As discussed in Sec. 4.4.4, the energies of the ground truth data were constant for all sequences during the training of STOVE. However, initial velocities are drawn from a random normal distribution. This is the standard procedure of generating the bouncing balls data set as used by previous publications. Under these circumstances, STOVE does indeed learn to discover and replicate the total energies of the system, while SQAIR and DDPAE do not.

Here, we study STOVE's ability to generalize to other energy levels. Even if trained on constant energy data, STOVE does to some extent generalise to unseen energies.

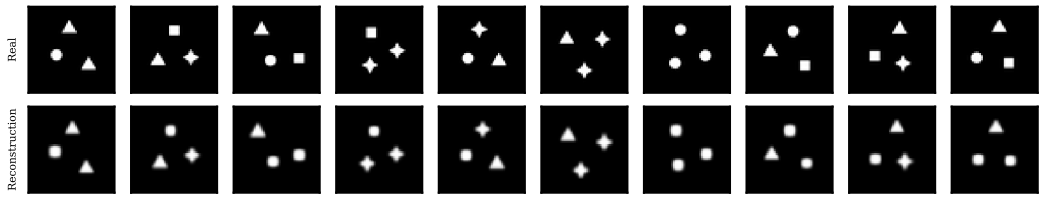

Figure A.1.: Reconstructions obtained from SuPAIR when using more varied shapes.

Velocities and therefore total energies which are inferred for the observed frames are highly correlated with the true total kinetic energies of the sequences. However, as prediction starts, STOVE quickly regresses to the energy of the training set, see Fig. A.2 (left). If trained on a dataset of diverse total energies, the performance of modeling sequences of different energies increases, see Fig. A.2 (right). Rollouts now initially represent the true energy of the observed sequence, although this estimate of the true energy diverges over a time span of around 500 frames to a constant but wrong energy value. This is an improvement over the model trained on constant energy data, where the regression to the training data energy happens much quicker within around 10 frames. Note that this slower regression does not drastically decrease the visual quality of the rollouts as the change of total energy over 500 frames is gradual enough to be convincing. We leave the reliable prediction of rollouts with physically valid constant energy for sequences of varying energies for future work.

A.3. Model Details

Here, we present additional details on the architecture and hyperparameters of STOVE.

A.3.1. Inference Architecture

The object detection network for $q(z_{t,\text{where}} \mid x_t)$ is realised by an LSTM (Hochreiter & Schmidhuber, 1997) with 256 hidden units, which outputs the mean and standard deviation of the objects' two-dimensional position and size distributions, i.e. $q(z^o_{t,\text{pos, size}} \mid x_t)$ with $2 \cdot 2 \cdot 2 = 8$ parameters per object. Given such position distributions for two consecutive timesteps $q(z_{t-1,\text{pos}} \mid x_{t-1}), q(z_{t,\text{pos}} \mid x_t)$, with parameters

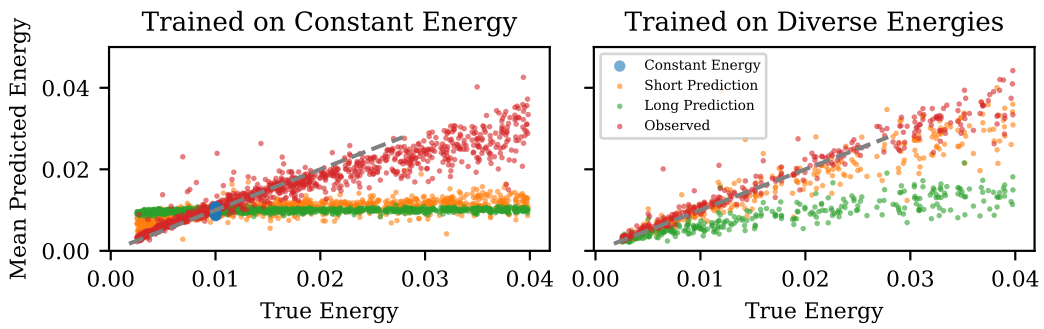

Figure A.2.: Mean kinetic energy observed/predicted by STOVE over true energy of the sequences. (left) STOVE is trained on sequences of constant kinetic energy. As can be seen from the blue scatter points, STOVE manages to predict sequences of arbitrary lengths which, on average, preserve the constant energy of the test set. When STOVE is applied to sequences of different energies, it manages to infer these energies from observed frames fairly well, with inaccuracies compounding at larger energies (red). In the following prediction, however, the mean predicted energies diverge quickly to the energy value of the training set (orange and green). (right) STOVE is now trained on sequences of varying energies. Compared to the constant energy training, energies from observed as well as predicted energies improve drastically. The predictions no longer immediately regress towards a specific value (orange). However after 100 frames, the quality of the predicted energies still regresses to a wrong value (green). (all) The observed values refers to energies obtained as the mean energy value over the six initially observed frames. The short (long) time frame refers to an energy obtained as the mean energy over the first 10 (100) frames of prediction.

$\mu_{z^o_{t-1,\text{pos}}}, \sigma_{z^o_{t-1,\text{pos}}}, \mu_{z^o_{t,\text{pos}}}, \sigma_{z^o_{t,\text{pos}}}$, the following velocity estimate based on the difference in position is constructed:

$$q(z^o_{t,\text{velo}} \mid x_t, x_{t-1}) = \mathcal{N}(\mu_{z^o_{t,\text{pos}}} - \mu_{z^o_{t-1,\text{pos}}}, \sigma^2_{z^o_{t,\text{pos}}} + \sigma^2_{z^o_{t-1,\text{pos}}}).$$

As described in Sec. 4.4.2, positions and velocities are also inferred from the dynamics model as $q(z^o_{t,\text{pos}} \mid z_{t-1})$ and $q(z^o_{t,\text{velo}} \mid z_{t-1})$. A joint estimate, including information from

both image model and dynamics prediction, is obtained by multiplying the respective distributions and renormalizing. Since both q-distributions are Gaussian, the normalized product is again Gaussian, with mean and standard deviation are given by

$$\begin{aligned} q(z_t \mid x_t, z_{t-1}) &\propto q(z_t \mid x_t) \cdot q(z_t \mid z_{t-1}) \\ &= \mathcal{N}(z_t; \mu_{t,i},\, \sigma_{t,i}^2) \cdot \mathcal{N}(z_t; \mu_{t,d},\, \sigma_{t,d}^2) \\ &= \mathcal{N}(z_t; \mu_t,\, \sigma_t^2) \\ \mu_t &= \frac{\sigma_{t,d}^2\, \mu_{t,i} + \sigma_{t,i}^2\, \mu_{t,d}}{\sigma_{t,d}^2 + \sigma_{t,i}^2} \\ \frac{1}{\sigma_t^2} &= \frac{1}{\sigma_{t,d}^2} + \frac{1}{\sigma_{t,i}^2}\,, \end{aligned}$$

where we relax our notation for readability $z_t \in [z^o_{t,\text{pos}}, z^o_{t,\text{velo}}]$ and the indices i and d refer to the parameters obtained from the image and dynamics model. This procedure is applied independently for the positions and velocities of each object.

For $z^o_{t,\text{latent}}$, we choose dimension 12, such that a full state $z^o_t = (z^o_{t,\text{pos}}, z^o_{t,\text{size}}, z^o_{t,\text{velo}}, z^o_{t,\text{latent}})$ is 18-dimensional.

A.3.2. Dynamics Network

The dynamics prediction is given by the following series of transformations applied to each input state of shape $(\texttt{batch size}, \texttt{number of objects}, l)$, where $l = 16$, since currently, size information is not propagated through the dynamics prediction.

- S_1: Encode input state with linear layer $[l, 2l]$.
- S_2: Apply linear layer $[2l, 2l]$ to S_1 followed by ReLU non-linearity.
- S_3: Apply linear layer $[2l, 2l]$ to S_2 and add result to S_2. This gives the dynamics prediction without relational effects, corresponding to $g(z^o_t)$ in Eq. 4.6.
- C_1: The following steps obtain the relational aspects of dynamics prediction, corresponding to $h(z^o_t, z^{o'}_t)$ in Eq. 4.6. Concatenate the encoded state S^o_1 pairwise with all state encoding, yielding a tensor of shape $(\texttt{batch size}, \texttt{number of objects}, \texttt{number of objects}, 4l)$.
- C_2: Apply linear layer $[4l, 4l]$ to C_1 followed by ReLU.

- C_3: Apply linear layer $[4l, 2l]$ to C_2 followed by ReLU.
- C_4: Apply linear layer $[2l, 2l]$ to C_3 and add to C_3.
- A_1: To obtain attention coefficients $\alpha(z_t^o, z_t^{o'})$, apply linear layer $[4l, 4l]$ to C_1 followed by ReLU.
- A_2: Apply linear layer $[4l, 2l]$ to A_1 followed by ReLU.
- A_3: Apply linear layer $[2l, 1]$ to A_2 and apply exponential function.
- R_1: Multiply C_4 with A_3, where diagonal elements of A_3 are masked out to ensure that R_1 only covers cases where $o \neq o'$.
- R_2: Sum over R_1 for all o', to obtain tensor of shape (`batch size`, `number of objects`, $2l$). This is the relational dynamics prediction.
- D_1: Sum relational dynamics R_2 and self-dynamics S_3, obtaining the input to f in Eq. 4.6.
- D_2: Apply linear layer $[2l, 2l]$ to D_1 followed by tanh non-linearity.
- D_3: Apply linear layer $[2l, 2l]$ to D_2 followed by tanh non-linearity and add result to D_2.
- D_4: Concatenate D_3 and S_1, and apply linear layer $[4l, 2l]$ followed by tanh.
- D_5: Apply linear layer $[2l, 2l]$ to D_4 and add result to D_4 to obtain final dynamics prediction.

The output D_5 has shape (`batch size`, `number of objects`, $2l$), twice the size of means and standard deviations over the next predicted state.

For the model-based control scenario, the one-hot encoded actions (`batch size`, `action space`) are transformed with a linear layer [`action space`, `number of objects` $\cdot$ `encoding size`] and reshaped to (`action space`, `number of objects`, `encoding size`). The action embedding and the object appearances (`batch size`, `number of objects`, 3) are then concatenated to the input state. The rest of the dynamics prediction follows as above. The reward prediction consists of the following steps:

- H_1: Apply linear layer $[2l, 2l]$ to D_1 followed by ReLU.
- H_2: Apply linear layer $[2l, 2l]$ to H_1.
- H_3: Sum over object dimension to obtain tensor of shape (`batch size`, l).

- H_4: Apply linear layer $[l, l/2]$ to H_3 followed by ReLU.
- H_5: Apply linear layer $[l/2, l/4]$ to H_4 followed by ReLU.
- H_5: Apply linear layer $[l/4, l]$ to H_4 followed by a sigmoid non-linearity.

H_5 then gives the final reward prediction.

A.3.3. State Initialization

In the first two timesteps, we cannot yet apply STOVE's main inference step $q(z_t \mid z_{t-1}, x_t, x_{t-1})$ as described above. In order to initialize the latent state over the first two frames, we apply a simplified architecture and only use a partial state at $t = 0$.

At $t = 0$, $z_0 \sim q(z_{0,(\text{pos, size})} \mid x_0)$ is given purely by the object detection network, since no previous states, which could be propagated, exist. z_0 is incomplete insofar as it does not contain velocity information or latents. At $t = 1$, $q(z_{1,\text{pos, size}} \mid x_1, x_0)$ is still given purely based on the object detection network. Note that for a dynamics prediction of z_1, velocity information at $t = 0$ would need to be available. However, at $t = 1$, velocities can be constructed based on the differences between the previously inferred object positions. We sample $z_{1,\text{latent}}$ from the prior Gaussian distribution to assemble the first full initial state z_1. At $t \geq 2$, the full inference network can be run: States are inferred both from the object detection network $q(z_t \mid x_t, x_{t-1})$ as well as propagated using the dynamics model $q(z_t \mid z_{t-1})$.

In the generative model, similar adjustments are made: $p(z_{0,\text{pos, size}})$ is given by a uniform prior, velocities and latents are omitted. At $t = 1$, velocities are sampled from a uniform distribution in planar coordinates $p(z_{1,\text{velo}})$ and positions are given by a simple linear dynamics model $p(z_{1,\text{pos}} \mid z_{0,\text{pos}}, z_{1,\text{velo}}) = \mathcal{N}(z_{0,\text{pos}} + z_{1,\text{velo}}, \sigma)$. Latents $z_{1,\text{latent}}$ are sampled from a Gaussian prior. Starting at $t = 2$, the full dynamics model is used.

A.3.4. Training Procedure

Our model was trained using the Adam optimizer (Kingma & Ba, 2015), with a learning rate of $2 \times 10^{-3} \exp(-40 \times 10^{-3} \cdot \text{step})$ for a total of $83\,000$ steps with a batch size of 256.

A.4. Data Details

For the billiards and gravitational data, 1000 sequences of length 100 were generated for training. From these, subsequences of lengths 8 were sampled and used to optimize the ELBO. A test dataset of 300 sequences of length 100 was also generated and used for all evaluations. The pixel resolution of the dataset was 32×32 for the billiards data and 50×50 for the gravity data. All models for video prediction were learned on grayscale data, with objects of identical appearance. The $O = 3$ balls were initialised with uniformly random positions and velocities, rejecting configurations with overlap. They are rendered using anti-aliasing. The billiards data models the balls as circular objects, which perform elastic collision with each other or the walls of the environment. For the gravity data, the balls are modeled as point masses, where, following Watters et al. (2017), we clip the gravitational force to avoid slingshot effects. Also, we add an additional basin of attraction towards the center of the canvas and model the balls in their center off mass system to avoid drift. Velocities here are initialised orthogonal to the center of the canvas for a stabilising effect. For full details we refer to the file `envs.py` in the provided code.

A.5. Baselines for Video Modeling

Following Kosiorek et al. (2018), we experimented with different hyperparameter configurations for VRNNs. We varied the sizes of the hidden and latent states $[h, z]$, experimenting with the values $[256, 16]$, $[512, 32]$, $[1024, 64]$, and $[2048, 32]$. We found that increasing the model capacity beyond $[512, 32]$ did not yield large increases in performance, which is why we chose the configuration $[512, 32]$ for our experiments. Our VRNN implementation is written in PyTorch and based on `https://github.com/emited/VariationalRecurrentNeuralNetwork`.

SQAIR can handle a variable number of objects in each sequence. However, to allow for a fairer comparison to STOVE, we fixed the number of objects to the correct number. This means that in the first timestep, exactly three objects are discovered, which are then propagated in all following timesteps, without further discoveries. Our implementation is based on the original implementation provided by the authors at `https://github.com/akosiorek/sqair`.

The DDPAE experiments were performed using the implementation available at `https://github.com/jthsieh/DDPAE-video-prediction`. Default parameters for

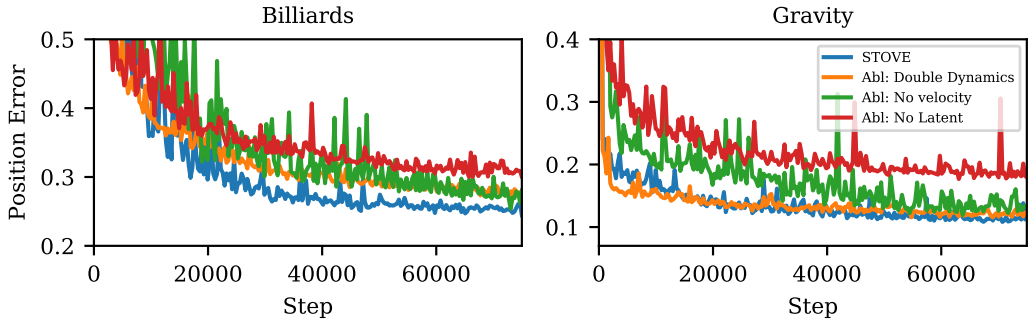

Figure A.3.: Displayed is the mean predicted position error over a rollout length of 8 frames as training progresses for the billiards (left) and gravity (right) scenario for STOVE and its ablations.

training DDPAE with billiards datasets are provided with the code. However, the resolution of our billiards (32 pixels) and gravity (64 pixels) datasets is different to the resolution DDPAE expects (64 pixels). While we experimented with adjusting DDPAE parameters such as the latent space dimension to fit our different resolution, best results were obtained when bilinearly scaling our data to the resolution DDPAE expects. DDPAE was trained for $400\,000$ steps, which sufficed for convergence of the models' test set error.

The linear baseline was obtained as follows: For the first 8 frames, we infer the full model state using STOVE. We then take the last inferred positions and velocities of each object and predict future positions by assuming constant, uniform motions for each object. We do not allow objects to leave the frame, i. e. when objects reach the canvas boundary after some timesteps, they stick to it.

Since our dynamics model requires only object positions and velocities as input, it is trivial to construct a supervised baseline for our physics prediction by replacing the SuPAIR-inferred states with real, ground-truth states. On these, the model can then be trained in supervised fashion.

A.6. Training Curves of Ablations

In Fig. A.3 we display learning curves for STOVE and presented ablations. As mentioned in the main text, the ablations demonstrate the value of the reuse of the dynamics model, the explicit inclusion of a velocity value, and the presence of unstructured latent space in the dynamics model.

A.7. Details on the Reinforcement Learning Models

Our MCTS implementation uses the standard UCT formulation for exploration/exploitation. The c parameter is set to $1.$ in all our experiments. Since the environment does not provide a natural endpoint, we cut off all rollouts at a depth of 20 timesteps. We found this to be a good trade-off between runtime and accuracy.

When expanding a node on the true environment, we compute the result of the most promising action, and then start a rollout using a random policy from the resulting state. For the final evaluation, a total of 200 nodes are expanded. To better utilize the GPU, a slightly different approach is used for STOVE. When we expand a node in this setting, we predict the results of all actions simultaneously, and compute a rollout from each resulting position. In turn, only 50 nodes are expanded. To estimate the node value function, the average reward over all rollouts is propagated back to the root and each node's visit counter is increased by 1. Furthermore, we discount the reward predicted by STOVE with a factor of 0.95 per timestep to account for the higher uncertainty of longer rollouts. This is not done in the baseline running on the real environment, since it behaves deterministically.

For PPO, we employ a standard convolutional neural network as an actor-critic for the evaluation on images and a MLP for the evaluation on states. The image network consists of two convolutional layers, each using 32 output filters with a kernel size of 4 and 3 respectively and a stride of 2. The MLP consists of two fully connected layers with 128 and 64 hidden units. In both cases, an additional fully connected layer links the outputs of the respective base to an actor and a critic head. For the convolutional base, this linking layer employs 512 hidden units, for the MLP 64. All previously mentioned layers use rectified linear activations. The actor head predicts a probability distribution over next actions using a softmax activation function while the critic head outputs a value estimation for the current state using a linear prediction. We tested several hyperparameter configurations but found the following to be the most efficient one.

To update the actor-critic architecture, we sample 32 trajectories of length 16 from separate environments in every batch. The training uses an Adam optimizer with a learning rate of 2×10^{-4} and and ϵ value of 1×10^{-5}. The clipping parameter of PPO is set to 1×10^{-1}. We update the network for 4 epochs in each batch using 32 mini-batches of the sampled data. The value loss is weighted at 5×10^{-1} and the entropy coefficient is set to 1×10^{-2}.

B. Details on ObSuRF

Here, we provide details on the ObSuRF model presented in Section 5.4. This material has been pre-published in (Stelzner et al., 2021).

B.1. Proofs and Derivations

We start by providing detailed derivations for the results referenced in the main text.

B.1.1. Derivation of the NeRF Depth Distribution

In Section 5.3.4, we noted that the distribution $\mathrm{p}(t)$ shown in Eq. (5.4) arises both as the distribution of points along $\mathbf{r}$ who contribute to the color observed at $\mathbf{x_0} = \mathbf{r}(0)$, and as the distribution of points illuminated by a light source located at $\mathbf{x_0}$. Here, we start by showing the latter.

Distribution of Light Originating at $\mathbf{x_0}$

Following the low albedo approximation due to Blinn (1982), we ignore indirect lighting. In order for light originating at $\mathbf{x_0}$ to illuminate $\mathbf{r}(t)$ directly, it must travel along the ray without any scattering events. The rate of scattering events is given by the inhomogenous spatial Poisson process with finite, non-negative density $\sigma(\mathbf{x})$ (Møller & Waagepetersen, 2003). We are therefore looking for the distribution over the position of the first event encountered while traveling along $\mathbf{r}$, also called the *arrival* distribution of the

process (Møller & Waagepetersen, 2003). In a Poisson process, the probability $T(t)$ of encountering no events along a given trajectory $\mathbf{r}(t)$ with $t \geq 0$ (transmittance) is

$$T(t) = \exp\left(\int_0^t -\sigma(\mathbf{r}(t'))dt'\right). \tag{B.1}$$

The probability that light scatters before moving past some point t_0 is therefore $\mathrm{p}(t \leq t_0) = 1 - T(t_0)$. Let us now assume that $\lim_{t_0 \to \infty} T(t_0) = 0$, i.e., each ray of light encounters a scattering event eventually. This is for instance guaranteed if the scene has solid background. In that case, $1 - T(t)$ is the cumulative distribution function for $\mathrm{p}(t)$, since it is non-decreasing, continuous, and fulfills $T(0) = 0, T(\infty) = 1$. We can therefore recover the density function $\mathrm{p}(t)$ by differentiating:

$$\begin{aligned}
\mathrm{p}(t) &= \frac{d}{dt}\left(1 - T(t)\right) && \text{(B.2)}\\
&= -T(t)\frac{d}{dt}\int_0^t -\sigma(\mathbf{r}(t'))dt' && \text{(B.3)}\\
&= \sigma(\mathbf{r}(t))T(t). && \text{(B.4)}
\end{aligned}$$

Distribution of Colors Observed at $\mathbf{x_0}$

As discussed in Section 5.3.4, NeRF does not model light sources explicitly, but assumes that lighting conditions are expressed by each point's RGB color value. The observed color $\hat{C}(\mathbf{r})$ at point $\mathbf{x_0}$ along ray $\mathbf{r}$ is then a blend of the color values along $\mathbf{r}(t)$ (Blinn, 1982; Kajiya & von Herzen, 1984). The intensity with which we can observe the color emitted from point $\mathbf{r}(t)$ in direction $-\mathbf{d}$ of the camera is proportional to the density $\sigma(\mathbf{r}(t))$. Let k denote the constant of proportionality. Since $T(t)$ is the probability that light reaches $\mathbf{x_0}$ from $\mathbf{r}(t)$, the we can observe the color $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$ at $\mathbf{x_0}$ with intensity $k\sigma(\mathbf{r}(t))T(t)$.

In order to obtain the distribution $\mathrm{p}(t)$ over the points from which the colors we observe at $\mathbf{x_0}$ originate, we must normalize this term by dividing by the *total* intensity along ray $\mathbf{r}$:

$$p(t) = \frac{k\sigma(\mathbf{r}(t))T(t)}{\int_0^\infty k\sigma(\mathbf{r}(t'))T(t')dt'} = \frac{\sigma(\mathbf{r}(t))T(t)}{\int_0^\infty \sigma(\mathbf{r}(t'))T(t')dt'}. \tag{B.5}$$

As we have shown in Eq. (B.4), the antiderivative of $\sigma(\mathbf{r}(t))T(t)$ is $-T(t)$. The value of the improper integral is therefore

$$\lim_{t\to\infty}\int_0^t \sigma(\mathbf{r}(t'))T(t')dt' = \lim_{t\to\infty} T(0) - T(t) = 1. \tag{B.6}$$

Consequently, $\mathrm{p}(t) = \sigma(\mathbf{r}(t))T(t)$. This procedure may be viewed as a continuous version of alpha compositing (Porter & Duff, 1984).

B.1.2. Estimating NeRF's Loss is Biased

Here, we show that estimating $\mathcal{L}_{\text{NeRF}}$ via stratified sampling is biased as noted in Section 5.4.1. As a counterexample, consider the following situation in which a ray $\mathbf{r}$ passes through a thin but dense white volume, before hitting black background. For simplicity, we use grayscale color values.

$$\sigma(\mathbf{r}(t)) = \begin{cases} 100 & \text{if } 50 \leq t \leq 51 \\ 10 & \text{if } t > 80 \\ 0 & \text{otherwise,} \end{cases} \tag{B.7}$$

$$c(\mathbf{r}(t), \mathbf{d}) = \begin{cases} 1 & \text{if } 50 \leq t \leq 51 \\ 0 & \text{otherwise.} \end{cases} \tag{B.8}$$

We then have $T(50) = 1$, but $T(51) = T(80) = \exp(-100)$. Since $T(\cdot)$ is the CDF of t (Eq. (B.4)), this means almost all of the incoming light is coming from the thin volume at $t \in [50, 51]$:

$$p(50 \leq t \leq 51) = 1 - \exp(-100) \tag{B.9}$$

$$p(t > 80) = \exp(-100). \tag{B.10}$$

Using Eq. (5.5), we find that the color value is almost exactly one:

$$\hat{C}(\mathbf{r}) = 1 - \exp(-100) \approx 1. \tag{B.11}$$

Let us now consider the result of estimating $\hat{C}(\mathbf{r})$ using $k = 50$ stratified samples $t_1, \ldots, t_k$, with $t_{\text{far}} = 100$. In this case, one sample t_i is drawn uniformly from each of k evenly sized bins:

$$t_i \sim \text{Uniform}\left(\frac{i-1}{k} t_{\text{far}}, \frac{i}{k} t_{\text{far}}\right). \tag{B.12}$$

Only one of the bins, the one for $t_{26} \sim \text{Uniform}(50, 52)$, covers the volume at $[50, 51]$. However, since this volume only makes up half the range of this bin, we have $p(50 \leq t_{26} \leq 51) = 1/2$. Therefore, 50% of the time, sampling in this way will result in samples which miss the volume at $[50, 51]$ entirely. In those cases, we have $c(\mathbf{r}(t_i), \mathbf{d}) = 0$ for all

i. As a result, the estimated color value is $\hat{C}^{|t_i}(\mathbf{r}) = 0$. Even if the estimated color is 1 in all other cases, this sampling scheme will be a biased estimator for the true color values:

$$\mathbb{E}_{t_i}\left[\hat{C}^{|t_i}(\mathbf{r})\right] \leq \frac{1}{2} < \hat{C}(\mathbf{r}). \tag{B.13}$$

Collecting a second round of samples based on the estimated densities as proposed by Mildenhall et al. (2020) does not change the result. In the 50% of cases where the thin volume was missed entirely during by the first set of samples, we have $\sigma(\mathbf{r}(t_i)) = 0$ for all $t_i \leq 80$. As a result, the piecewise-constant PDF used for sampling the second round of samples will equal 0 for $50 \leq t \leq 51$, and none of the newly collected samples will cover the relevant range. Therefore, Eq. (B.13) also holds when the samples t_i were collected using hierarchical sampling.

We note that this effect may be somewhat mitigated by the fact that NeRF models typically use different scene functions for coarse and fine grained sampling (Mildenhall et al., 2020; Kosiorek et al., 2021). The coarse scene function may then learn to model wider volumes which are easier to discover while sampling, instead of the exact scene geometry. However, it seems clear that this will not allow decreasing the number of samples arbitrarily, e.g. to 2 as we did for ObSuRF.

B.1.3. Superposition of Constant Densities Yields a Mixture Model

Here, we illustrate how a mixture model, like the one we use in Section 5.4.2, naturally arises when a superposition of volumes with constant densities σ_i and colors c_i is rendered. Following Eq. (5.6) and writing $\sigma = \sum_i \sigma_i$, the transmittance along a ray $\mathbf{r}$ is

$$T(t) = \exp\left(-\int_0^t \sigma dt'\right) = \exp(-\sigma t). \tag{B.14}$$

As in Section 5.4.1, we can then write

$$p(t, i) = \sigma_i(\mathbf{r}(t))T(t) = \sigma_i \exp(-\sigma t). \tag{B.15}$$

Integrating over t to obtain the distribution over the volumes i yields

$$p(i) = \int_0^\infty p(t, i)dt = \left[\frac{\sigma_i}{-\sigma}\exp(-\sigma t)\right]_0^\infty = \frac{\sigma_i}{\sigma}. \tag{B.16}$$

Using $c_i(\mathbf{x}) = c_i$, and following Eq. (5.8), we find that the observed color will then be

$$\mathbb{E}_{i\sim p(\cdot)}\left[c_i\right] = \sum_{i=1}^{n} c_i \frac{\sigma_i}{\sigma}, \tag{B.17}$$

which is the desired mixture model.

B.2. Architectures

We now present the architectures used in our experiments. When we refer to some of the dimensions in terms of a variable, we specify the concrete values used in Appendix B.4. ObSuRF uses exclusively ReLU activations.

B.2.1. Encoder Architecture

Due to the differences in resolution and complexity, we used different feature extractor architectures for the 2D and 3D experiments. The slot attention module was identical in both cases.

Feature Extractor for 2D Datasets. Following Locatello et al. (2020), we use a convolutional network with 4 layers to encode the input image for the 2D datasets. Each layer has a kernel size of 5×5, and padding of 2, keeping the spatial dimensions of the feature maps constant at 64×64. Each layer outputs a feature map with d_h channels. After the last layer, we apply the spatial encoding introduced by Locatello et al. (2020). Again following Locatello et al. (2020), we process each pixel in the resulting feature map individually using a layer normalization step, followed by two fully connected (FC) layers with output sizes $[d_h, d_z]$.

Feature Extractor for 3D Datasets. For the 3D experiments, we adapt the ResNet-18 architecture to achieve a higher model capacity. We start by appending the camera position and a positional encoding (Mildenhall et al., 2020) of the viewing direction to each pixel of the input image, and increase the input size of the ResNet accordingly. We remove all downsampling steps from the ResNet except for the first and the third, and keep the number of output channels of each layer constant at d_h. Given an input

Parameter	Description	2D Data	CLEVR-3D	MSN
d_z	Dim. of slots	64	128	256
d_h	Dim. of hidden layers	64	128	256
m	# of Slot Attention iterations	3	5	5
n_f	# of frequencies for pos. enc.	8	16	16
k_f	Exponent for lowest frequency	0	-5	-5
σ_{max}	Maximum density	-	10	10
σ_c	Standard deviation of color dist.	1	0.2	0.2
δ	Noise added to depths t	-	0.07	0.07
$\hat{k}_O$	Maximum overlap loss coefficient	-	0.05	0.03
	Start of $\mathcal{L}_O$ ramp up	-	20000	20000
	End of $\mathcal{L}_O$ ramp up	-	40000	40000
	Batch Size	128	64	64
	Rays per instance	-	4096	2048

Table B.1.: Hyperparameters used for the experiments with ObSuRF.

image of size 240×320, the extracted feature map therefore has spatial dimensions 60×80, and d_h channels. We apply the same pixelwise steps described for the 2D case (layer normalization followed by two FC layers) to obtain the final feature map with d_z channels.

Slot Attention. We apply slot attention as described by Locatello et al. (2020) with n slots for m iterations, except for the following change. At each iteration, after the GRU layer, we insert a multi-head self attention step with 4 heads to facilitate better coordination between the slots. Specifically, we update the current value of the slots via

$$\mathbf{z} := \mathbf{z} + \text{Att}(\mathbf{z}, \mathbf{z}, \mathbf{z}). \tag{B.18}$$

After m iterations, the values of the slots form our latent representation $z_1, \dots z_n$. We apply the same number of iterations m during both training and testing.

B.2.2. Decoder Architecture

Here, we describe how density and colors are predicted from the spatial coordinates $\mathbf{x}$ and a conditioning vector z. If pixel conditioning is not used, then z is simply equal to

one of the latent codes z_i. Otherwise, we concatenate z_i with the feature map pixel $d_z^{r,c}$, where r, c are the image space coordinates of the query point $\mathbf{x}$ projected onto the feature map d_z, following Yu et al. (2021a).

To then predict color and density, we first encode the $\mathbf{x}$ via positional encoding (Mildenhall et al., 2020), using n_f frequencies, with the lowest one equal to $2^{k_f}\pi$. We process the encoded positions using a 5 layer MLP with hidden size d_h. After each hidden layer, we scale and shift the current hidden vector h elementwise based on z. Specifically, at each layer, we use a learned linear map to predict scale and shift parameters α, β (each of size d_h) from z, and update $\mathbf{h}$ via $\mathbf{h} := (\mathbf{h} + \beta) \cdot \alpha$.

In the 2D case, we output 4 values at the last layer, namely the log density $\log \sigma$ and the RGB-coded colors $\mathbf{c}$. In the 3D case, we output one value for the density, and d_h values to condition the colors. We append the view direction $\mathbf{d}$ to the latter, and predict the actual color output using two additional FC layers of size $[d_h, 3]$, followed by a sigmoid activation. In the 3D case, we find that it is necessary to set a maximum density value, as the model can otherwise arbitrarily inflate the depth likelihood $p(t)$ by increasing the density in regions which are known to always be opaque, such as the ground plane. We do this by applying a sigmoid activation to the decoder's density output, and multiplying the result by σ_{max}, obtaining values in the range $[0, \sigma_{\text{max}}]$.

B.2.3. NeRF-AE Baseline Details

For the NeRF-AE baseline (Table 5.2), we adapt our encoder architecture to output a single vector instead of a feature map. To this end, we first append a positional encoding (Mildenhall et al., 2020) of each pixel's coordinates to the input image. This replaces the spatial encoding described above, since the method described by Locatello et al. (2020) requires a full-sized feature map to be applicable. We double the number of channels to $2d_h$, but set the convolutional layers to use stride 2, obtaining an output with spatial dimensions 4×4. As above, each of these vectors is processed via layer normalization and two fully connected layers, this time with output sizes $[2d_h, 2d_h]$. We flatten the result to obtain a vector of size $32 \cdot d_h$, and obtain the final vector using an MLP of sizes $[4d'_z, 2d'_z, d'_z]$. This replaces the slot attention component described below. To keep the total dimensionality of the latent space identical, we set $d'_z = nd_z$.

B.2.4. NeRF-VAE Baseline Details

We compare ObSuRF to NeRF-VAE on the quality of image reconstruction and depth estimation. We use NeRF-VAE with the attentive scene function and without iterative inference as reported by Kosiorek et al. (2021), with the majority of parameters the same. The only differences stem from different image resolution used in the current work (240×320 as opposed to 64×64 in (Kosiorek et al., 2021)). To accommodate bigger images, we use larger encoder with more downsampling steps. Specifically, we use [(3, 2), (3, 2), (3, 2), (3, 2) (4, 1)] block groups as opposed to [(3, 2), 3, 2), (4, 1)], which results in an additional $4\times$ downsampling and 6 additional residual blocks. We used the same parameters to train the NeRF-VAE on both datasets: Models were trained for 2.5×10^5 iterations with batch size 192, learning rate of 5×10^{-4} and Adam optimizer. The training objective is to maximize the ELBO, which involves reconstructing an image with volumetric rendering. We sample 32 points from the coarse scene net, and we reuse those points plus sample additional 64 points for evaluating the fine scene net. Kosiorek et al. (2021) were able to set the integration cutoff $t_{\text{far}} = 15$, but for the data generated for ObSuRF. this value was too small and we used $t_{\text{far}} = 40$. This effectively reduces the number of points sampled per unit interval, which might explain higher reconstruction and depth-estimation errors.

B.2.5. NeRF Training Ablation Details

The ablations involving training via NeRF-style ray marching largely use the same architecture as ObSuRF, except for the following changes:

- No pixel conditioning is used, as we have found that this causes the model to fail to segment, likely because it allows the model to achieve good loss scores without relying on the slots.
- No type of overlap loss is used.
- The maximum density is not limited to a maximum value, instead it is predicted using a simple ReLU activation.
- Following Mildenhall et al. (2020), we learn two instances of the decoder for the coarse and fine rendering passes, respectively.

- For training on CLEVR3D, we collect 48 samples each for the coarse and fine rendering pass. For shapenet, we use 64 samples each. To account for the increased memory cost, we reduce the batch size to 16, and the number of rays per instance to 256 and 192, respectively.
- For depth supervision, an MSE loss was placed on the estimated depths. Following Deng et al. (2021), this loss was weighted with the factor $\lambda = 0.1$ for CLEVR3D. For MultiShapeNet, we tried 0.1 and 0.04, but found that neither causes the model to properly segment the scene. The results above were obtained using $\lambda = 0.04$.

B.3. Datasets

Here, we describe both of the newly introduced 3D datasets.

B.3.1. CLEVR-3D

To allow for maximum interoperability with the 2D version of CLEVR, we use the scene metadata for the CLEVR dataset provided by Kabra et al. (2019). While this data does not specify the original (randomized) camera positions, we were able to recover them based on the provided object coordinates in camera space via numerical root finding. We were therefore able to exactly reconstruct the scenes in Blender, except for the light positions, which were rerandomized. To ensure a coherent background in all directions, we added a copy of the backdrop used by CLEVR, facing in the opposite direction. We rendered RGB-D views of the scene from the original camera positions and two additional positions obtained by rotating the camera by $120°/240°$ around the z axis. We test on the first 500 scenes of the validation set, using RGB images from the original view points as input.

B.3.2. MultiShapeNet

For the MultiShapeNet dataset, we start with the same camera, lighting, and background setup which was used for CLEVR-3D. For each scene, we first choose the number of objects uniformly between 2 and 4. We then insert objects one by one. For each one, we first uniformly select one of the *chair*, *table*, or *cabinet* categories. We then uniformly choose a shape out of the training set of shapes provided for that category in the

ShapeNetV2 dataset (Chang et al., 2015), leaving the possibility for future evaluations with unseen shapes. We insert the object at a random x, y position in $[-2.9, 2.9]^2$, scale its size by a factor of 2.9, and choose its z position such that the bottom of the object is aligned with $z = 0$. To prevent intersecting objects and reduce the probability of major occlusions, we compute the radius r of the newly inserted object's bounding circle in the xy-plane. If a previously inserted object has radius r', and their xy-distance $d < 1.1(r + r')$, we consider the objects to be too close, remove the last inserted object, and sample a new one. If the desired number of objects has not been reached after 20 iterations, we remove all objects and start over.

B.4. Model Parameters

We report the hyperparameters used for ObSuRF in Table B.1. In addition to the architectural parameters introduced above, we note the importance of the standard deviation of the color distribution σ_C for tuning the relative importance of color compared to depth. We also report how we ramp up the overlap loss $\mathcal{L}_O$: From the beginning of training to the iteration at which we start the ramp up, we set $k_O = 0$. During the ramp up period, we linearly increase k_O until it reaches the maximum value $\hat{k}_O$. We train using the Adam optimizer with default parameters, and an initial learning rate of $4e - 4$. We reduce the learning rate by a factor of 0.5 every 100k iterations. Finally, we note that for the 3D models, we used gradient norm clipping during training, i.e. , at each iteration, we determine the L2 norm of the gradients of our model as if they would form a single vector. If this norm exceeds 1, we divide all gradients by that norm. When training on MultiShapeNet, we find that very rare, extremely strong gradients can derail training. After the first 5000 iterations, we therefore skip all training steps with norm greater than 200 entirely.