Université de Montréal

# Advances in Generative Models for Dynamic Scenes

par

**Lluis Castrejon**

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en informatique

24 mai 2022

**Université de Montréal**

Faculté des arts et des sciences

Cette thèse intitulée

**Advances in Generative Models for Dynamic Scenes**

présentée par

**Lluis Castrejon**

a été évaluée par un jury composé des personnes suivantes :

*Pascal Vincent*
(président-rapporteur)

*Aaron Courville*
(directeur de recherche)

*Nicolas Ballas*
(codirecteur)

*Christopher Pal*
(membre du jury)

*Sungjin Ahn*
(examinateur externe)

# Résumé

Les réseaux de neurones sont un type de modèle d'apprentissage automatique (ML) qui résolvent des tâches complexes d'intelligence artificielle (AI) sans nécessiter de représentations de données élaborées manuellement. Bien qu'ils aient obtenu des résultats impressionnants dans des tâches nécessitant un traitement de la parole, d'image, et du langage, les réseaux de neurones ont encore de la difficulté à résoudre des tâches de compréhension de scènes dynamiques. De plus, l'entraînement de réseaux de neurones nécessite généralement de nombreuses données annotées manuellement, ce qui peut être un processus long et coûteux. Cette thèse est composée de quatre articles proposant des modèles génératifs pour des scènes dynamiques. La modélisation générative est un domaine du ML qui étudie comment apprendre les mécanismes par lesquels les données sont produites. La principale motivation derrière les modèles génératifs est de pouvoir, sans utiliser d'étiquettes, apprendre des représentations de données utiles ; c'est un sous-produit de l'approximation du processus de génération de données. De plus, les modèles génératifs sont utiles pour un large éventail d'applications telles que la super-résolution d'images, la synthèse vocale ou le résumé de texte.

Le premier article se concentre sur l'amélioration de la performance des précédents auto-encodeurs variationnels (VAE) pour la prédiction vidéo. Il s'agit d'une tâche qui consiste à générer les images futures d'une scène dynamique, compte tenu de certaines observations antérieures. Les VAE sont une famille de modèles à variables latentes qui peuvent être utilisés pour échantillonner des points de données. Comparés à d'autres modèles génératifs, les VAE sont faciles à entraîner et ont tendance à couvrir tous les modes des données, mais produisent souvent des résultats de moindre qualité. En prédiction vidéo, les VAE ont été les premiers modèles capables de produire des images futures plausibles à partir d'un contexte donné, un progrès marquant par rapport aux modèles précédents car, pour la plupart des scènes dynamiques, le futur n'est pas une fonction déterministe du passé. Cependant, les premiers VAE pour la prédiction vidéo produisaient des résultats avec des artefacts visuels visibles et ne fonctionnaient pas sur des ensembles de données réalistes complexes. Dans cet article, nous identifions certains des facteurs limitants de ces modèles, et nous proposons pour chacun d'eux une solution pour en atténuer l'impact. Grâce à ces modifications, nous montrons que

les VAE pour la prédiction vidéo peuvent obtenir des résultats de qualité nettement supérieurs par rapport aux références précédentes, et qu'ils peuvent être utilisés pour modéliser des scènes de conduite autonome.

Dans le deuxième article, nous proposons un nouveau modèle en cascade pour la génération vidéo basé sur les réseaux antagonistes génératifs (GAN). Après le succès des VAE pour prédiction vidéo, il a été démontré que les GAN produisaient des échantillons vidéo de meilleure qualité pour la génération vidéo conditionnelle à des classes. Cependant, les GAN nécessitent de très grandes tailles de lots ainsi que des modèles de grande capacité, ce qui rend l'entraînement des GAN pour la génération vidéo coûteux computationnellement, à la fois en termes de mémoire et en temps de calcul. Nous proposons de scinder le processus génératif en une cascade de sous-modèles, chacun d'eux résolvant un problème plus simple. Cette division nous permet de réduire considérablement le coût computationnel tout en conservant la qualité de l'échantillon, et nous démontrons que ce modèle peut s'adapter à de très grands ensembles de données ainsi qu'à des vidéos de haute résolution.

Dans le troisième article, nous concevons un modèle basé sur le principe qu'une scène est composée de différents objets, mais que les transitions de trame (également appelées règles dynamiques) sont partagées entre les objets. Pour mettre en œuvre cette hypothèse de modélisation, nous concevons un modèle qui extrait d'abord les différentes entités d'une image. Ensuite, le modèle apprend à mettre à jour la représentation de l'objet d'une image à l'autre en choisissant parmi différentes transitions possibles qui sont toutes partagées entre les différents objets. Nous montrons que, lors de l'apprentissage d'un tel modèle, les règles de transition sont fondées sémantiquement, et peuvent être appliquées à des objets non vus lors de l'apprentissage. De plus, nous pouvons utiliser ce modèle pour prédire les observations multimodales futures d'une scène dynamique en choisissant différentes transitions.

Dans le dernier article nous proposons un modèle génératif basé sur des techniques de rendu 3D qui permet de générer des scènes avec plusieurs objets. Nous concevons un mécanisme d'inférence pour apprendre les représentations qui peuvent être rendues avec notre modèle et nous optimisons simultanément ce mécanisme d'inférence et le moteur de rendu. Nous montrons que ce modèle possède une représentation interprétable dans laquelle des changements sémantiques appliqués à la représentation de la scène sont rendus dans la scène générée. De plus, nous montrons que, suite au processus d'entraînement, notre modèle apprend à segmenter les objets dans une scène sans annotations et que la représentation apprise peut

être utilisée pour résoudre des tâches de compréhension de scène dynamique en déduisant la représentation de chaque observation.

**Mots clés.** Réseaux de neurones, apprentissage profond, auto-encodeurs variationnels, réseaux antagonistes génératifs, prédiction vidéo, génération de vidéo, champs de rayonnement neuronal.

# Abstract

Neural networks are a type of Machine Learning (ML) models that solve complex Artificial Intelligence (AI) tasks without requiring handcrafted data representations. Although they have achieved impressive results in tasks requiring speech, image and language processing, neural networks still struggle to solve dynamic scene understanding tasks. Furthermore, training neural networks usually demands lots data that is annotated manually, which can be an expensive and time-consuming process. This thesis is comprised of four articles proposing generative models for dynamic scenes. Generative modelling is an area of ML that investigates how to learn the mechanisms by which data is produced. The main motivation for generative models is to learn useful data representations without labels as a by-product of approximating the data generation process. Furthermore, generative models are useful for a wide range of applications such as image super-resolution, voice synthesis or text summarization.

The first article focuses on improving the performance of previous Variational AutoEncoders (VAEs) for video prediction, which is the task of generating future frames of a dynamic scene given some previous occurred observations. VAEs are a family of latent variable models that can be used to sample data points. Compared to other generative models, VAEs are easy to train and tend to cover all data modes, but often produce lower quality results. In video prediction VAEs were the first models that were able to produce multiple plausible future outcomes given a context, marking an advancement over previous models as for most dynamic scenes the future is not a deterministic function of the past. However, the first VAEs for video prediction produced results with visible visual artifacts and could not operate on complex realistic datasets. In this article we identify some of the limiting factors for these models, and for each of them we propose a solution to ease its impact. With our proposed modifications, we show that VAEs for video prediction can obtain significant higher quality results over previous baselines and that they can be used to model autonomous driving scenes.

In the second article we propose a new cascaded model for video generation based on Generative Adversarial Networks (GANs). After the success of VAEs in video prediction, GANs were shown to produce higher quality video samples for class-conditional video generation. However, GANs require very large batch sizes and high capacity models, which makes training GANs for video generation computationally expensive, both in terms of memory and training time. We propose to split the generative process into a cascade of submodels, each of them solving a smaller generative problem. This split allows us to significantly reduce the computational requirements while retaining sample quality, and we show that this model can scale to very large datasets and video resolutions.

In the third article we design a model based on the premise that a scene is comprised of different objects but that frame transitions (also known as dynamic rules) are shared among objects. To implement this modeling assumption we design a model that first extracts the different entities in a frame, and then learns to update the object representation from one frame to another by choosing among different possible transitions, all shared among objects. We show that, when learning such a model, the transition rules are semantically grounded and can be applied to objects not seen during training. Further, we can use this model for predicting multimodal future observations of a dynamic scene by choosing different transitions.

In the last article we propose a generative model based on 3D rendering techniques that can generate scenes with multiple objects. We design an inference mechanism to learn representations that can be rendered with our model and we simultaneously optimize this inference mechanism and the renderer. We show that this model has an interpretable representation in which semantic changes to the scene representation are shown in the output. Furthermore, we show that, as a by product of the training process, our model learns to segment the objects in a scene without annotations and that the learned representation can be used to solve dynamic scene understanding tasks by inferring the representation of each observation.

**Keywords.** Neural Networks, Deep Learning, Variational AutoEncoders, Generative Adversarial Networks, Video Prediction, Video Generation, Neural Radiance Fields.

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| ML | Machine Learning |
| VAE | Variational AutoEncoder |
| NN | Neural Network |
| IID | Independent and Identically Distributed |
| MLP | Multilayer Perceptron |
| CNN | Convolutional Neural Network |
| ReLU | Rectified Linear Unit |
| BN | Batch Normalization |
| LN | Layer Normalization |
| IN | Instance Normalization |
| RNN | Recurrent Neural Network |
| LSTM | Long Short-Term Memory |
| GRU | Gated Recurrent Unit |
| SGD | Stochastic Gradient Descent |
| MSE | Mean Squared Error |

# Acknowledgements

First I would like to thank my immediate family Gemma, Màrius, Àngela and Mariona as well as Emma, Gala, Lila, Martí, Manolo, Pilar, Edu and Peni. Being far away from them was not easy, but their warm support helped me make it through the cold Montréal winters.

Then, I would like to thank my supervisor Aaron Courville and my co-supervisor Nicolas Ballas. The stimulating discussions and interactions we had throughout the years were always encouraging and they showed me how to approach research challenges and problems. This thesis would not be possible without them.

I would also like to thank all my co-authors Yoshua Bengio and Rim Assouel as well as the Facebook AI Research team in Montréal, including Joelle Pineau, Pascal Vincent, Mike Rabbat, Adriana Romero, Michal Drodzal, Koustuv Sinha, Mido Assran, Quentin Duval and many others. They are incredible researchers and it was a privilege to be a part of this environment.

Finally, I would like to thank all the amazing people I met during my years in Canada, including César, Arantxa, Faruk, Camille, Amaia, Ivan, Mercè and Eli. Their friendship was invaluable and the fun times we had made me feel like at home.

# Chapter 1

# Introduction

The field of Artificial Intelligence (AI) and Machine Learning (ML) has seen large advances over the last years. Much of this progress has been fueled by Neural Networks (NNs) (LeCun et al., 2015), which are a type of machine learning model defined by compositions of functions. Each of these functions or *neurons* performs a computation with adjustable parameters and, by composing multiple of these units, neural networks can compute highly non-linear operations. Through optimization, neural networks learn to solve tasks by extracting informative data representations, as opposed to other machine learning techniques that require hand-engineered data features. In particular, among many other applications, NNs have been successfully used for machine translation (Radford et al., 2018), speech recognition (Wang et al., 2017), image recognition (Krizhevsky et al., 2012) and generation (Brock et al., 2018), reinforcement learning (Mnih et al., 2013), board game playing (Silver et al., 2017) and protein folding (Jumper et al., 2021).

While neural networks have revolutionized AI, there are still tasks and challenges at which NNs have so far been unsuccessful. Specifically, neural networks have difficulties solving tasks involving *dynamic scenes*. Dynamic scenes are scenes that evolve over time. Typical tasks associated with dynamic scenes include object tracking, temporal reasoning or video generation. To illustrate the current capabilities of NNs when it comes to dynamic scene tasks, let us consider the snitch localization task in the CATER dataset (Girdhar & Ramanan, 2019), which involves tracking an object, the snitch, while moving and being occluded by other objects in an evolving visual scene. While this is a simple task for humans, the current state-of-the-art at CATER (Ding et al., 2020) fails more than once every three times at predicting the right location of the snitch at the end of a video with camera motion. Another example is the PHYRE dataset (Bakhtin et al., 2019), in which the goal is to place an object in a simple 2D scene so that the scene reaches a target state after the new placed object interacts with the environment. While the environment has very simple dynamics and object interactions, the state-of-the-art models fail to solve the task more than half the time (Qi et al., 2020). As a last example, let us consider the task of video generation. The current state-of-the-art techniques (e.g., DVD-GAN (Clark et al., 2019)) require large amounts of computational resources and still have trouble generating coherent scenes. While in recent years there have been advancements in all these areas, there is still much room for improvement.

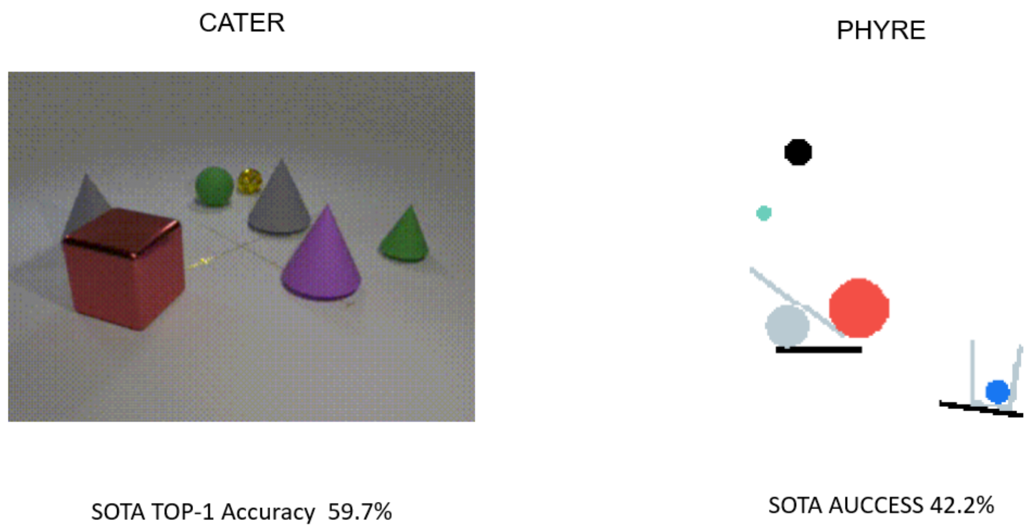


**Figure 1.1 – Examples of Dynamic Scene Understanding tasks:** We show two images of scenes from two dynamic scene understanding datasets. On the left, we show an image of the CATER (Girdhar & Ramanan, 2019) dataset. The goal is to observe the evolution of the scene and predict the location of the small yellow ball, called the snitch. On the right, we show an image from the PHYRE dataset (Bakhtin et al., 2019). The goal is to place an object in the scene, so that after the interaction of the object with the environment the scene reaches a target goal. Despite their simplicity, both tasks are not yet systematically solved.

Why are these examples important? We argue that dynamic scene tasks should not be overlooked as they are often proxy tasks for AI capabilities that can enable applications with significant real-world impact. Let us consider the task of autonomous driving, which holds the promise to revolutionize transportation and reduce the number of road accidents. Autonomous driving requires tracking and forecasting the state of all the entities (cars, bicycles, pedestrians, etc.) in a scene to avoid collisions. Human drivers are able to drive safely under different weather conditions, with partial views of the scene and even with object occlusions. We can consider the snitch localization task in CATER as a simplified version of the problem, in which we have to track the state of multiple objects and predict the localization of the snitch when it is occluded by other objects. Arguably, we would need AI systems that can solve the simplified CATER task systematically before we can safely replace human drivers.

Another relevant example is robotics. Robots that can successfully interact with varying environmental conditions and objects can revolutionize our daily lives by automating menial tasks and assisting humans. In robotic manipulations it is crucial to understand the 3D geometry of objects and the rules governing different object interactions in order to accomplish a particular goal. Similarly, in the PHYRE dataset the objective is to place an object in a dynamic scene so that it reaches a target state due to the interactions of the newly placed object. While certain robotics tasks can be accomplished with great accuracy in constrained environments, the current methods to solve PHYRE dataset illustrate the difficulties faced when trying to learn and simulate the dynamics of an environment over long temporal horizons.

This thesis consists of articles that describe models to generate videos. Generative video models, which we define as machine learning methods capable of generating views of dynamic scenes, are an interesting area of research for two main reasons. First, they can be used to aid tasks involving content generation. For example, one of the most promising new technologies is Virtual Reality (VR) headsets, which can be used to display and interact with 3D content. One crucial aspect in VR is to display detailed visuals to produce immersive simulations, which requires high resolution content. Further, VR requires rendering consistent motions from one frame that react to user movements. This requires high framerates and a lightweight rendering process to be able to incorporate user interactions without delay. These requirements make generating VR content very expensive in terms of computational resources, which are usually limited by the size of the VR headset. Video generation models can help reduce the rendering costs. By designing models that can upscale and increase the framerate of a particular video, traditional rendering techniques can be combined with video generation models to create high resolution VR content while reducing the computational requirements.

On the other hand, generative models can also be used to learn useful data representations. For instance, the task of video prediction consists in extrapolating the future frames in a given context sequence. Arguably, a model that is able to perform this task captures the dynamic rules of an environment, as it is able to simulate its future states. The representations learned by these models can subsequently be used to perform downstream dynamic understanding tasks. As an example, if we have a model that can perfectly predict the future frames in sequences from the CATER dataset, then these model captures the location of the different objects in the scene and we can use its representation to find the location of the snitch.

In the first article in this thesis we investigate variational autoencoders for video. Variational autoencoders are a type of generative models for video that are well-known due to being able to capture stochasticity in the generation. Because of this, they are often used for video prediction, in which multiple future states might be possible given a context sequence. In this article we propose several improvements over previous variational autoencoders for video prediction that result in sharper generated videos. We also showcase that these improvements allow our model to generate data from more realistic scenarios than possible before.

In the second article we investigate a different type of generative model - generative adversarial networks. Compared to variational autoencoders, generative adversarial networks often produce more realistic generations, and therefore are popular to generate high fidelity content. In this article we propose a model that can produce state-of-the-art videos while requiring half the computational resources needed with competing approaches. The core idea is to decompose the generative process into multiple simpler steps, and use an independence assumption that reduces the memory and computational complexity of the overall model. Furthermore, we showcase that this model can generate long and high resolution videos beyond what is possible with previous methods due to its better scaling properties.

While the first and second article investigated different types of generative models and proposed improvements that increased the quality of the generations or reduced their cost, the rest of the articles in this thesis focus on designing methods incorporating inductive biases known to better capture different aspects of dynamic scenes. In the third article, we design a video prediction model that considers that a scene state can be represented by the state of the multiple objects contained in it. Further, all objects in our model representation share the same set of dynamic rules, which are used to update predict their future state. These design decisions represent the fact that a scene is composed of multiple objects and that the rules of motion (gravity, object interactions, etc.) are the same for all objects. We show that this model can be used for video prediction, and we show that, as a result of incorporating these inductive biases, our model has the capability to represent and generate sequences not seen during the training phase.

Finally, in the last article we design a model that incorporates 3D understanding of the different objects in the scene. Similarly to the previous article, our model represents a scene state as the state of its different objects. However, this model represents each object as a 3D model that can be rendered with ray marching, a technique from the computer graphics literature, and explicitly factorizes the information about its pose and its appearance. With this design we can use the model to generate novel scene views by adding or removing objects, or by modifying their appearances or poses. Furthermore, we show that the representations learned by our model can be used to tackle the snitch localization task of the CATER dataset.

## 1.1. Structure of this Document

The main content of this thesis are four articles. First, in Chapter 2 we introduce the basics of machine learning needed to understand the rest of the contents of this thesis. Chapters 4, 6, 8 and 10 contain the articles that form the core of the thesis. Each one of this articles is prefaced by a short prologue chapter that contains details about the context, contributions and recent developments related to the article. Finally, in Chapter 11 we discuss the main findings in each of the articles and discuss possible future follow-up work.

# Chapter 2

# Background

In this section we review concepts in artificial intelligence necessary to understand the contents of this thesis. We start by presenting the basics of machine learning. Then, we review neural networks, including the layers and optimizers. We then present a family of machine learning algorithms called generative models which are the main subject of this thesis. Finally, we provide a brief introduction to neural rendering techniques, which are a key component of the last article in this thesis.

## 2.1. Machine Learning

### 2.1.1. Introduction

Software engineering consists in using programming languages to implement algorithms, which can be understood as a set of steps to perform a calculation. For example, we can code a program that implements the rules to map roman numbers to integers. Such program would receive as input a character string such as 'XV' and output the corresponding integer 15. There are a few steps required to transform roman numerals to the decimal system, they mostly include translating letters to a value, adding these values and taking into account special cases in which we substract values instead of adding them, as in the case of the number 'IX' (9), in which the value of 'I' (1) is substracted to the value of 'X' (10). For many common computations we can design efficient algorithms that perform them.

However, we cannot always design algorithms that cover all possible input-output pairs for a particular task. Let's assume that we are interested in writing a program that decides whether there is a dog in an image. This program takes as input a picture, which can be understood as a matrix of pixels, in which for each pixel we have its corresponding color value (usually represented in 24 bits, 8 bits for each red, green and blue color channels). It is hard to design an algorithm that would correctly classify all images containing (or not) a dog, as there are many different breeds and poses leading to very different pixel patterns.

The field of machine learning (ML) studies algorithms that perform flexible computations influenced by data. These algorithms, which we often call *models*, usually specify a transformation of inputs into outputs. These transformations are not static, and instead can be modified to adjust the input to output mapping. In ML, these transformations are tuned by data.

There are different types of ML models. We categorize them according to the task they perform as follows:

- **Supervised learning:** supervised models try to learn an input-output mapping from a dataset consisting in examples and associated labels.
- **Unsupervised learning:** unsupervised models only have access to unlabeled data with the main objective of learning a data representation. There are different types of unsupervised learning, including *clustering*, *density estimation* or *generative modeling.*
- **Reinforcement learning:** in which an agent interacts with an environment and learns a policy of actions that maximizes a reward signal.

The articles in this thesis focus on unsupervised learning. More specifically, they focus on a type of unsupervised learning algorithms called *generative models*, which we describe in more detail in Section 2.3. For a detailed overview of the field of machine learning we refer the reader to the excellent book Murphy (2012).

### 2.1.2. Unsupervised Learning

In unsupervised learning the main goal is to maximize the likelihood of the data given by a model. More formally, given a dataset of data examples $\mathbf{x} = \{x_1, ..., x_N\}$ and a model with parameters $\boldsymbol{\theta} = \{\theta_1, ..., \theta_K\}$, we would like to solve the following optimization problem:

$$\max_{\boldsymbol{\theta}} p(\mathbf{x}|\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} \log p(\mathbf{x}|\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} -\log p(\mathbf{x}|\boldsymbol{\theta}) \tag{2.1}$$

In general we assume that the data instances are independent and identically distributed (IID assumption), which allows us to further expand this objective:

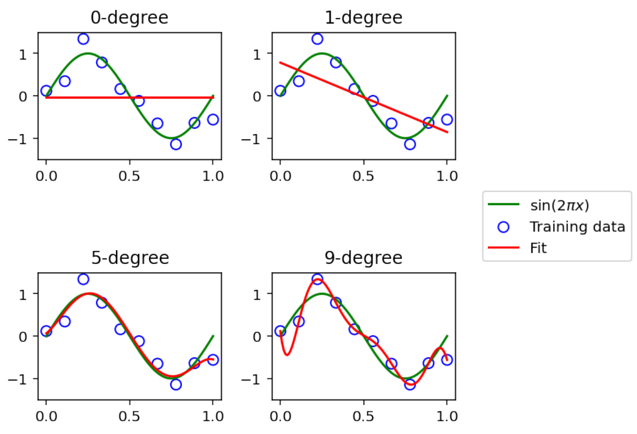


**Figure 2.1 – Model capacity and overfitting** We fit M-degree polynomial models to noisy samples from a sin function. Low degree polynomials ($M$ = 0,1) have low capacity and do not fit well the sin function. As we increase the polynomial degree ($M = 5$), the model has higher representational power and fits well the unknown function. By further increasing the capacity ($M = 9$) the model can fit perfectly the training samples, but this fit does not represent well the unknown function, as phenomenon known as *overfitting*. Example insipired by Bishop & Nasrabadi (2006).

$$p(\mathbf{x}) = p(x_1, ..., x_N) = \prod_{i=0}^{N} p(x_i) \tag{2.2}$$

or, equivalently when using logarithms:

$$\log p(\mathbf{x}) = \log p(x_1, ..., x_N) = \sum_{i=0}^{N} \log p(x_i) \tag{2.3}$$

Maximizing the empirical data likelihood under certain models can be trivial. However, in general we care about maximizing the likelihood of the true data distribution, for which our training set is only a sample. This concept is commonly known as *generalization*, and it is tightly related to the concept of *model capacity*. The capacity of a model usually refers to its number of parameters. In general, higher capacity models can represent higher order functions and more easily fit the training data, but that fit might not necessarily mean that the model is better approximating the true data distribution. We can measure the generalization capabilities of a model by using a held-out data set, which is not used for training. A model that fits well the empirical data likelihood but not the validation data is said to be *overfitting*.

To illustrate this phenomenon, consider the example shown in Figure 2.1, inspired by a similar problem in Bishop & Nasrabadi (2006). We want to fit a model to an unknown function $\sin(2\pi x)$ for which we have some noisy observations. These observations are obtained by adding Gaussian noise to samples from the unknown function $y = \sin(2\pi x) + \mathcal{N}(0, 1)$. Our model consists of a polynomial function of the form $f(x) = \sum_{i=0}^{M} w_i x^i$, where $M$ is the degree of the polynomial and $w_i$ are the parameters of the model. The capacity of the model is defined by the degree of the polynomial, with higher degree polynomial having more parameters and higher representational power. Figure 2.1 shows the fitting of different models of increased capacity to the observations using a method known as linear regression. Using $M = 0$ or $M = 1$, our model is too simple and does not provide a good fit to the true function, which commonly known as *underfitting*. When we choose $M = 5$, our model approximates the unknown function well. As we further increase the capacity of the model, it starts to overfit to the observations, not generalizing well to the unknown function.

The models presented in this thesis are all *neural networks* (NNs). We discuss all aspects related to neural networks in the following section.

## 2.2. Neural Networks

### 2.2.1. Definition

Neural networks compute a function $g_{\boldsymbol{\theta}} : \mathcal{X} \rightarrow \mathcal{Y}$ mapping inputs in input space $\mathcal{X}$ to outputs in space $\mathcal{Y}$. Neural networks are parametrized by a vector $\boldsymbol{\theta} = \{\theta_1, ..., \theta_K\} \in \mathbb{R}^K$ also known as a vector of *weights*. These parameters can be adjusted to modify the mapping computed by the neural network. Generally, the function $g$ compute by a neural network is the result of composing $L$ simpler functions $g^l$ also called *layers*:

$$g_{\boldsymbol{\theta}} = g^L_{\boldsymbol{\theta}^L} \circ g^{L-1}_{\boldsymbol{\theta}^{L-1}} \circ \cdots \circ g^1_{\boldsymbol{\theta}^1} \tag{2.4}$$

Each layer has its own parameters $\boldsymbol{\theta}^l$, and $\boldsymbol{\theta}$ can alternatively be expressed as a concatenation of layer parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^L\} = \{\theta_1, \ldots, \theta_K\}$. Not all the layers in a neural network have weights. For example, it is common to compose non-linear functions without parameters such as tanh, also known as *activation* functions, which make the mapping $g$ capable of representing a wider family of functions. The number of layers in a neural network is often known as its *depth*. The use of neural networks with a large number of layers has led to a revolution in the field of machine learning, and the study of these networks is known as *deep learning*.

### 2.2.2. Types of Layers

In this section we provide an overview of the most common layers used in current neural networks.

**Fully-connected layers.** FC layers compute an affine mapping of inputs to outputs. They are parametrized by a weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$ and perform the following computation:

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{2.5}$$

There are a few special cases to consider for fully-connected layers. When using a network with a single fully-connected layer and a mean-square error loss, we have a model that is equivalent to *linear regression*. A network that consists of multiple fully-connected layers computes a linear function of its input, as the composition of linear functions is a linear function. A network that stacks multiple fully-connected layers interspersed with non-linear activation functions is called a *multilayer perceptron* (MLP).

**Convolutional layers.** Convolutional layers compute functions on sequences of inputs. Similarly to fully-connected layers, they are parametrized by a weight matrix $\mathbf{W}$ also called the *kernel* and a bias vector $\mathbf{b}$, and compute the following function:

$$f(\mathbf{x}) = \mathbf{W} * \mathbf{x} + \mathbf{b} \tag{2.6}$$

where $*$ denotes the convolution operation.

Convolutions can be used to process sequences with different dimensionalities. For example, audio signals can be processed with one dimensional convolutions, images can be processed with two-dimensional convolutions, and videos can be seen as 3D sequences. When defining a convolutional layer, there are a few design choices to be made. In particular, the *kernel size* determines how many elements are considered at each processing step, the *stride* determines how many elements are skipped at each convolution location, and the *padding* scheme determines which values to use when a position uses elements outside of the input sequence length. For a more detailed explanation of convolutions with intuitive visualizations we refer the reader to the tutorial by Dumoulin & Visin (2016). Similarly to how MLPs define networks of fully connected layers, a network composing convolutions with interspersed activation functions is commonly known as a Convolutional Neural Network (CNN) or a ConvNet.

**Activation functions.** Activation functions are mappings that are used in neural networks so that they can represent non-linear functions. Traditionally neural networks used the *sigmoid* (noted by $\sigma(x)$) or hyperbolic tangent $\tanh(x)$ activations functions, defined as follows:

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \sigma(x) = \frac{1}{1 + \exp(x)} \tag{2.7}$$

In recent years Rectified Linear Units (ReLUs) are used instead:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.8}$$

The final activation function to mention is the softmax function:

$$\mathrm{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.9}$$

**Normalization layers.** Normalization layers, as their name indicate, normalize the values of their inputs. There are different normalization schemes that can be used, and when inputs have multiple dimensions, an important consideration is over which dimensions are values normalized. Normalization helps the optimization of neural networks, and are currently an important part of state-of-the-art models. We consider three different normalization schemes in this thesis, namely *batch*, *layer* and *instance* normalization.

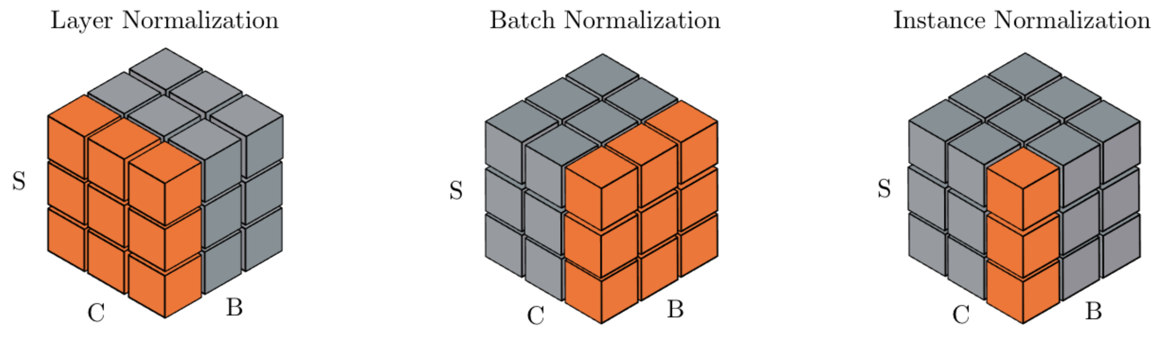


**Figure 2.2 – Normalization schemes** For each normalization scheme we show the dimensions used to compute the mean and variance used in the normalization step. We show a tensor for each scheme with B as the batch dimension, C the features dimension and S the sequence dimension(s). Orange tensor elements are normalized using the same mean and variance, which is obtained by aggregating the values of these elements. Figure inspired by Wu & He (2018).

These normalization layers perform the following computation:

$$\text{Norm}(\mathbf{x}_k) = \gamma_k \frac{\mathbf{x}_k - \mu_k}{\sqrt{\sigma_k^2}} + \beta_k \tag{2.10}$$

where $\mu_k$ and $\sigma_k^2$ denote an empirical mean and variance:

$$\mu_k = \frac{1}{M}\sum_{i=1}^{M} \mathbf{x}_{i,k} \qquad \text{and} \qquad \sigma_k^2 = \frac{1}{M}\sum_{i=1}^{M} (\mathbf{x}_{i,k} - \mu_k)^2 \tag{2.11}$$

The dimensions over which these mean and variance are computed differ across normalization schemes. We assume that inputs are sequences and that they come in batches, so that they are tensors with dimensions $B \times S \times C$, where B denotes batch dimension, S denotes sequence dimension(s), and C denotes the feature dimension. Figure 2.2 shows over which dimensions the mean and variance are computed for each normalization scheme.

**Recurrent layers.** Another alternative to process sequential data is to use recurrent layers (Rumelhart et al., 1985). While convolutional layers process elements in a sequence independently, the main characteristic of recurrent layers is that their computation is affected by previous values in the sequence. This allows recurrent layers to model long-term correlations in a sequence.

There are different types of recurrent layers. In general, they all perform the following computation:

$$\mathbf{y}_t = g_{\boldsymbol{\theta}}(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{2.12}$$

with each type of recurrent layer implementing a different recurrent function $g$. The most basic recurrent layer is commonly known as Tanh-RNN and operates as follows:

$$\mathbf{h}_t = \tanh(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}) \tag{2.13}$$

Optimizing neural networks with Tanh-RNN layers is difficult. To alleviate these training difficulties, the Long Short-Term Memory (LSTM) layer (Hochreiter & Schmidhuber, 1997) was proposed. LSTMs perform the following computation:

$$\begin{pmatrix} \mathbf{i_t} \\ \mathbf{f_t} \\ \mathbf{o_t} \\ \mathbf{g_t} \end{pmatrix} = \mathbf{W_h} * \mathbf{h_{t-1}} + \mathbf{W_x} * \mathbf{x_t} + \mathbf{b} \tag{2.14}$$
$$\mathbf{c_t} = \sigma(\mathbf{f_t}) \odot \mathbf{c_{t-1}} + \sigma(\mathbf{i_t}) \odot \tanh(\mathbf{g_t})$$
$$\mathbf{h_t} = \sigma(\mathbf{o_t}) \odot \tanh(\mathbf{c_t})$$

Here $i$, $f$, $o$ denote the input, forget, and output gate, $h$ is the hidden state and $c$ is the cell state. $\sigma$ denotes the sigmoid function, $\odot$ indicates an element-wise product and $*$ a convolution. $W_h$ denotes the hidden-to-state convolution kernel and $W_x$ the input-to-state convolution kernel.

Another alternative with similar properties to LSTMs is the Gated Recurrent Unit (GRU) (Cho et al., 2014):

$$\begin{pmatrix} \mathbf{r_t} \\ \mathbf{z_t} \end{pmatrix} = \mathbf{W_u} * \mathbf{h_{t-1}} + \mathbf{U_u} * \mathbf{x_t} + \mathbf{b}_u \tag{2.15}$$
$$\hat{\mathbf{h}}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{r}_t \odot \mathbf{U}_h \mathbf{x}_t + \mathbf{b}_h)$$
$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \hat{\mathbf{h}}_\mathbf{t}$$

A neural network formed by recurrent layers is called Recurrent Neural Network (RNN).

### 2.2.3. Optimization

In unsupervised learning we are usually interested in optimizing the data likelihood given by our model. More generally, when optimizing neural networks we are interested in optimization problems of the following form:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \mathcal{L}(\boldsymbol{\theta}) \tag{2.16}$$

where the solution is a weight vector $\boldsymbol{\theta}$ that minimizes a loss function $\mathcal{L}(\boldsymbol{\theta})$. In general the loss function can be a non-linear non-convex function of the model parameters and we might not have a closed-form solution to the problem. However, we can use first-order optimization methods to obtain approximate solutions to the problem. In particular, neural networks are often optimized through *gradient descent*, which consists in iteratively updating a candidate weight vector in the opposite direction of the gradient of $\mathcal{L}(\boldsymbol{\theta})$ wrt to $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i - \lambda \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i} \tag{2.17}$$

$\lambda$ is called the learning rate, and controls how far along the direction of the negative gradient the weight vector is updated. $\lambda$ is a *hyperparameter* - a variable defined by the user that influences the behavior of the model.

In general the gradient of the loss function with respect to the model parameters should be computed on all training examples used to fit the model. However, in practice this might not be possible due to computational constraints. Instead, it is more common to use only a subset of training examples at each optimization step. In this case, the optimization procedure is called *stochastic* gradient descent (SGD), and the examples used for a particular optimization step are called a *mini batch*.

When it comes to the loss function, the particular choice usually resides in the type of model and task that we are trying to solve. Some common losses include the mean-squared error (MSE) loss:

$$\ell_{MSE}(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{t}_i) = \frac{1}{2} \left\| f_{\boldsymbol{\theta}}(\mathbf{x}_i) - \mathbf{t}_i \right\|_2^2 \tag{2.18}$$

or the cross-entropy (CE) loss:

$$\ell_{CE}(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{t}_i) = -\sum_{k=1}^{K} \mathbf{t}_{ik} \log \left( f_{\boldsymbol{\theta}}(\mathbf{x}_i)_k \right) \tag{2.19}$$

To compute gradients wrt to the model parameters in neural networks we use *backpropagation*. Backpropagation consists in applying the chain rule to compute derivates noting that neural networks are composition of functions. Starting from equation 2.4 we apply the chain rule backwards and, for a given layer $l$, we obtain the following expression for computing the derivative of the loss function wrt to its parameters $\boldsymbol{\theta}^l$:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^l} = \frac{\partial f^l_{\boldsymbol{\theta}^l}}{\partial \boldsymbol{\theta}^l} \frac{\partial f^{l+1}_{\boldsymbol{\theta}^{l+1}}}{\partial f^l_{\boldsymbol{\theta}^l}} \cdots \frac{\partial f^L_{\boldsymbol{\theta}^L}}{\partial f^{L-1}_{\boldsymbol{\theta}^{L-1}}} \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial f^L_{\boldsymbol{\theta}^L}} \tag{2.20}$$

This can be efficiently implemented by computing the gradient layer by layer, starting from the last layer $L$, and *backpropagating* it through the network up to the first layer. The computation that each layer needs to perform can be summarised in two equations:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^l} = \frac{\partial f^l_{\boldsymbol{\theta}^l}}{\partial \boldsymbol{\theta}^l} \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial f^l_{\boldsymbol{\theta}^l}} \tag{2.21}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial f^{l-1}_{\boldsymbol{\theta}^{l-1}}} = \frac{\partial f^l_{\boldsymbol{\theta}^l}}{\partial f^{l-1}_{\boldsymbol{\theta}^{l-1}}} \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial f^l_{\boldsymbol{\theta}^l}} \tag{2.22}$$

where Equation 2.21 computes the gradient with respect to the input of the layer and Equation 2.22 computes the gradient with respect to its own parameters.

While SGD is still used to train neural networks, specially a variant called SGD with *momentum*, other optimization schemes have been proposed that sometimes converge faster or to better solutions. In this thesis most of our models are trained using the Adam optimizer (Kingma & Ba, 2014). Although great care is taken to parameterise and initialise neural networks to ease gradient propagation, it might still be difficult to move around the loss landscape $\mathcal{L}(\boldsymbol{\theta})$ following only stochastic gradient information. To alleviate this issue, one can use the more fancy optimisers presented below to train neural networks more efficiently.

We refer the reader to the excellent book on deep learning (Goodfellow et al., 2016) for a more complete overview of the field of neural networks.

## 2.3. Generative Models

### 2.3.1. Introduction

We define *generative* models as statistical models that can generate data instances. This is in contrast with *discriminative* models which, as their name indicates, focus on discriminating between different types of data instances.

More formally, generative models capture the joint data $X$ and label $Y$ distribution $p(X, Y)$, while discriminative models capture the conditional probability $p(Y|X)$. If there are no labels present, then generative models capture the data distribution $p(X)$. Note that in this case generative models are a specific type of unsupervised learning that involves models that can generate data instances. Not all unsupervised models are generative, as models that perform representation learning or clustering are not necessarily able to generate data instances.

### 2.3.2. Latent Variable Models

Most generative models investigated in this thesis are *latent variable* models. Latent variable models are a type of probabilistic graphical model (PGM). PGMs are a formal way of defining joint distributions on sets of random variables through graphs, in which nodes denote random variables and (the absence of) edges between nodes indicate conditional independence relationship between random variables.

In particular, the latent variable models presented in this thesis are Directed Graphical Models (DGMs), also known as Bayesian Networks. PGMs use directed acyclical graphs to define probability distributions. A full in-depth explanation of DGMs goes beyond the scope of this thesis, for an overview we refer the reader to (Murphy, 2023).

The generative models in this thesis follow a simple PGM, with hidden variables $z$ and data instances $x$. The latent variables capture hidden factors of variation in the data. More specifically, they define a generative process in which a deterministic function maps each possible value of the latent variables to data instances.

Formally, this latent variable model defines a joint probability over latents and data factorized as follows:

$$p(x, z) = p(x|z)p(z) \tag{2.23}$$

In general we are interested in obtaining the data probability $p(x)$, which can be obtained by marginalizing over $z$:

$$p(x) = \int_z p(x|z)p(z) \tag{2.24}$$

However, computing this integral is usually intractable. The generative models we introduce subsequently provide alternatives to approximating this computation.

### 2.3.3. Variational AutoEncoders

Variational AutoEncoders (VAEs) are a type of latent variable generative model. They use variational inference to approximate the data distribution.

As we saw before, we consider the following computation:

$$p(x) = \int_z \underbrace{p(x|z)}_{\text{likelihood}} \underbrace{p(z)}_{\text{prior}} \tag{2.25}$$

Computing the data marginal distribution is intractable. We can resort to *variational inference* to approximate this distribution. Variational inference introduces an *approximate posterior distribution* $q(z|x)$, that approximates the true posterior distribution $p(z|x)$.

Using the posterior distribution, we can derive what is known as the Evidence Lower Bound (ELBO) on the data probability as follows:

$$\begin{aligned}
\log p(x) &= \log(\int_z p(x|z)p(z)) \\
&= \log(\int_z p(x|z)p(z)\frac{q(z|x)}{q(z|x)}) \\
&= \log(\mathcal{E}_{q(z|x)}\frac{p(z)}{q(z|x)}p(x) \\
&\geq \mathbb{E}_{q(z|x)} \log(\frac{p(z)}{q(z|x)}p(x)) \\
&\geq \mathbb{E}_{q(z|x)} \log p(x|z) - \mathcal{D}_{KL}(p(z)||q(z|x))
\end{aligned}$$

Since we are interested in maximizing the likelihood of the data, we can maximize this lower bound and formulate the following loss for a VAE:

$$\min_\theta \quad \underbrace{\mathbb{E}_{q(z|x)} - \log p(x|z)}_{\text{reconstruction term}} + \underbrace{\mathcal{D}_{KL}(p(z)||q(z|x))}_{\text{divergence term}} \tag{2.26}$$

VAEs additionally define $q(z|x)$ and $p(x|z)$ as diagonal gaussian distributions with parameters obtained through a neural network. To train VAEs, the expectation over samples from the approximate posterior is approximated with Monte Carlo sampling, while the KL divergence can be computed in closed-form since we typically use gaussian distributions. To estimate the gradients through sampling q(z|x), VAEs use the reparametrization trick (Kingma & Welling, 2013).

## 2.3.4. Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a family of latent variable models. Different from VAEs, GANs do not learn explicitly data distributions $p(x)$. Instead, they can generate data samples which implicitly follow the data distribution, thus they belong to a family of models commonly called *implicit* generative models.

To approximate the data distribution without having an explicit data likelihood distribution, GANs rely on discriminating populations of samples. Intuitively, if the samples generated from the model are undistinguishable from those of the data distribution, then the model has recovered the data generation process.

GANs define a two-player game between a generator network $G$, whose goal is to transform samples from a latent noise distribution into data samples such as those from the data distribution, and a discriminator $D$ function that tries to tell apart *real* datapoints coming from the data distribution from the *fake* ones produced by $G$.

Formally, the original GAN formulation defines the following optimization problem:

$$\min_G \max_D V(G, D) = \mathbb{E}_{\mathbf{x}\sim P_x}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim P_z}[1 - \log D(G(\mathbf{z}))] \tag{2.27}$$

where $\mathbf{x}$ denotes datapoints and $\mathbf{z}$ is a latent variable vector.

Optimizing such minimax objectives with Stochastic Gradient Descent when G and D are parametrized by deep neural networks is difficult. Empirically, it has been found that the optimization is highly unstable and often diverges, only converging with very particular sets of hyperparameters. Different alternative versions of the original GAN have been proposed such as Wasserstein GANs (Arjovsky et al., 2017) or Least Squares GANs (Mao et al., 2017), with different objective functions. These alternative formulations usually aim to make GAN training more stable. Regularization techniques such as Spectral Normalization (Miyato et al., 2018) have also been found to stabilize the GAN training regime.

Despite the abundant literature on alternatives to the original GAN formulation, it has been disputed whether these modified GAN objectives have any advantages in practice over the original objective (Lucic et al., 2018). Instead, the current state of the art GAN models for images rely on powerful neural architectures, strong regularization and careful hyperparameter tuning (Brock et al., 2018).

### 2.3.5. Generative models for Video

GANs and VAEs are often used to generate images. In this thesis we focus on improving generative models for videos. We distinguish between different types of tasks in generative video modeling. *Video generation* is the task of generating complete videos from scratch. In contrast, in *video prediction* tasks the goal is to generate plausible future frames given certain past frames, which form what is known as the context for the video prediction.

In both settings we might or might not have additional conditioning information such as information about the actions taken by different entities of the scene through time or a general video class label. When such information is available, we talk about *conditional* tasks. The articles in this thesis mostly focus on class-conditional video generation and unconditional video prediction.

# Chapter 3

# Prologue to the First Article

## 3.1. Article Details

**Improved Conditional VRNNs for Video Prediction.** Lluis Castrejon, Nicolas Ballas and Aaron Courville, In *Proceedings of the 2009 IEEE/CVF International Conference on Computer Vision (ICCV 2019).*

**Author contributions.** I developed and implemented the method presented in this article, ran most of the experiments, and co-wrote the article. Nicolas Ballas helped define benchmarks and experiments, provided support with the implementation, and co-wrote the article. Aaron Courville supervised the project and revised the article.

## 3.2. Context

SV2P (Babaeizadeh et al., 2018) proposed a model based on VAEs for video prediction that was able to generate multiple plausible future outcomes given a few context frames. However, this model used a single latent variable to capture uncertainty over full video sequences. Learning a mapping from a single latent variable to all possible future outcomes is a complex task, and the proposed model only worked properly on toy datasets. Improving upon this, Denton & Fergus (2018) proposed a model with a latent variable per time-step and a learnable prior, which allowed it to go beyond toy datasets to more complex scenarios. This model, called SVG, is still unable to model realistic videos, which limits its applications to tasks involving real world data.

## 3.3. Contributions

In this work we consider improvements over the VAE models presented in (Babaeizadeh et al., 2018; Denton & Fergus, 2018). We identify limitations in the main components of these models and propose solutions that improve their performance, validated through model ablations. Finally, we use our proposed model in an autonomous driving dataset, showing for the first time successful video predicitions in the Cityscapes dataset without using annotations.

## 3.4. Recent Developments

Our model highlighted important issues with VAEs for video prediction.

Similar to our paper, Villegas et al. (2019) highlighted that increasing the capacity of video prediction models lead to better generation quality. Inspired by hierarchical latent structures such as the one presented in our model, Saxena et al. (2021) proposed a model with hierarchical latent variables operating at different temporal resolutions, which allows to model long-term dynamics. Chatterjee et al. (2021) extends our model with a weighted loss based on the prediction uncertainty at each time-step that leads to improved generations under certain metrics.

More recently, our finding that video prediction models underfit to the training data has been highlighted as a main limiting factor in video prediction, and FitVid (Babaeizadeh et al., 2021) builds upon it to propose a high-capacity model that is currently the state-of-the-art model in video prediction.

# Chapter 4

# Improved Conditional VRNNs for Video Prediction

**Abstract.** Predicting future frames for a video sequence is a challenging generative modeling task. Promising approaches include probabilistic latent variable models such as the Variational AutoEncoder (VAE). While VAEs can handle uncertainty and model multiple possible future outcomes, they have a tendency to produce blurry predictions. In this work we argue that this is a sign of underfitting. To address this issue, we propose to increase the expressiveness of the latent distributions and to use higher capacity likelihood models. Our approach relies on a hierarchy of latent variables, which defines a family of flexible prior and posterior distributions in order to better model the probability of future sequences. We validate our proposal through a series of ablation experiments and compare our approach to current state-of-the-art latent variable models. Our method performs favorably under several metrics in three different datasets.

## 4.1. Introduction

We investigate the task of video prediction, a particular instantiation of self-supervision (Devlin et al., 2018; Gidaris et al., 2018) where generative models learn to predict future frames in a video. Training such models does not require any annotated data, yet the models need to capture a notion of the complex dynamics of real-world phenomena (such as physical interactions) to generate coherent sequences.

Uncertainty is an inherent difficulty associated with video prediction, as many future outcomes are plausible for a given sequence of observations (Babaeizadeh et al., 2018; Denton & Fergus, 2018). Predictions from deterministic models rapidly degrade over time as uncertainty grows,

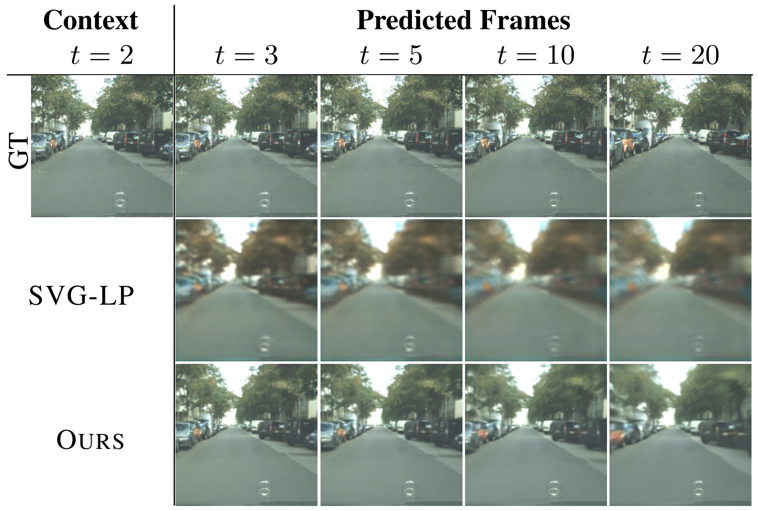


**Figure 4.1 – Can generative models predict the future?** We propose an improved VAE model for video prediction. Our model uses hierarchical latents and a higher capacity likelihood network to improve upon previous VAE approaches, generating more visually appealing samples that remain coherent for longer temporal horizons.

converging to an average of the possible future outcomes (Srivastava et al., 2015a). To address this issue, probabilistic latent variable models such as Variational Auto-Encoders (VAEs) (Kingma & Welling, 2013; Rezende et al., 2014), and more specifically Variational Recurrent Neural Networks (VRNNs) (Chung et al., 2015), have been proposed for video prediction (Babaeizadeh et al., 2018; Denton & Fergus, 2018). These models define a prior distribution over a set of latent variables, allowing different samples from these latents to capture multiple outcomes.

It has been empirically observed that VAE and VRNN-based models produce blurry predictions (Larsen et al., 2015; Lee et al., 2018). This tendency is usually attributed to the use of a similarity metric in pixel space (Larsen et al., 2015; Mathieu et al., 2015) such as Mean Squared Error (corresponding to a log-likelihood loss under a fully factorized Gaussian distribution). This has lead to alternative models such as the VAE-GAN (Larsen et al., 2015; Lee et al., 2018), which extends the traditional VAE objective with an adversarial loss in order to obtain more visually compelling generations.

In addition, the lack of expressive latent distributions has been shown to lead to poor model fitting (Hoffman & Johnson, 2016). Training VAEs involves defining an approximate posterior distribution over the latent variables which models their probability after the generated data has been observed. If the approximate posterior is too constrained, it will not be able to match the true posterior distribution and this will prevent the model from accurately fitting the training data. On the other hand, the prior distribution over the latent variables can be interpreted as a model of uncertainty.

The decoder or likelihood network needs to transform latent samples into data observations covering all plausible outcomes. Given a simple prior, this transformation can be very complex and require high capacity networks. We hypothesize that the reduced expressiveness of current VRNN models is limiting the quality of their predictions and investigate two main directions to improve video prediction models. First, we propose to scale the capacity of the likelihood network. We empirically demonstrate that by using a high capacity decoder we can ease the latent modeling problem and better fit the data.

Second, we introduce more flexible posterior and prior distributions (Sønderby et al., 2016b). Current video prediction models usually rely on one shallow level of latent variables and the prior and approximate posterior are parameterized using diagonal Gaussian distributions (Babaeizadeh et al., 2018). We extend the VRNN formulation by proposing a hierarchical variant that uses multiple levels of latents per timestep.

Models leveraging a hierarchy of latents are known to be hard to optimize as they are required to backpropagate through a stack of stochastic latent variables, usually resulting in models that only make use of a small subset of the latents (Kingma & Welling, 2013; Maaløe et al., 2016; Sønderby et al., 2016b). We mitigate this problem by using a warmup regime for the KL loss (Sønderby et al., 2016a) and a dense connectivity pattern (Huang et al., 2017; Maaløe et al., 2019) between the input and latent variables. Specifically, each stochastic latent variable is connected to the input and to all subsequent stochastic levels in the hierarchy. Our empirical findings confirm that only with these techniques our model is able to take advantage of different layers in a latent hierarchy.

We validate our hierarchical VRNN in three datasets with varying levels of future uncertainty and realism: Stochastic Moving MNIST (Denton & Fergus, 2018), the BAIR Push Dataset (Ebert et al., 2017) and Cityscapes (Cordts et al., 2016). When compared to current state of the art models (Denton & Fergus, 2018; Lee et al., 2018), our approach performs favorably under several metrics. In particular for the BAIR Push Dataset, our hierarchical-VRNN shows an improvement of 44% in Video Fréchet Distance (FVD) (Unterthiner et al., 2018) and 9.8% in terms of LPIPS score (Zhang et al., 2018a) over SVG-LP (Denton & Fergus, 2018), the previous best VAE-based model. It also achieves a similar FVD than the SAVP VAE-GAN model (Lee et al., 2018), while showing a 11.2% improvement in terms of LPIPS over this baseline.

## 4.2. Related Work

Since Ranzato et al. (2014) and Srivastava et al. (2015a) applied techniques from the language modeling literature to model videos, there have been a number of papers that investigate different models for generating videos.

Initial video prediction approaches relied on deterministic models. For example, Ranzato et al. (2014) divided frames into patches and predicted their evolution in time given previous neighboring patches. In (Srivastava et al., 2015a) the authors used LSTM networks on pretrained image embeddings to predict the future. Similarly, Oh et al. (2015) used LSTMs on CNN representations to predict frames in Atari games when given the player actions.

ConvLSTMs (Xingjian et al., 2015) adapt the LSTM equations to spatial feature maps by replacing matrix multiplications with convolutions. They were originally used for precipitation nowcasting and are commonly used for video prediction.

Other works have proposed to disentangle the motion and context of the frames to generate (Villegas et al., 2017a; Tulyakov et al., 2018; Denton et al., 2017). They assume that a scene can be decomposed as multiple objects, which allows them to use a fixed representation for the background. Our approach does not follow this modeling assumption and instead tries to capture the stochasticity in predicting the visual future.

Autoregressive models (Kalchbrenner et al., 2017; Reed et al., 2017) approximate the full joint data distribution $p(x_1, x_2, ..., x_N)$ over pixels, which allows them to capture complex pixel dependencies but at the expense of making their inference mechanism slow and not scalable to high resolutions. Latent variable models using GANs (Goodfellow et al., 2014) were proposed in Vondrick et al. (2016b,a); Tulyakov et al. (2018). Training GAN video models is still an open research direction, as training is unstable and most models require auxiliary losses.

A more successful approach so far has been based on VAEs (Kingma & Welling, 2013; Rezende et al., 2014) and VRNN (Chung et al., 2015) models. SV2P (Babaeizadeh et al., 2018) proposed to capture sequence uncertainty in a single set of latent variables kept fixed for each predicted sequence. SVG (Denton & Fergus, 2018) adopted the VRNN formulation (Chung et al., 2015), introducing per-step latent variables (SVG-FP) and a variant with a learned prior (SVG-LP), which makes the prior at a certain timestep a function of previous frames. In recent work, SAVP (Lee et al., 2018) proposed to use the VAE-GAN (Larsen et al., 2015) framework for video, a hybrid model that offers a trade-off between the accuracy of VAEs and the visual results of GANs. Our model extends the VRNN formulation by introducing a hierarchy of latents to better approximate the data likelihood.

There are multiple works addressing hierarchical VAEs for non-sequential data (Ranganath et al., 2016; Maaløe et al., 2016; Sønderby et al., 2016a; Kingma et al., 2016). While hierarchical VAEs can model more flexible latent distributions, training them is usually difficult due to the multiple layers of conditional latents (Sønderby et al., 2016b), resulting in most latents being unused. Ladder Variational Autoencoders (Sønderby et al., 2016a) proposed a series of techniques to partially alleviate this issue. IAF (Kingma et al., 2016) used a similar architecture to Ladder VAEs and extended it with a novel normalizing flow. Recent work (Maaløe et al., 2019) has trained very deep hierarchical models that produce visually compelling samples. We extend hierarchical latent variable models to sequential data and apply them to the task of video prediction.

## 4.3. Preliminaries

We follow previous work in video prediction (Denton & Fergus, 2018). Given $D$ context frames $\mathbf{c} = (c_1, c_2, ..., c_D)$ and the $T$ following future frames $\mathbf{x} = (x_1, x_2, ..., x_T)$, our goal is to learn a generative model that maximizes the probability $p(\mathbf{x}|\mathbf{c})$.

VRNN follows the VAE formalism and introduces a set of latent variables $\mathbf{z} = (z_1, z_2, ..., z_T)$ to capture the variations in the observed variables $\mathbf{x}$ at each timestep $t$. It defines a *likelihood* model $p(\mathbf{x}|\mathbf{z}, \mathbf{c}) = \prod_{t=1}^{T} p(x_t|\mathbf{z}_{\leq \mathbf{t}}, \mathbf{x}_{<\mathbf{t}}, \mathbf{c})$ and a *prior* distribution $p(\mathbf{z}|\mathbf{c}) = \prod_{t=1}^{T} p(z_t|\mathbf{z}_{<\mathbf{t}}, \mathbf{x}_{<\mathbf{t}}, \mathbf{c})$ which are parametrized in an autoregressive manner; i.e. at each timestep observed and latent variables are conditioned on the past latent samples and observed frames. VRNN therefore uses a *learned prior* (Chung et al., 2015; Denton & Fergus, 2018). Taking into account the temporal structure of the data, the probability $p(\mathbf{x}, \mathbf{z}|\mathbf{c})$ is factorized as

$$p(\mathbf{x}, \mathbf{z}|\mathbf{c}) = \prod_{t=1}^{T} p(x_t|\mathbf{z}_{\leq \mathbf{t}}, \mathbf{x}_{<\mathbf{t}}, \mathbf{c}) p(z_t|\mathbf{z}_{<\mathbf{t}}, \mathbf{x}_{<\mathbf{t}}, \mathbf{c}). \tag{4.1}$$

Computing $p(\mathbf{x}|\mathbf{c})$ requires marginalizing over the latent variables $\mathbf{z}$, which is computationaly intractable. Instead, VRNN relies on Variation Inference (Jordan et al., 1999) and defines an *amortized approximate posterior* $q(\mathbf{z}|\mathbf{x}, \mathbf{c}) = \prod_{t=1}^{T} q(z_t|\mathbf{z}_{<\mathbf{t}}, \mathbf{x}_{\leq \mathbf{t}}, \mathbf{c})$ that approximates the *true posterior* distribution $p(\mathbf{z}|\mathbf{x},\mathbf{c})$. We then can derive the evidence lower bound (ELBO), a lower bound to the marginal log-likelihood $p(\mathbf{x}|\mathbf{c})$:

$$\log p(\mathbf{x}|\mathbf{c}) \geq \sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{z_{\leq t}}|\mathbf{x_{\leq t}},\mathbf{c})} \log p(x_t|\mathbf{z_{\leq t}}, \mathbf{x_{<t}}, \mathbf{c}) \\ - D_{KL}(q(z_t|\mathbf{z_{<t}}, \mathbf{x_{\leq t}}, \mathbf{c})||p(z_t|\mathbf{z_{<t}}, \mathbf{x_{<t}}, \mathbf{c})) \quad (4.2)$$

VRNN can be optimized to fit the training data by maximizing the ELBO using stochastic backpropagation and the reparameterization trick (Kingma & Welling, 2013; Rezende et al., 2014).

## 4.4. Hierarchical VRNN

We now introduce a hierarchical version of the VRNN model. At each timestep, we consider $L$ levels of latents variables $\mathbf{z_t} = (z_t^1, ..., z_t^L)$. We then further factorize the latent *prior* as

$$p(\mathbf{z_t}|\mathbf{z_{<t}}, \mathbf{x_{<t}}, \mathbf{c}) = \prod_{l=1}^{L} p(z_t^l|\mathbf{z_t^{<l}}, \mathbf{z_{<t}^l}, \mathbf{x_{<t}}, \mathbf{c}). \quad (4.3)$$

The sampling process of the latent variable $z_t^l$ now depends on the latent variables from previous time steps $\mathbf{z_{<t}^l}$ for that level and on the latent variables of the previous levels at the current timestep $\mathbf{z_t^{<l}}$. Similarly, we can write the *approximate posterior* as:

$$q(\mathbf{z_t}|\mathbf{z_{<t}}, \mathbf{x_{\leq t}}, \mathbf{c}) = \prod_{l=1}^{L} q(z_t^l|\mathbf{z_t^{<l}}, \mathbf{z_{<t}^l}, \mathbf{x_{\leq t}}, \mathbf{c}). \quad (4.4)$$

Using eq. 4.3 and eq. 4.4, we can rewrite the ELBO as

$$\log p(\mathbf{x}|\mathbf{c}) \geq \sum_{t=1}^{T} [\mathbb{E}_{q(\mathbf{z_t}|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})} \log p(x_t|\mathbf{z_{\leq t}}, \mathbf{x_{<t}}, \mathbf{c}) \\ - \sum_{l=1}^{L} D_{KL}(q(z_t^l|\mathbf{z_t^{<l}}, \mathbf{z_{<t}^l}, \mathbf{x_{\leq t}}, \mathbf{c})||p(z_t^l|\mathbf{z_t^{<l}}, \mathbf{z_{<t}^l}, \mathbf{x_{<t}}, \mathbf{c}))]. \quad (4.5)$$

Refer to Appendix 4.A.1 for a full derivation of the ELBO.

### 4.4.1. Dense Latent Connectivity

Training a hierarchy of latent variables is known to be challenging as it requires to backpropagate through multiple stochastic layers. Usually this results in models that only use one specific level of the hierarchy (Kingma & Welling, 2013; Maaløe et al., 2016; Sønderby et al., 2016b). To ease the optimization we use a dense connectivity pattern between latent levels

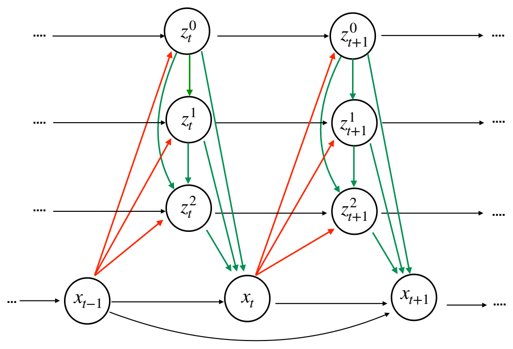


**Figure 4.2 – Graphical model with a learned prior and a dense latent connectivity pattern**. Arrows in red show the connections from the input at the previous timestep to current latent variables. Arrows in green highlight skip connections between latent variables and connections to outputs. Arrows in **black** indicate recurrent temporal connections. We empirically observe that this dense-connectivity pattern eases the training of latent hierarchies.

both for the prior and the approximate posterior, following Huang et al. (2017); Maaløe et al. (2019).

Figure 4.2 illustrates the dense connection of the learned prior (refer to Appendix 4.A.2 for the approximate posterior). For each latent level, the prior and posterior are implemented using recurrent neural networks which take as input a deterministic transformation of $x_{t-1}$ (red arrows in Figure 4.2), and to all the latent variables from the previous levels (green arrows in in Figure 4.2). In addition, each latent variable has a direct connection to the output variables $x_t$.

## 4.4.2. Model Parametrization

We now describe an instantiation of the VRNN model that we will use in the experiments, illustrated in Figure 4.3. First we compute features for each context frame and use them to initialize the hidden state of the prior/posterior/decoder networks, all of which have recurrent components. At a given timestep $t$, the model takes as input the latent variable samples $\mathbf{z_t} = (z_t^0, ..., z_t^L)$ with the embedding of the previously generated frame $x_{t-1}$ and outputs the next frame $\hat{x}_t$. During training we draw latent samples from the approximate posterior distribution $q(\mathbf{z_t}|\mathbf{z_{<t}}, \mathbf{x_{\leq t}}, \mathbf{c})$ and maximize the ELBO. To generate unseen sequences, we sample $\mathbf{z_t}$ from the learned prior $p(\mathbf{z_t}|\mathbf{z_{<t}}, \mathbf{x_{<t}}, \mathbf{c})$. Note that since we have multiple levels of conditional latents we use ancestral sampling to generate $\mathbf{z_t}$, i.e. we first sample from the top level of the hierarchy and we then sequentially sample the lower levels conditioning on the sampled values of the previous layers in the hierarchy.

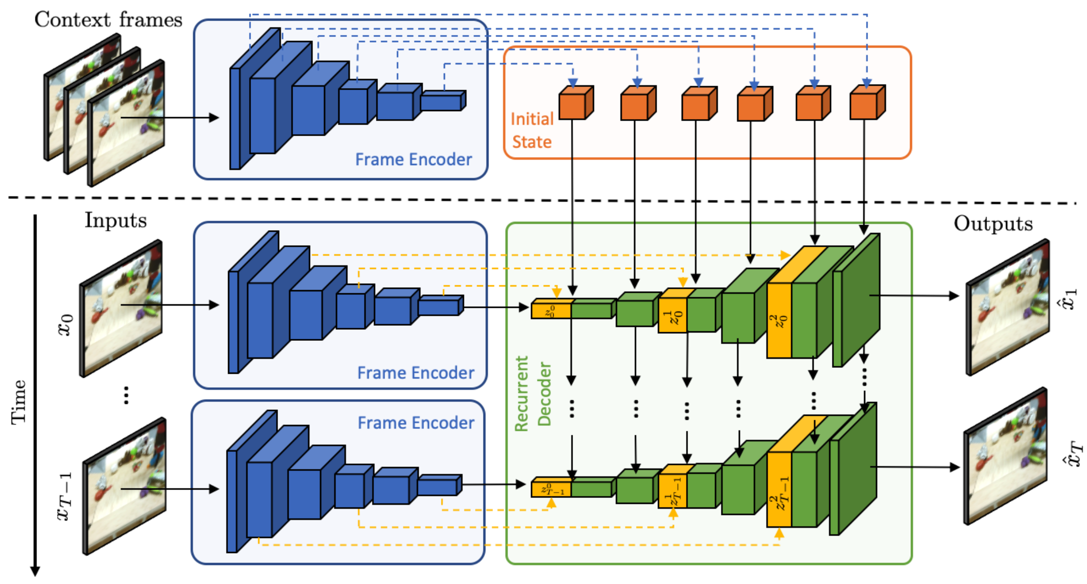


**Figure 4.3 – Model Parametrization.** Our model uses a CNN to encode frames individually. The representation of the context frames is used to initialize the states of the prior, posterior and likelihood networks, all of which use recurrent networks. At each timestep, the decoder receives an encoding of the previous frame, a set of latent variables (either from the prior or the posterior) and its previous hidden state and predicts the next frame in the sequence.

**Frame Encoder.** We use a 2D CNN with ResNet (He et al., 2016) blocks and max-pooling layers to represent input frames.

**Prior/Approximate Posterior.** We parametrize both the prior and the posterior as a hierarchy of diagonal Normal distributions $\mathcal{N}(\mu, \sigma)$, where the parameters $\mu$ and $\sigma$ are recurrent functions of samples from i) previous levels in the hierarchy and ii) the frame encoder features. Each level in the hierarchy operates at a different spatial resolution, with the top level features operating at a 1x1 resolution, i.e. not having a spatial topology. At a given timestep $t$, the parameters for a specific latent level $z_t^l$ are given by a ConvLSTM that consumes i) a previous hidden state, ii) samples from the previous levels in the hierarchy $z_t^{<l}$, iii) the feature map of a frame with the same spatial resolution as the ConvLSTM. For the prior network, the input frame embedding corresponds to the previously generated frame $x_{t-1}$, while for the posterior the input comes from the frame to generate $x_t$.

**Likelihood/Frame Decoder.** At each timestep $t$, the decoder takes a representation of the previously generated frame $x_{t-1}$ and the samples $\mathbf{z_t} = (z_t^1, ..., z_t^L)$ and generates $x_t$ according to $p(x_t | \mathbf{z_t}, \mathbf{x_{<t}}, \mathbf{c})$. The decoder consists of ConvLSTMs interleaved with transposed convolutions that upscale the feature maps back to the input resolution.

**Initial State.** The initial states of our prior, posterior and decoder/likelihood models are functions of the context. We use small CNNs to initialize each of the ConvLSTMs layers used in the VAE components.

## 4.5. Experiments

All our models are trained with Adam (Kingma & Ba, 2014) and a batch size of $b = 128$ on Nvidia DGX-1s. We use a learning rate warmup (Goyal et al., 2017) starting with an initial learning rate $\lambda = 2\text{e-}5$ that is linearly increased at each timestep until reaching $\lambda = 1.6\text{e-}4$ in 5 epochs. We use $\beta_1 = 0.5$ and $\beta_2 = 0.9$ and weight decay $\delta = 1\text{e-}4$. We train the autoregressive components of our models using teacher forcing (Williams & Zipser, 1989).

Our models are also trained using beta warmup (Sønderby et al., 2016a), which consists in gradually increasing the weight of the KL divergence in the ELBO, turning the model from an unregularized Autoencoder into a VAE progressively. VAEs trained with beta warmup usually encode more information in the latent variables. Refer to the Appendix 4.B for a complete description of our models.

### 4.5.1. Ablation Study

We first investigate the importance of each VRNN component, namely the likelihood, the prior and the posterior. We focus on the BAIR Push dataset (Ebert et al., 2017) with 64x64 color sequences of a robotic arm interacting with children toys in a sandbox. Similarly to previous works (Lee et al., 2018), we use trajectories 256 to 511 as our test set and the rest for training, resulting in the 43264 train and 256 test sequences. At training we randomly subsample 12 frames from each train sequence, use the first 2 frames as the context, and learn to predict the remaining 10 frames. To evaluate the different model variations, we report the training objective (ELBO) obtained for the training set and the test set.

**Scaling the Likelihood Model.** We assess the importance of the likelihood model $p(x_t|\mathbf{z_{\leq t}}, \mathbf{x_{<t}}, \mathbf{c})$. For this purpose, we build a VRNN with a single level of latent variables and modify the number of ConvLSTM layers in the decoder. Our aim is to investigate whether increasing the capacity of the mapping from latent to the observations results in better predictions.

In this experiment, our baseline likelihood model has one LSTM at 1x1 spatial resolution. We then gradually replace convolutional layers in the decoder with ConvLSTM layers, which

| Model | Parameters | Train/Test ELBO(↓) |
|---|---|---|
| 1-ConvLSTM | 62.22M | 3237/3826 |
| 3-ConvLSTM | 86.64M | 1948/2355 |
| 6-ConvLSTM | 93.81M | 1279.21/1731.31 |
| + higher capacity | 194.15M | 1113.31/1589.72 |

**Table 4.1 – Ablation - Likelihood** We compare models with different number of recurrent layers for the likelihood network. We observe that the model performance increases monotonically as we add more ConvLSTMs. We further increase the size of the recurrent hidden states for the 6-ConvLSTM model (+ higher capacity variant), also leading to a better data fit. These results suggest that current video prediction models might underfit the data because of reduced likelihood capacity.

increases the amount of information that can be carried from previous timesteps and, by extension, also increases the overall likelihood model capacity. We compare to a model with 3 ConvLSTM layers at resolutions 1x1, 4x4 and 8x8 and a model with 6 ConvLSTM layers at 1x1, 4x4, 8x8, 16x16, 32x32 and 64x64. Additionally, we also increase the size of the ConvLSTM layers for the model with 6 layers as another way of adding capacity.

Results can be found in Figure 4.1. We observe that, as a general trend, both the training and test ELBO decrease as we increase the model capacity, which suggests that current video prediction models might operate in an underfitting regime and benefit from higher capacity decoders.

**More Flexible Prior and Posterior.** We now investigate the importance of having more flexible prior and approximate posterior distributions and augment the 6-ConvLSTM VRNN model with a hierarchy of latent variables. For all models, we fix the frame encoder and likelihood model[1] and change the networks that estimate the learned prior $p(\mathbf{z_t}|\mathbf{z_{<t}}, \mathbf{x_{<t}}, \mathbf{c})$, and the approximate posterior $q(\mathbf{z_t}|\mathbf{z_{<t}}, \mathbf{x_{\leq t}}, \mathbf{c})$ over the latent variables. All these models use a dense connectivity pattern and beta warmup.

We compare a VRNN baseline with a single level of latents with no spatial topology, with a model with two levels of latents at resolutions 1x1 and 8x8 (1-8), three levels of latents at 1x1, 8x8 and 32x32 (1-8-32), and four levels of latents (1-8-16-32) in the top half of Table 4.2. All models are trained with beta warmup and dense latent connectivity. We observe that in general adding more levels of latents with higher resolution reduces the train and test ELBOs, supporting the hypothesis that a more flexible prior and posterior leads to a better data fit. However, we observe diminishing returns past 3 levels, as our 1-8-16-32 model does

1. To add the multiple levels of latents in the decoder we need to modify the likelihood network and slightly increase the number of parameters. However, most (> 85%) of the added capacity when adding a new level of latents goes towards the prior and posterior networks.

| Model | Parameters | Train/Test ELBO (↓) |
|---|---|---|
| 1 | 166.55M | 1141.85/1536.93 |
| 1-8 | 220.60M | 989.39/1313.02 |
| 1-8-32 | 230.74M | **883.10/1162.24** |
| 1-8-16-32 | 245.19M | 956.63/1256.22 |
| Naive Training | 224.18M | 1127.33/1440.58 |
| BW | 224.18M | 1101.39/1440.62 |
| Dense | 230.74M | 1182.60/1547.05 |
| BW and Dense | 230.74M | **883.10/1162.24** |

**Table 4.2 – Ablation - Hierarchy of Latents** *Top half:* We compare a VRNN baseline with a single level of latents with no spatial topology (1), a model with two levels of latents at resolutions 1x1 and 8x8 (1-8), our full model with three levels of latents at 1x1, 8x8 and 32x32 (1-8-32), and a model with 4 levels of latents (1-8-16-32). Adding more levels of latents leads to a better fit, with reduced ELBOs. However, adding extra levels of latents without increasing the spatial resolution reduces the performance of the model due to the difficulties in training hierarchical latent variable models. *Bottom half:* To highlight the difficulties in training hierarchies of latents, we investigate the effects of using beta warmup (BW) (Sønderby et al., 2016a) and having a dense connectivity (Dense) between latents when training the 1-8-32 model. Without these techniques the hierarchy of latents does not bring any benefit compared to the VRNN with 1 level of latent.

not outperform the 3 layers model. We attribute this to the difficulties in training deep hierarchies of latents, which remains a challenging optimization problem.

To further highlight the difficulties in training hierarchies of latents, we investigate the importance of using beta warmup (Sønderby et al., 2016a) and having a dense connectivity between latents.

The results of this experiment can be found in the bottom half of Table 4.2. We observe that these techniques are required to make our 1-8-32 model make use of the hierarchy of latents and improve upon the single level model.

This is analyzed in more detail in Figure 4.4, where we visualize the KL between the prior and the posterior distributions for the test sequences of the BAIR Push dataset for the 1-8-32 model and the variant without warmup or dense connectivity (Naive training). We consider a channel to be active if its average KL is higher than 0.01 following Maaløe et al. (2019), and consider that a unit with a KL higher than 0.15 is maximally activated. We observe that without these techniques the model only uses a few latents of the top level in the hierarchy. However, when using beta warmup and a dense connectivity most of the latents are active across levels.

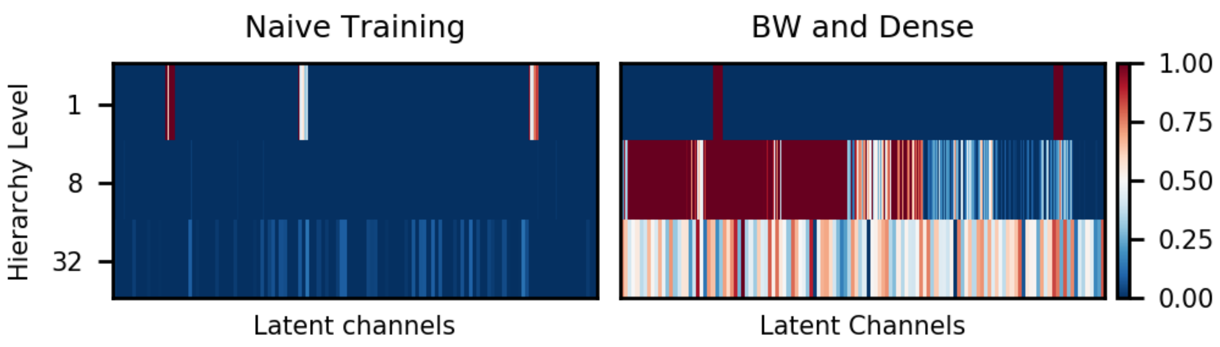


**Figure 4.4 – Average normalized KL divergence per latent channel.** We visualize the mean normalized KL for each latent channel for models from Table 4.2. Without beta warmup and dense connectivity the hierarchy of latents is underutilized, with most information being encoded in a few latents of the top level. In contrast, the same model with these techniques utilizes all latent levels.

### 4.5.2. Comparisons to Previous Approaches

Next, we compare our single latent level VRNN (Ours w/o Hier), and our hierarchical VRNN with 3 levels of latents (Ours w/ Hier) to previous video approaches on Stochastic Moving MNIST (Denton & Fergus, 2018), BAIR Push (Ebert et al., 2017) and the Cityscapes (Cordts et al., 2016) datasets.

**Evaluation and Metrics.** Defining evaluation metrics for video prediction is an open research question. In general we want models to predict sequences that are *plausible*, look *realistic* and *cover* all possible outcomes. Unfortunately, we are not aware of any metric that reflects all these aspects.

To measure *coverage* and *plausibility* we adopt the evaluation protocol proposed in (Denton & Fergus, 2018; Lee et al., 2018). For each ground truth test sequence, we sample $N$ random predictions from the model which are conditioned on the test sequence initial frames. Then we find the sample that best matches the ground truth sequence according to a given metric and report that metric value. Some common metric choices are Mean-Square Error (MSE), Structural Similarity (SSIM) Wang et al. (2004b) or Peak Signal-to-Noise Ratio (PSNR). In practice, these metrics have been shown to not correlate well with human judgement as they tend to prefer blurry predictions over sharper but imperfect generations (Zhang et al., 2018a; Lee et al., 2018; Unterthiner et al., 2018). LPIPS (Zhang et al., 2018a), on the other hand, is a perceptual metric that employs CNN features and has better correlation to human judgment. For this evaluation we choose to produce $N = 100$ samples following previous work and use SSIM and LPIPS as metrics. We have empirically observed that using 100

| Model | FVD (↓) | LPIPS (↓) | SSIM (↑) |
|---|---|---|---|
| SVG-LP | 90.81 | 0.153 ± 0.03 | 0.668 ± 0.04 |
| Ours w/o hier | 63.81 | **0.102** ± **0.04** | **0.763** ± **0.09** |
| Ours w/ hier | **57.17** | **0.103** ± **0.03** | **0.760** ± **0.08** |

**Table 4.3 – Stochastic Moving MNIST comparison.** We compute the FVD metric between samples from different models and test sequences as well as the average LPIPS and SSIM of the best sample for each test sequence. Our models outperform the SVG-LP baseline on all metrics by a significant margin. While our model with hierarchical latent variables obtains a better FVD score, both variants obtain comparable results in this relatively simple dataset.

samples the results stay fairly consistent across different samplings. We report the metric average over the test set.

Additionally, we also use the recently proposed Fréchet Video Distance (FVD), which measures sample *realism*. FVD uses features from a 3D CNN and has also been shown to correlate well with human perception Unterthiner et al. (2018). FVD compares populations of samples to assert whether they were both generated by the same distribution (it does not compare pairs of ground truth/generated frames directly). We form the ground truth population by using all the test sequences with their context. For the predicted population we randomly sample one video out of the $N$ generated for each test sequence. We repeat this process 5 times and report the mean of the FVD scores obtained, which stay fairly stable across samplings.

**Stochastic Moving MNIST.** Stochastic Moving MNIST is a synthetic dataset proposed in (Denton & Fergus, 2018) which consists of black and white sequences of MNIST digits moving over a black background and bouncing off the frame borders. As opposed to the original Moving MNIST dataset (Srivastava et al., 2015a) with completely deterministic motion, Stochastic Moving MNIST has uncertain digit trajectories - the digits bounce off the border with a random new trajectory. We train two variants of our model and compare to the SVG-LP baseline (Denton & Fergus, 2018), for which we use a pretrained model from the official codebase. All models are trained using 5 frames of context and 10 future frames to predict. To evaluate the models, we follow the procedure in (Denton & Fergus, 2018) described in section 4.5.2.

We report the results of the experiment in Table 4.3. We observe that both versions of our model (with/out the latent hierarchy) outperform the SVG-LP baseline by a significant margin on all metrics. Note that LPIPS and FVD might not be suited to this dataset as they use features from CNNs trained on real world images, but we report them for completeness. Visually, our samples (found in Appendix 4.D) reproduce the digits more faithfully with reduced degradation over time. There are small differences between the two versions of our

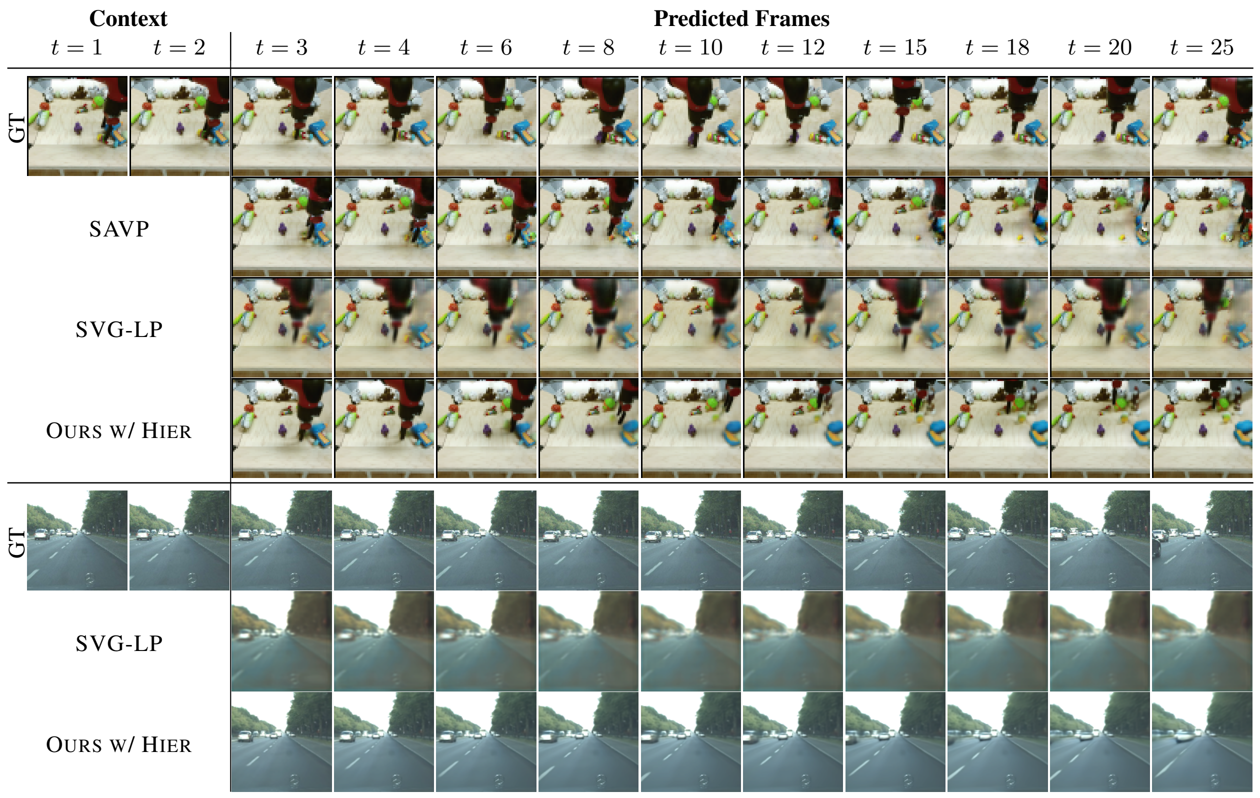
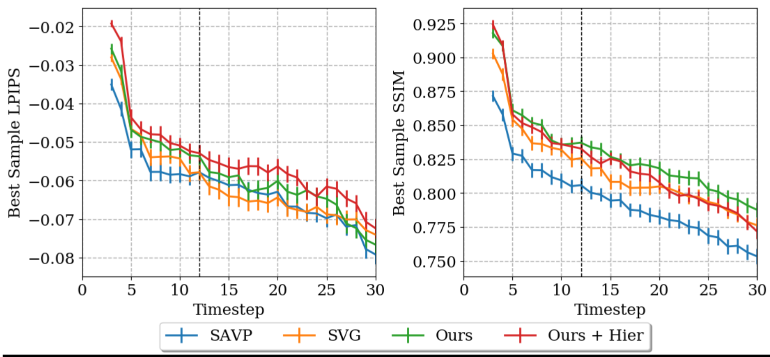


**Figure 4.5 – Selected Samples for BAIR Push and Cityscapes.** We show a sequence for BAIR Push and Cityscapes and random generations from our model and baselines. On BAIR Push we observe that the SAVP predictions are crisp but sometimes depict inconsistent arm-object interactions. SVG-LP produces blurry predictions in uncertain areas such as occluded parts of the background or those showing object interactions. Our model generates plausible interactions with reduced blurriness relatively to SVG-LP. On Cityscapes, the SVG-LP baseline is unable to model any motion. Our model, using a hierarchy of latents, generates more visually compelling predictions. More samples can be found in the Appendix.

model, suggesting that the extra expressiveness of the hierarchical model is not necessary in this synthetic dataset.

**BAIR Push.** We compare our VRNN models to SVG-LP (Denton & Fergus, 2018) and SAVP (Lee et al., 2018). We use their official implementations and pretrained models to reproduce their results. We use the experimental setup of previous works (Denton & Fergus, 2018; Lee et al., 2018), using 2 context frames and generating 28 frames.

Results can be found on Figure 4.6. When the robotic arm is interacting with an object, SVG-LP tends to generate blurry predictions characterized by a high FVD score. SAVP exhibits a lower FVD as it produces more realistically looking predictions. However, SAVP does not have a better coverage of the ground truth sequences compared to SVG-LP as measured by LPIPS and SSIM. By inspecting the SAVP samples we notice that the SAVP generations tend to be sharper but sometimes they exhibit temporal inconsistencies or implausible interactions



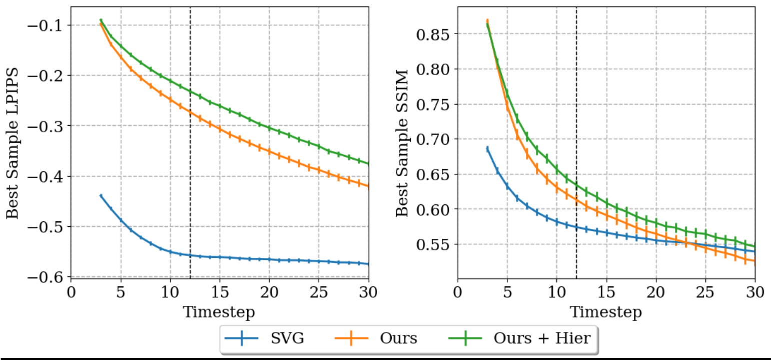
| Model | FVD (↓) | LPIPS (↓) | SSIM (↑) |
|---|---|---|---|
| SVG-LP | 256.62 | $0.061 \pm 0.03$ | $0.816 \pm 0.07$ |
| SAVP | **143.43** | $0.062 \pm 0.03$ | $0.795 \pm 0.07$ |
| Ours w/o Hier | 149.22 | $0.058 \pm 0.03$ | $\mathbf{0.829 \pm 0.06}$ |
| Ours w/ Hier | **143.40** | $\mathbf{0.055 \pm 0.03}$ | $0.822 \pm 0.06$ |

**Figure 4.6 – BAIR Push - Results**. *Left:* We show the evolution in time of the Average LPIPS and SSIM of the best predicted sample per test sequence. *Right:* We report the Average FVD, SSIM and LPIPS of the best sample for each test sequence. Compared to SVG-LP, both our model with a single level of latents and the hierarchical models improve all metrics. Compared to SAVP, we obtain better LPIPS and SSIM. Our model with a single level of latents performs better in SSIM but worse on perceptual metrics. When adding the hierarchy of latents, our model matches the FVD of SAVP and improves the LPIPS, indicating samples of similar visual quality and better coverage of the ground-truth sequences.

(see Figure 4.5). Our w/o Hier VRNN models obtain better scores than SVG-LP, the previous best VAE-type model. This supports the importance of having a high-capacity likelihood model. In addition, our hierarchical VRNN further improves both the FVD and LPIPS metrics, suggesting that the hierarchy of latents helps modeling the data In particular, our hierarchical VRNN shows an improvement of 44% in terms of FVD and 9.8% in terms of LPIPS over SVG-LP, the previous best VAE-based model. It also achieves a similar FVD than the SAVP GAN-VAE model, while outperforming it in terms of LPIPS by 11.2%.

**Cityscapes.** The Cityscapes dataset contains sequences recorded from a car driving around multiple cities under different conditions. Cityscapes is a challenging dataset - while contiguous frames are locally similar, uncertainty grows significantly with time. Compared to previous datasets, the backgrounds in Cityscapes do not stay constant across time.

We consider sequences with 30 frames from the training set cities for a total of 1877 train sequences and randomly select 256 test sequences. We use 2 context and 10 prediction frames to train the models. At test time we predict 28 frames following the BAIR Push experimental protocol. We preprocess the videos by taking a 1024x1024 center crop of the original sequences and resizing them to 128x128 pixels. For evaluating the models we use



| Model | FVD (↓) | LPIPS (↓) | SSIM (↑) |
|---|---|---|---|
| SVG-LP | 1300.26 | $0.549 \pm 0.06$ | $0.574 \pm 0.08$ |
| Ours w/o Hier | 682.08 | $0.304 \pm 0.10$ | $0.609 \pm 0.11$ |
| Ours w/ Hier | **567.51** | $\mathbf{0.264 \pm 0.07}$ | $\mathbf{0.628 \pm 0.10}$ |

**Figure 4.7 – Cityscapes - Quantitative Results** We report FVD, SSIM and LPIPS scores on Cityscapes at 128x128 resolution for the SVG-LP Denton & Fergus (2018) baseline and two variants of our model. Increasing the capacity of the likelihood model brings an improvement in all metrics over the SVG baseline. When adding a hierarchy of latents we observe further improvements, validating its usefulness. Even though SVG matches our models in SSIM at later timesteps, this does not correlate well with human judgement, as the generated SVG samples show more blurriness (see Figure 4.5).

the standard setup where we generate 100 samples per test sequence and report FVD, SSIM and LPIPS metrics. Since none of the baselines from previous experiments are trained on Cityscapes, we use the official SVG implementation (that defines models with 128x128 inputs) and train a SVG-LP model. We train all models for 100 epochs.

Results for this experiment can be found in Figure 4.7. SVG-LP has trouble modelling motion in the dataset, usually predicting a static image similar to the last context frame. In contrast, our model without a hierarchy of latents is able to model the changing scene. When adding hierarchical latents our model is able to capture more fine-grained details, and as a result, it produces more visually appealing samples with a boost in all metrics. We note that the SSIM scores for SVG-LP match those of our models at later timesteps in the prediction, however this does not translate to better samples as can be seen in Figure 4.5 or in the Appendix. This further indicates that SSIM might not be suitable to evaluate video prediction models.

## 4.6. Conclusions

We propose a hierarchical VRNN for video prediction that features an improved likelihood model and a hierarchy of latent variables. Our approach compares favorably to current state of the art models in terms of the Fréchet Video Distance, LPIPS and SSIM metrics, producing visually appealing and coherent samples. Our results demonstrate that current video prediction models benefit from increased capacity, opening the door to further gains with bigger and more flexible generative models.

**Acknowledgements.** We acknowledge the support from the FAIR and Mila teams. L.C. was supported by an IVADO PhD fellowship.

## 4.A. Appendix - Hierarchical VRNN

### 4.A.1. ELBO Derivation

We start from the ELBO for VRNNs (Chung et al., 2015):

$$\log p(\mathbf{x}|\mathbf{c}) \geq \sum_{t=1}^{T} \mathbb{E}_{q(z_t|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})} \log p(x_t|\mathbf{z_{\leq t}}, \mathbf{x_{<t}}, \mathbf{c}) - D_{KL}(q(z_t|\mathbf{z_{<t}}, \mathbf{x_{\leq t}}, \mathbf{c})||p(z_t|\mathbf{z_{<t}}, \mathbf{x_{<t}}, \mathbf{c})) \tag{4.6}$$

Recall we defined $\mathbf{z_t} = (z_t^1, ..., z_t^L)$ and factorized the prior as:

$$p(\mathbf{z_t}|\mathbf{z_{<t}}, \mathbf{x_{<t}}, \mathbf{c}) = \prod_{l=1}^{L} p(z_t^l|\mathbf{z_t^{<l}}, \mathbf{z_{<t}^l}, \mathbf{x_{<t}}, \mathbf{c}). \tag{4.7}$$

And the posterior:

$$q(\mathbf{z_t}|\mathbf{z_{<t}}, \mathbf{x_{\leq t}}, \mathbf{c}) = \prod_{l=1}^{L} q(z_t^l|\mathbf{z_t^{<l}}, \mathbf{z_{<t}^l}, \mathbf{x_{\leq t}}, \mathbf{c}). \tag{4.8}$$

We then substitute these terms in the VRNN ELBO, first looking at the reconstruction term inside the summation over time:

$$\begin{aligned}
&\sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{z_t}|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})} \log p(x_t|\mathbf{z_{\leq t}}, \mathbf{x_{<t}}, \mathbf{c}) = \\
&\qquad \sum_{t=1}^{T} \mathbb{E}_{q(z_t^1,...,z_t^L|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})} \log p(x_t|z_t^1, ..., z_t^L, \mathbf{z_{<t}}, \mathbf{x_{<t}}, \mathbf{c}) = \qquad (4.9)\\
&\qquad \sum_{t=1}^{T} \mathbb{E}_{q(z_t^1|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})...q(z_t^L|\mathbf{z_t^{<L}},\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})} \log p(x_t|z_t^1, ..., z_t^L, \mathbf{z_{<t}}, \mathbf{x_{<t}}, \mathbf{c})
\end{aligned}$$

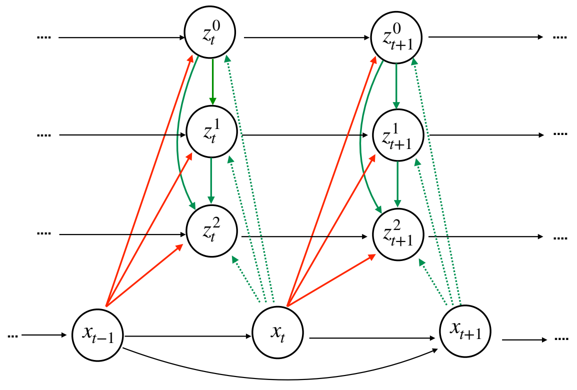


**Figure 4.8 – Schematic view of the approximate posterior with the dense-connectivity pattern**. Arrows in red show the connections from the input at the previous timestep to current latent variables. Arrows in green highlight skip connections between latent variables to outputs. Arrows in **black** indicate recurrent temporal connections. We empirically observe that this dense-connectivity pattern eases the training of latent hierarchy.

And then looking at the summation of KL divergences:

$$
\begin{aligned}
-\sum_{t=1}^{T}\mathbb{E}_{q(\mathbf{z_t}|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})}\log\frac{q(\mathbf{z_t}|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})}{p(\mathbf{z_t}|\mathbf{z_{<t}},\mathbf{x_{<t}},\mathbf{c})} &= \\
-\sum_{t=1}^{T}\mathbb{E}_{q(z_t^1,\dots,z_t^L|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})}\log\frac{q(z_t^1,\dots,z_t^L|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})}{p(z_t^1,\dots,z_t^L|\mathbf{z_{<t}},\mathbf{x_{<t}},\mathbf{c})}& \\
= -\sum_{t=1}^{T}\mathbb{E}_{q(z_t^1|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})\dots q(z_t^L|\mathbf{z_t^{<L}},\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})}\log\frac{q(z_t^1|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})\dots q(z_t^L|\mathbf{z_t^{<L}},\mathbf{z_{<t}},\mathbf{x_{\leq t}})}{p(z_t^1|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})\dots q(z_t^L|\mathbf{z_t^{<L}},\mathbf{z_{<t}},\mathbf{x_{<t}})}& \\
= -\sum_{t=1}^{T}\mathbb{E}_{q(z_t^1|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})\dots q(z_t^L|\mathbf{z_t^{<L}},\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})}\log\frac{q(z_t^1|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})\dots q(z_t^L|\mathbf{z_t^{<L}},\mathbf{z_{<t}},\mathbf{x_{\leq t}})}{p(z_t^1|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})\dots q(z_t^L|\mathbf{z_t^{<L}},\mathbf{z_{<t}},\mathbf{x_{<t}})}& \\
\text{(by definition of conditional KL divergence)}& \\
= -\sum_{t=1}^{T}\sum_{l=1}^{L}D_{KL}((q(z_t|\mathbf{z_{<t}},\mathbf{x_{\leq t}},\mathbf{c})||p(z_t|\mathbf{z_{<t}},\mathbf{x_{<t}},\mathbf{c}))&
\end{aligned}
\tag{4.10}
$$

Adding both terms together we obtain the ELBO defined in eq. 4.5.

## 4.A.2. Posterior Dense Connectivity

Figure 4.8 illustrates the dense connection of the approximate posterior. For each latent variable has a deterministic connection to $x_{t-1}$ (red arrows in Figure 4.2), in addition to all the latent variables from the layers below (green arrow in in Figure 4.2). Finally , each latent variable has a direct connection to the output variables $x_t$, corresponding to the inference path.

## 4.B. Model Specification

We specify the architecture used for the 64x64 model. Convolutional layers in our model use 3x3 kernels with stride s = 1 and padding p = 1 unless otherwise specified. We use modified Resnet blocks made up of two groups of ReLU + Conv2D + GroupNorm. GroupNorm layers use g = 16 groups. Transposed Convolutions use 4x4 kernels with stride s = 2 and padding p = 1, which upscales 2x the input tensor. ConvLSTM layers use 3x3 kernels with stride s = 1 and padding p = 1 and GroupNorm.

| Layers | Details |
|---|---|
| Conv2D | input → 64 |
| ResNet Block | 64 → 64 |
| MaxPool | 2x2, s = 2 |
| ResNet Block | 64 → 128 |
| ResNet Block | 128 → 128 |
| MaxPool | 2x2, s = 2 |
| ResNet Block | 128 → 256 |
| ResNet Block | 256 → 256 |
| MaxPool | 2x2, s = 2 |
| ResNet Block | 256 → 512 |
| ResNet Block | 512 → 512 |
| MaxPool | 2x2, s = 2 |
| ResNet Block | 512 → 512, ks = 4 |
| ResNet Block | 512 → 512 |

**Table 4.4 – Frame Encoder Architecture**

| Layers | Details |
|---|---|
| ConvLSTM | 512 → 512, ks = 4 |
| UpConv | 512 → 512, scale = 4 |
| ConvLSTM | 512 → 512 |
| UpConv | 512 → 512 |
| ConvLSTM | 512 → 512 |
| UpConv | 512 → 256 |
| ConvLSTM | 256 → 256 |
| UpConv | 256 → 128 |
| ConvLSTM | 128 → 128 |
| UpConv | 128 → 64 |
| ConvLSTM | 64 → 64 |
| Conv2D + GroupNorm + ReLU | 64 → 64 |
| Conv2D | 64 → input |

**Table 4.5 – Likelihood/Decoder Architecture**

| Level | Layer | Details |
|---|---|---|
| 1x1 | Conv2D + GroupNorm | 128 → 128, ks = 1 |
| | ConvLSTM | 128 → 128 |
| | Conv2D + GroupNorm | 128 → 128x2, ks = 1 |
| 8x8 | Conv2D + GroupNorm | 512 → 512, ks = 1 |
| | ConvLSTM | 512 → 512 |
| | Conv2D + GroupNorm | 512 → 512x2, ks = 1 |
| 32x32 | Conv2D + GroupNorm | 512 → 512, ks = 1 |
| | ConvLSTM | 512 → 512 |
| | Conv2D + GroupNorm | 512 → 512x2, ks = 1 |

**Table 4.6 – Prior/Posterior Architecture**

| Level | Layer | Details |
|---|---|---|
| 1x1 | Conv2D + GroupNorm + ReLU | 512 → 512, ks = 1 |
| | Conv2D + GroupNorm | 512 → 512x2, ks = 1 |
| 4x4 | Conv2D + GroupNorm + ReLU | 512 → 512, ks = 1 |
| | Conv2D + GroupNorm | 512 → 512x2, ks = 1 |
| 8x8 | Conv2D + GroupNorm + ReLU | 512 → 512, ks = 1 |
| | Conv2D + GroupNorm | 512 → 512x2, ks = 1 |
| 16x16 | Conv2D + GroupNorm + ReLU | 256 → 256, ks = 1 |
| | Conv2D + GroupNorm | 256 → 256x2, ks = 1 |
| 32x32 | Conv2D + GroupNorm + ReLU | 128 → 128, ks = 1 |
| | Conv2D + GroupNorm | 128 → 128x2, ks = 1 |
| 64x64 | Conv2D + GroupNorm + ReLU | 64 → 64, ks = 1 |
| | Conv2D + GroupNorm | 64 → 64x2, ks = 1 |

**Table 4.7 – Initial State Network Architecture**

## 4.C. PredNet Comparison

We additionally compare to PredNet on Cityscapes using the official implementation. Note that PredNet is deterministic and can't model future uncertainty. The model is only able to correctly predict a few timesteps before becoming blurry, as the uncertainty increases with time. PredNet obtained a FVD score of 1079.19 and a LPIPS score of 0.397, while ours with the hierarchical model are 567.51 and 0.264 respectively (lower is better).

## 4.D. Additional Samples

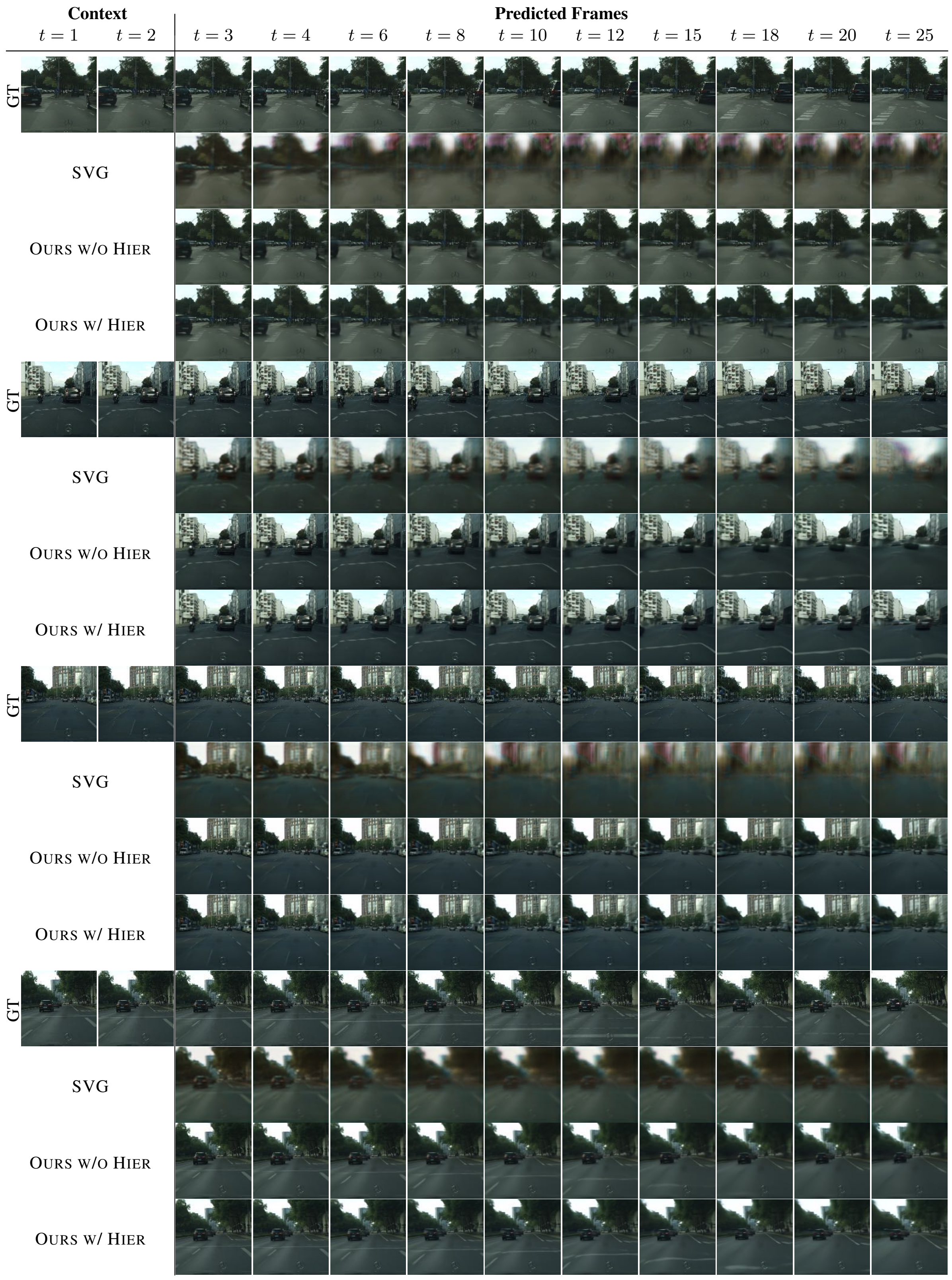


**Figure 4.9 – Additional samples on the Cityscapes dataset (1)**

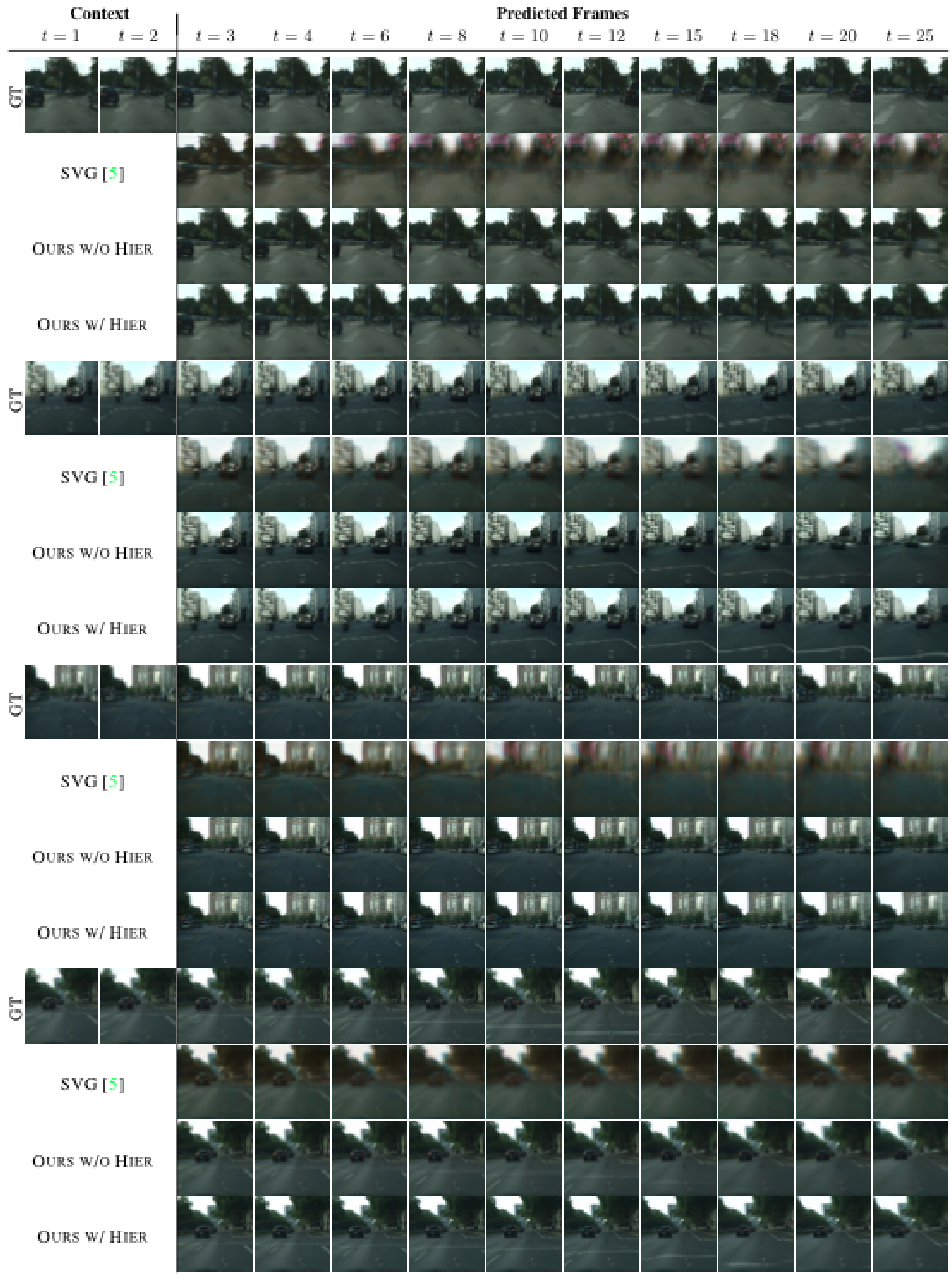


**Figure 4.10 – Additional samples on the Cityscapes dataset (2)**

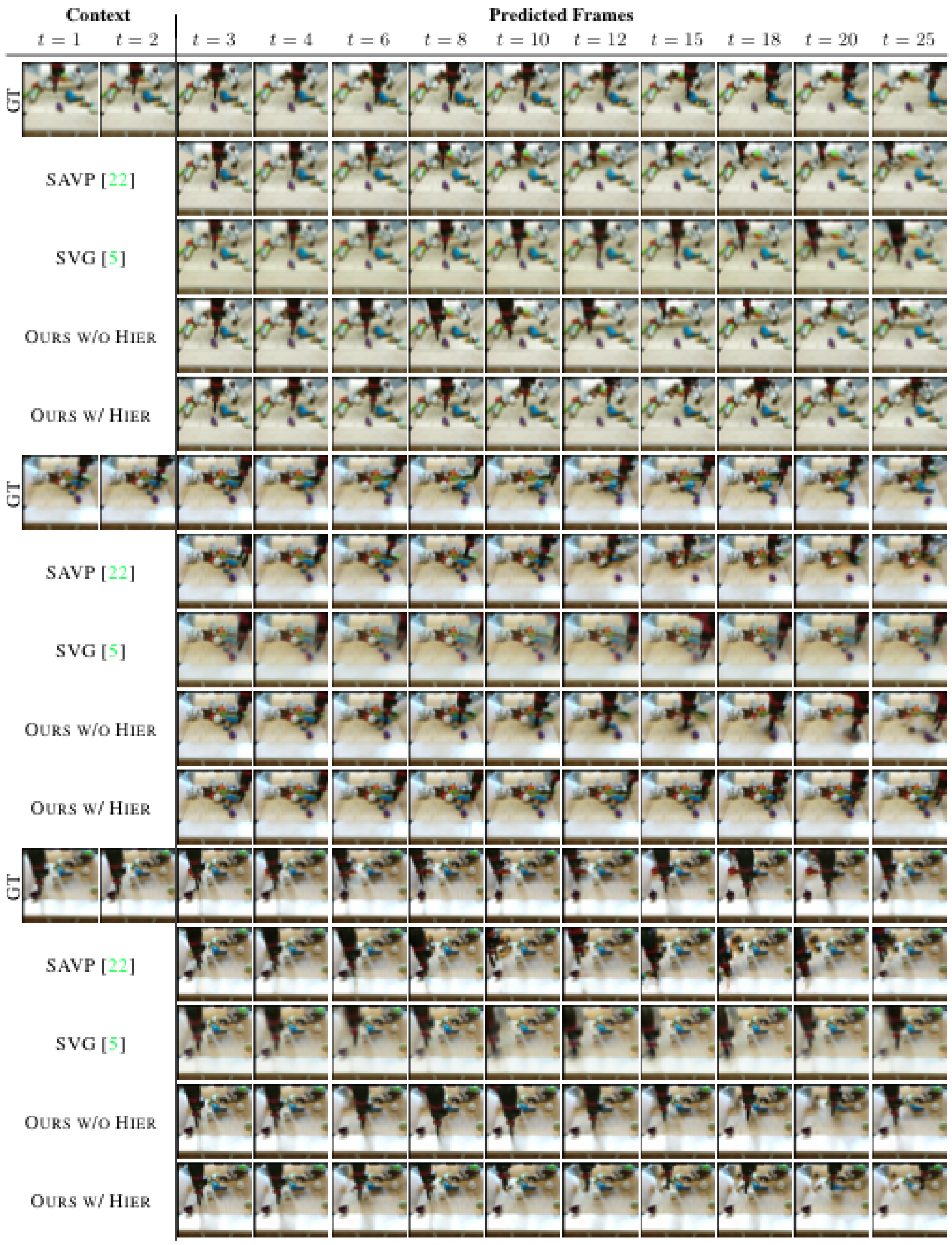


**Figure 4.11 – Additional samples on the BAIR Push dataset (1)**

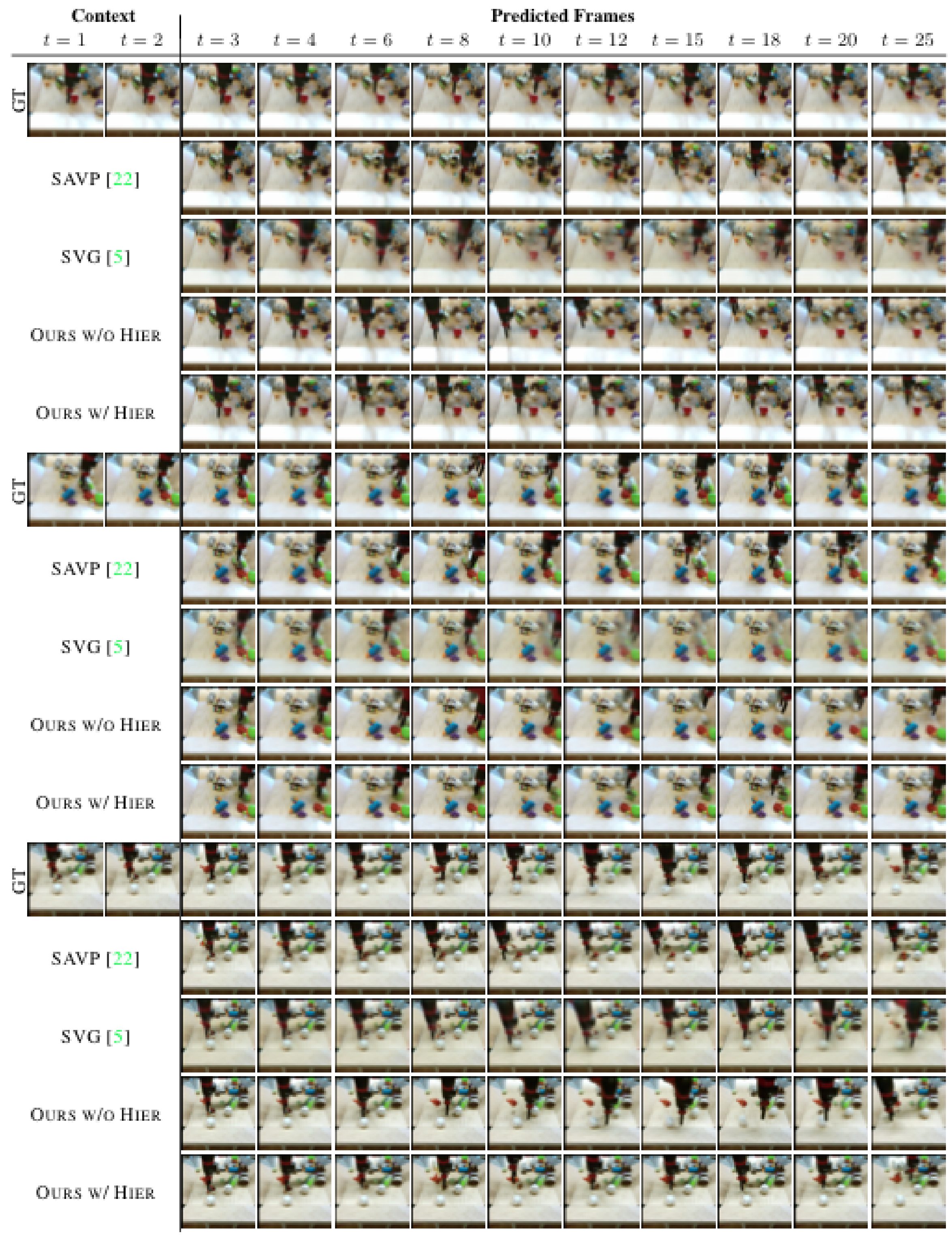


**Figure 4.12 – Additional samples on the BAIR Push dataset (2)**

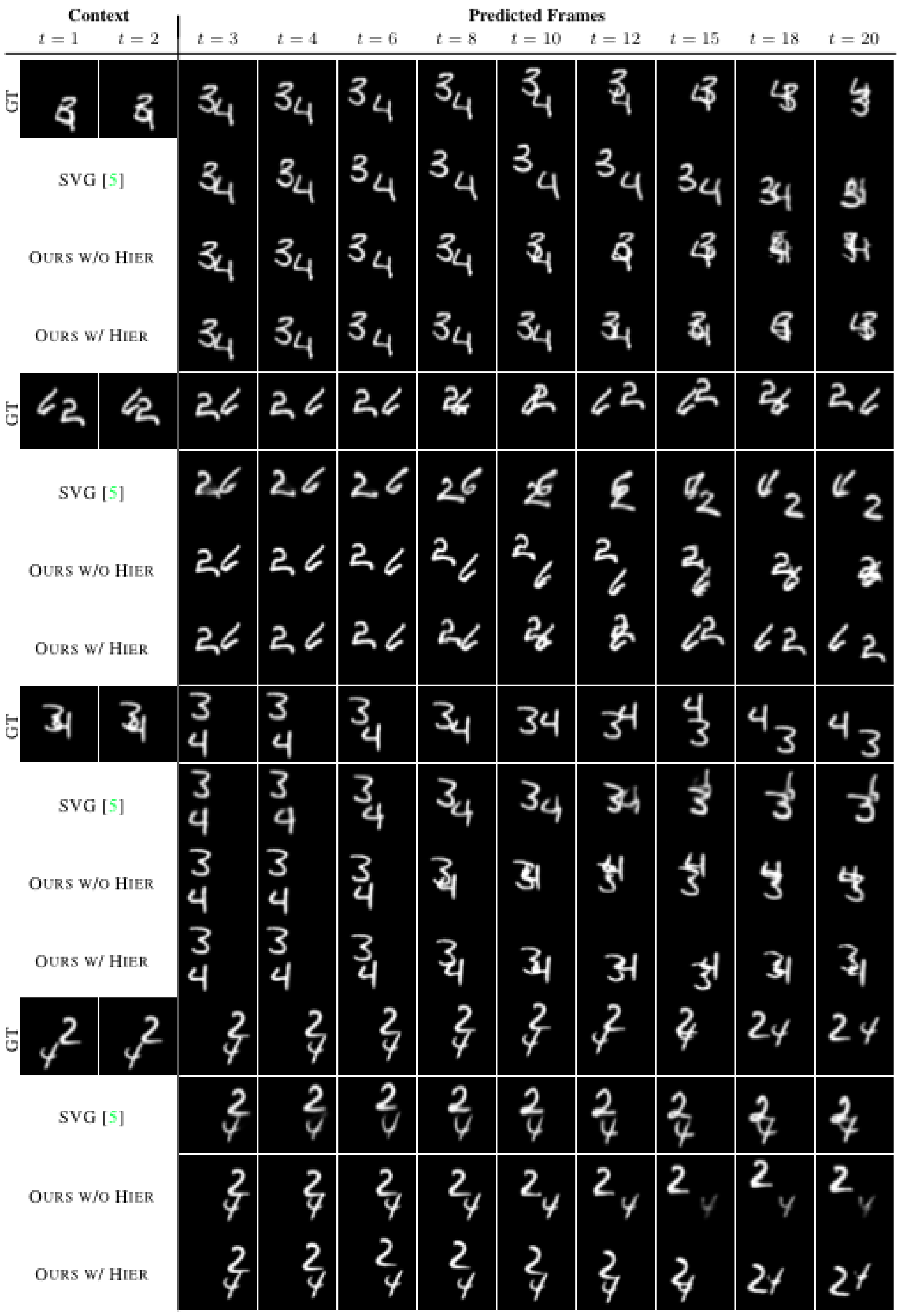


**Figure 4.13 – Additional samples on the Stochastic Moving MNIST dataset (1)**

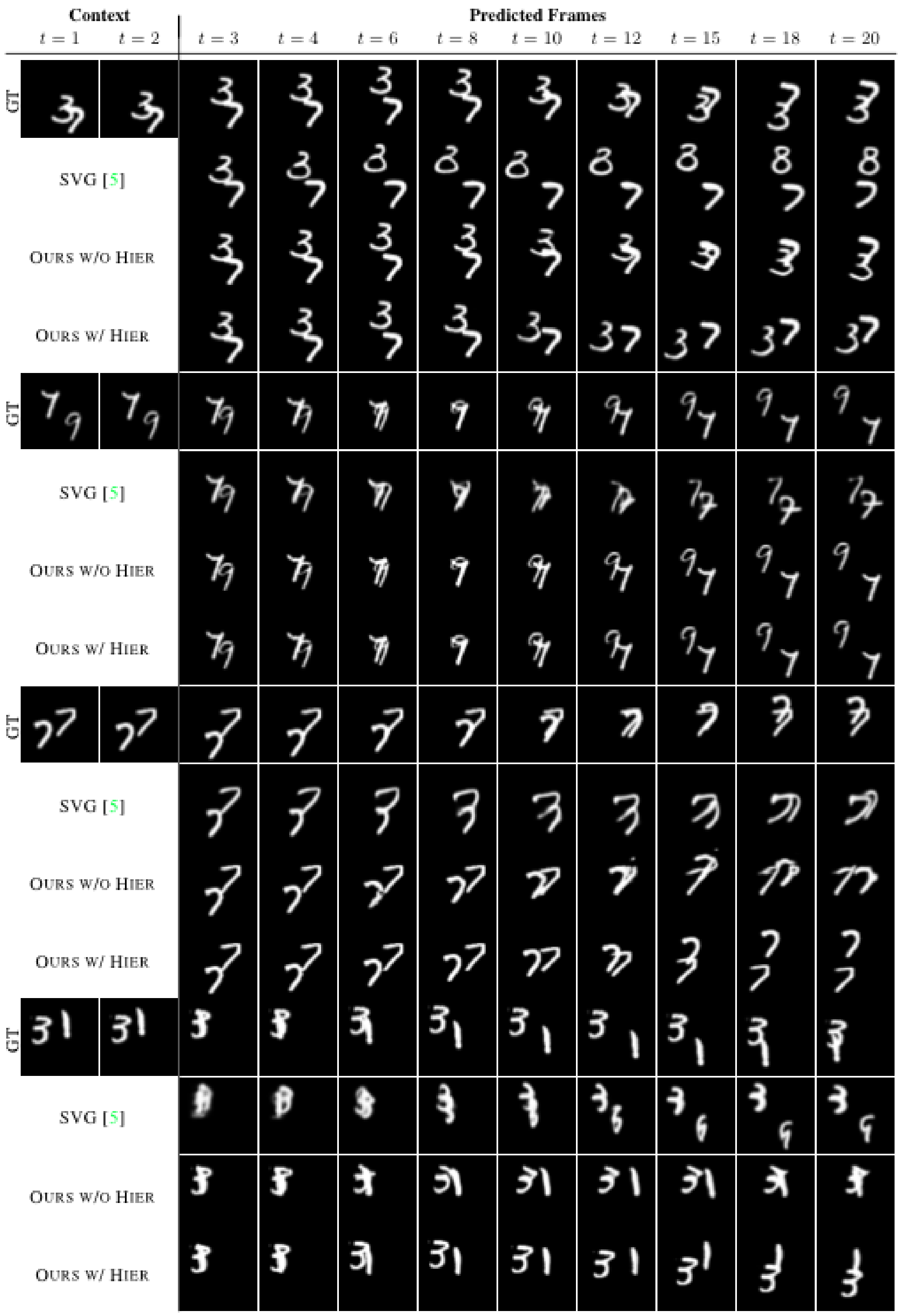


**Figure 4.14 – Additional samples on the Stochastic Moving MNIST dataset (2)**

# Chapter 5

# Prologue to the Second Article

## 5.1. Article Details

**Cascaded Video Generation for Videos In-the-Wild.** Lluis Castrejon, Nicolas Ballas and Aaron Courville, In *Proceedings of the 26th International Conference in Pattern Recognition (ICPR 2022).*

**Authors contributions.** I developed and implemented the method presented in this article, ran most of the experiment and co-wrote the article. Nicolas Ballas helped with benchmarks and experiments, provided support with the implementation and co-wrote the article. Aaron Courville supervised the project and revised the article.

## 5.2. Context

While VAEs had been proved as effective methods to capture stochasticity in video prediction, VAE generations are often blurry and lower quality than other generative methods. This was a well-known phenomenon in the image domain. Following the success of BigGAN (Brock et al., 2018) for images, DVD-GAN (Clark et al., 2019) successfully adapted ideas from the image GAN community to the field of video generation. However, DVD-GAN required a large amount of computational resources and had very long training times, which difficulted its adoption..

## 5.3. Contributions

To reduce the computational cost of GANs for video, we design a cascaded approach to video generation in which a low resolution video is first generated and then refined by one or more upscaling stages. Each stage in our model is trained independently and deals with a lower dimensional view of a video, thus reducing the overall computational requirements and training time. Furthermore, we use the better scaling capabilities of our model to generate long high-dimensional videos beyond what was possible with previous GAN approaches.

## 5.4. Recent Developments

While our article proposed more scalable GANs for video, they still remain expensive and difficult to train. The most notable development in GANs for video has been TriVD-GAN (Luc et al., 2020), which builds upon DVD-GAN and obtains good results in video prediction, but still requires a large amount of computational resources.

# Chapter 6

# Cascaded Video Generation for Videos In-the-Wild

**Abstract.** Videos can be created by first outlining a global view of the scene and then adding local details. Inspired by this idea we propose a cascaded model for video generation which follows a coarse to fine approach. First our model generates a low resolution video, establishing the global scene structure, which is then refined by subsequent cascade levels operating at larger resolutions. We train each cascade level sequentially on partial views of the videos, which reduces the computational complexity of our model and makes it scalable to high-resolution videos with many frames. We empirically validate our approach on UCF101 and Kinetics-600, for which our model is competitive with the state-of-the-art. We further demonstrate the scaling capabilities of our model and train a three-level model on the BDD100K dataset which generates 256x256 pixels videos with 48 frames.

## 6.1. Introduction

Humans have the ability to simulate visual objects and their dynamics using their imagination. This ability is linked to the ability to perform temporal planning or counter-factual thinking. Replicating this ability in machines is a longstanding challenge that video generative models try to address. Advances in generative modeling and increased computational resources have enabled the generation of realistic high-resolution images (Brock et al., 2018) or coherent text in documents (Brown et al., 2020). Yet, video generation models have been less successful, in part due to their high memory requirements that scale with the generation resolution and length.

When creating visual data, artists often first produce a rough outline of the scene, to which then they add local details in multiple iterations (Locher, 2010). The outline ensures global scene consistency and divides the creative process into multiple tractable local steps. Inspired by this process we propose CVG, a cascaded video generation model which divides the generative process into a set of simpler problems. CVG first generates a rough video that depicts a scene at a reduced framerate and resolution. This scene outline is then progressively upscaled and temporally interpolated to obtain the desired final video by one or more upscaling levels as depicted in Figure 6.1. Every cascade level outputs a video that serves as the input to the next one, with each level specializing in a particular aspect of the generation.

Levels in our model are trained greedily, i.e. in sequence and not end-to-end. This allows to train only one level at a time and thus reduce the overall training memory requirements. We formulate each level as a adversarial game that we solve leveraging the GAN framework. Our training setup has the same global solution as an end-to-end model.

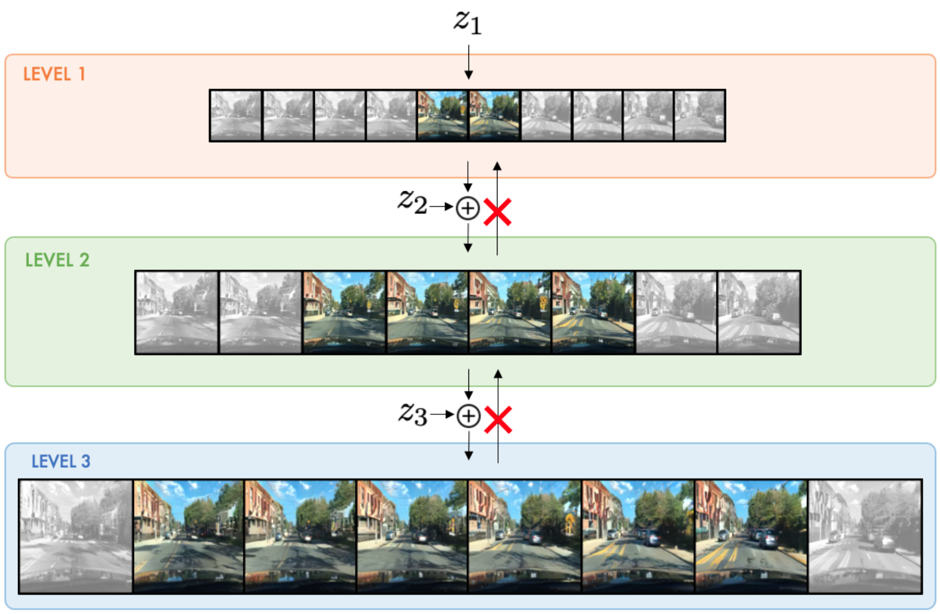


**Figure 6.1 – Cascaded Video Generation** We propose to divide the generative process into multiple simpler problems. CVG first generates a low resolution video that depicts a full scene at a reduced framerate. This scene outline is then progressively upscaled and temporally interpolated. Levels are trained sequentially and do not backprogagate gradient to previous levels. Additionally, upscaling levels can be trained on temporal crops of previous level outputs (illustrated by the non-shaded images) to reduce their computational requirements. Our model outperforms or matches the state-of-the-art in video generation and enables the generation of longer high resolution videos due to better scaling properties than previous methods.

To further reduce the computational needs of our model, upscaling cascade levels can be applied only on temporal crops from previous outputs during training. Despite this temporally-local training, upscaling levels are capable of producing videos with temporal coherence at inference time, as they upscale the output of the first level, which is temporally complete albeit at a reduced resolution. This makes CVG more scalable than previous methods, making it capable of generating high resolution videos with a larger number of frames.

Our contributions can be summarized as follows:

— We define a cascade model for video generation which divides the generation process into multiple tractable steps.

— We empirically validate our approach on UCF101, Kinetics-600 and BDD100K, large-scale datasets with complex videos in real-world scenarios. CVG matches or outperforms the state-of-art video generation models on these datasets.

— We demonstrate that our approach has better scaling properties than comparable non-cascaded approaches and train a three-level model to generate videos with 48 frames at a resolution of 256x256 pixels.

## 6.2. Cascaded Video Generation

Video scenes can be created by first outlining the global scene and then adding local details. Following this intuition we propose CVG, a cascade model in which each level only treats a lower dimensional view of the data. Video generation models struggle to scale to high frame resolutions and long temporal ranges. The goal of our method is to break down the generation process into smaller steps which require less computational resources when considered independently.

**Problem Setting** We consider a dataset of videos $(\mathbf{x}_1, ..., \mathbf{x}_n)$ where each video $\mathbf{x}_i = (\mathbf{x}_{i;0}, ..., \mathbf{x}_{i;T})$ is a sequence of $T$ frames $\mathbf{x}_{i;t} \in \mathbb{R}^{H\times W\times 3}$. Let $f_s$ denote a spatial bilinear downsampling operator and $f_t$ a temporal subsampling operator. For each video $\mathbf{x}_i$, we can obtain lower resolution views of our video by repeated application of $f_s$ and $f_t$, i.e. $\mathbf{x}_i^l = f_s(f_t(\mathbf{x}_i^{l+1}), \forall l \in [1..L]$ with $\mathbf{x}_i^L = \mathbf{x}_i$.

Each $(\mathbf{x}_i^1$, ..., $\mathbf{x}_i^L)$ comes from a joint data distribution $p_d$. The task of video generation consists in learning a generative distribution $p_g$ such that $p_g = p_d$.

**Cascaded Generative Model** We define a generative model that approximates the joint data distribution according to the following factorization:

$$p_g(\mathbf{x}^1, ..., \mathbf{x}^L) = p_{g_L}(\mathbf{x}^L|\mathbf{x}^{L-1})...p_{g_2}(\mathbf{x}^2|\mathbf{x}^1)p_{g_1}(\mathbf{x}^1). \tag{6.1}$$

Each $p_{g_i}$ defines a level in our model. This formulation allows us to decompose the generative process in to a set of smaller problems. The first level $p_{g_1}$ produces low resolution and temporally subsampled videos from a latent variable. For subsampling factors $K_T$ and $K_S$ (for time and space respectively), the initial level generates videos $\mathbf{x}_i^1 = (\mathbf{x}_{i;0}^1, \mathbf{x}_{i;K_T}^1, \mathbf{x}_{i;2K_T}^1, ..., \mathbf{x}_{i;T}^1)$, which is a sequence of $\frac{T}{K_T}$ frames $\mathbf{x}_{i;t}^1 \in \mathbb{R}^{\frac{H}{K_S} \times \frac{W}{K_S} \times 3}$. The output of the first level is spatially upscaled and temporally interpolated by one or more subsequent upscaling levels.

**Training** We train our model greedily one level at a time and in order, i.e. we train the first level to generate global but downscaled videos, and then we train upscaling stages on previous level outputs one after each other. We do not train the levels in an end-to-end fashion, which allows us to break down the computation into tractable steps by only training one level at a time. We formulate a GAN objective for each stage of our model. We consider the distribution $p_{g_1}$ in eq. 6.1 and solve a min-max game with the following value function:

$$\mathbb{E}_{\mathbf{x}^1 \sim p_d}[\log(D_1(\mathbf{x}^1))] + \mathbb{E}_{\mathbf{z}_1 \sim p_{z_1}}[\log(1 - D_1(G_1(\mathbf{z}_1)))], \tag{6.2}$$

where $G_1$ and $D_1$ are the generator/discriminator associated with the first stage and $p_{z_1}$ is a noise distribution. This is the standard GAN objective (Goodfellow et al., 2014). For upscaling levels corresponding to $p_{g_l}, l > 1$, we consider the following value function:

$$\begin{aligned}&\mathbb{E}_{\mathbf{x}^{l-1},...,\mathbf{x}^1 \sim p_d} \mathbb{E}_{\mathbf{x}^l \sim p_d(.|\mathbf{x}^{l-1},...,\mathbf{x}^1)}[\log(D_l(\mathbf{x}^l, \mathbf{x}^{l-1}))] + \\ &\mathbb{E}_{\hat{\mathbf{x}}^{l-1} \sim p_{g_{l-1}}} \mathbb{E}_{\mathbf{z}_l \sim p_{z_l}}[\log(1 - D_l(G_l(\mathbf{z}_l, \hat{\mathbf{x}}^{l-1}), \hat{\mathbf{x}}^{l-1}))],\end{aligned} \tag{6.3}$$

where $G_l$, $D_l$ are the generator and discriminator of the current level and $p_{g_{l-1}}$ is the generative distribution of the level $l-1$. The min-max game associated with this value function has a global minimum when the two joint distributions are equal, $p_d(\mathbf{x}, ..., \mathbf{x}^\mathbf{l}) = p_{g_l}(\mathbf{x}^\mathbf{l}|\mathbf{x}^{l-1})..p_{g_1}(\mathbf{x}^1)$ Dumoulin et al. (2016); Donahue et al. (2016). We also see from eq. 6.3 that the discriminator for upscaling stages operates on pairs $(\mathbf{x}^l, \mathbf{x}^{l-1})$ videos to determine whether they are real or fake. This ensures that the upscaling stages are grounded on their inputs, i.e that $\mathbf{x}^l$ "matches" its corresponding $\mathbf{x}^{l-1}$ .

**Partial View Training** Computational requirements for upscaling levels can be high when generating large outputs. As we increase the length and resolution of a generation, the need to store activation tensors during training increases the amount of GPU memory required. To further reduce the computational requirements, we propose to train the upscaling levels on only temporal crops of their inputs. This strategy reduces training costs since we upscale smaller tensors, at the expense of having less available context to interpolate frames. We define convolutional upscaling levels that learn functions that can be applied in a sliding window manner over their inputs. At inference time, we do not crop the inputs and CVG is applied to all possible input windows, thus generating full-length videos.

## 6.3. Model Parametrization

In this section we describe the parametrization of the different levels of CVG. We keep the discussion at a high level, briefly mentioning the main components of our model. Precise details on the architecture are provided in the appendix.

**First Level** The first level generator stacks units composed by a ConvGRU layer (Ballas et al., 2015), modeling temporal information, and 2D-ResNet blocks that upsample the spatial resolution. Similar to MoCoGAN (Tulyakov et al., 2018) and DVD-GAN (Clark et al., 2019), we use a dual discriminator with both a spatial discriminator that randomly samples $k$ full-resolution frames and discriminates them individually, and a temporal discriminator that processes spatially downsampled but full-length videos.

**Upsampling Levels** The upsampling levels are composed by a conditional generator and three discriminators (spatial, temporal and matching). The conditional generator produces an upscaled version $\hat{\mathbf{x}}^{\mathbf{l}}$ of a lower resolution video $\hat{\mathbf{x}}^{\mathbf{l-1}}$. To discriminate samples from real videos, upscaling stages use a spatial and temporal discriminator, as in the first level. Additionally, we introduce a matching discriminator. The goal of the matching discriminator is to ensure that the output is a valid upsampling of the input, and its necessity arises from the model formulation. Without this discriminator, the upsampling generator could learn to ignore the low resolution input video. The conditional generator is trained jointly with the spatial, temporal and matching discriminators.

**Conditional Generator** The conditional generator takes as input a lower resolution video $\hat{\mathbf{x}}^{l-1}$, a noise vector $\mathbf{z}$ and optionally a class label $y$, and generates $\hat{\mathbf{x}}^{l}$. Our conditional generator stacks units composed by one 3D-ResNet block and two 2D-ResNet blocks. Spatial upsampling is performed gradually by progressively increasing the resolution of the generator blocks. To condition the generator we add residual connections (He et al., 2016; Srivastava et al., 2015b) from the low-resolution video to the output of each generator unit. We sum nearest-neighbor interpolations of the lower resolution input to each unit output.

**Matching Discriminator** The matching discriminator uses an architecture like that of the temporal discriminator. It discriminates real or generated input-output pairs. The output is downsampled to the same size as the input, and both tensors are concatenated on the channel dimension. A precise description of all discriminator architectures can be found in the appendix.

## 6.4. Related Work

The modern video generation literature (Ranzato et al., 2014; Srivastava et al., 2015a) first started as a result of adapting techniques for language modeling to video. Since then, many papers have proposed different approaches to represent and generate videos (Luc et al., 2017, 2018; Villegas et al., 2017a,b; Xue et al., 2016), including different kinds of tasks, conditionings and models. We review the most common types of generative video models below.

Autoregressive models (Larochelle & Murray, 2011; Dinh et al., 2016; Kalchbrenner et al., 2017; Reed et al., 2017; Weissenborn et al., 2020; Yan et al., 2021) model the conditional probability of each pixel value given the previous ones. They do not use latent variables and their training can be easily parallelized. Inference in autoregressive models often requires a full forward pass for each output pixel, which does not scale well to long high resolution videos.

Normalizing flows (Rezende & Mohamed, 2015; Kingma & Dhariwal, 2018; Kumar et al., 2019) learn bijective functions that transform latent variables into data samples. Normalizing flows are able to directly maximize the data likelihood. However, they require the latent variable to have the same dimensionality as its output, which becomes an obstacle when generating videos due to their large dimensionality.

Variational AutoEncoders (VAEs) (Kingma & Welling, 2013; Rezende et al., 2014; Babaeizadeh et al., 2018) also transform latent variables into data samples. While more scalable, VAEs often produce blurry results when compared to other generative models. Models based on VRNNs (Chung et al., 2015; Denton & Fergus, 2018; Lee et al., 2018; Castrejon et al., 2019) use one latent variable per video frame and often produce better results.

Generative Adversarial Networks (GANs) are also latent variable models and optimize a min-max game between a generator G and a discriminator D trained to tell real and generated data apart (Goodfellow et al., 2014). Empirically, GANs usually produce better samples than competing approaches but might suffer from mode collapse. GAN models for video were first proposed in (Vondrick et al., 2016b,a; Mathieu et al., 2015). In recent work, SAVP (Lee et al., 2018) proposed to use the VAE-GAN (Larsen et al., 2015) framework for video. TGANv2 (Saito & Saito, 2018) improves upon TGAN (Saito et al., 2017) and proposes a video GAN trained on data windows, similar to our approach. However, unlike TGANv2, our model is composed of multiple stages which are not trained jointly. MoCoGAN (Tulyakov et al., 2018) first introduced a dual discriminator architecture for video, with DVD-GAN (Clark et al., 2019) scaling up this approach to high resolution videos in the wild. DVD-GAN outperforms MoCoGAN and TGANv2, and is arguably the current state-of-the-art in adversarial video generation.

Our model is also related to work that proposes hierarchical or progressive training approaches for generative models (Karras et al., 2017; Denton et al., 2015; Xiong et al., 2018; Zhao et al., 2020). Our model is different in that our stages are trained greedily in separate steps without backpropagation from one stage to the other, which reduces its computational requirements.

## 6.5. Experiments

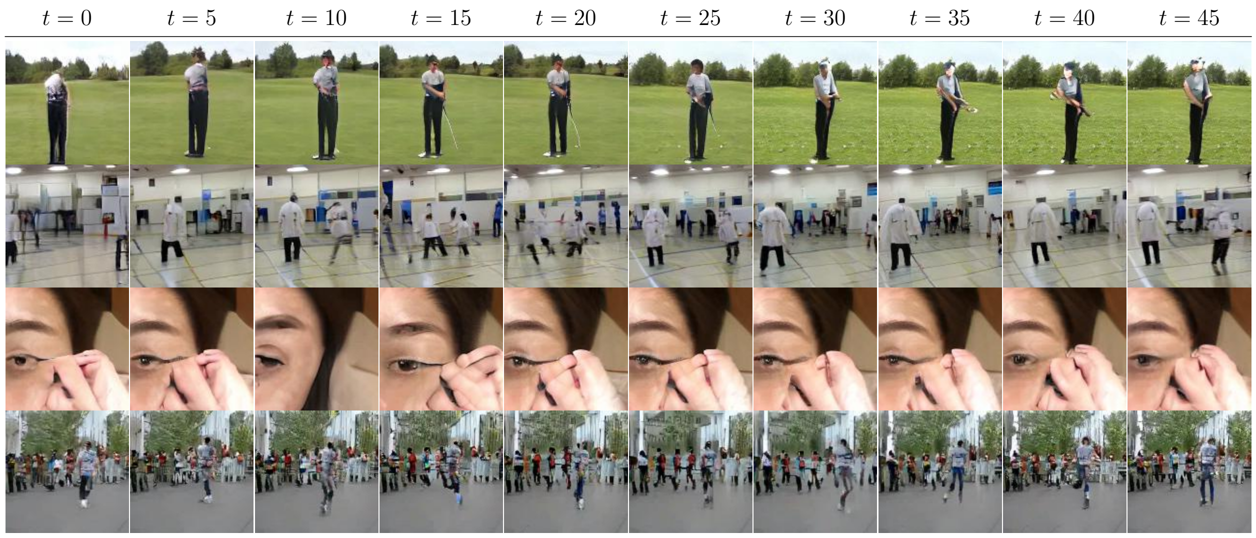


**Figure 6.2 – Randomly selected CVG 48/128x128 frame samples for Kinetics-600:** These samples were generated by unrolling CVG 12/128x128 to generate 48 frame sequences, 4 times its training horizon. Each row shows frames from the same sample at different timesteps. The generations are temporally consistent and the frame quality does not degrade over time.

In this section we empirically validate our proposed approach. First, we show that our approach outperforms or matches the state-of-the-art on Kinetics-600 and UCF101. Then, we analyze the scaling properties of our model in Section 6.5.4. Finally, we ablate the main components of our model in Section 6.5.5.

**Table 6.1 – Results on Kinetics-600 128x128** We compare our two-level CVG against the reported metrics for DVD-GAN (Clark et al., 2019). Our model is trained on 12-frame windows and matches the performance of the 12-frame DVD-GAN model. Furthermore, the same CVG model is able to generate 48 frames when applied convolutionally over a full-length first level output. In that setup our model also matches the quality of a 48-frame DVD-GAN model, but has significantly lower computational requirements.

| | | Evaluated on 12 frames | | | Evaluated on 48 frames | | |
|---|---|---|---|---|---|---|---|
| Model | Trained on | IS (↑) | FID (↓) | FVD (↓) | IS (↑) | FID (↓) | FVD (↓) |
| DVD-GAN | 12/128x128 | 77.45 | **1.16** | - | N/A | N/A | N/A |
| DVD-GAN | 48/128x128 | N/A | N/A | N/A | **81.41** | 28.44 | - |
| 2-Level CVG | 12/128x128 | **104.00** | 2.09 | 591.90 | 77.36 | **14.00** | 517.21 |

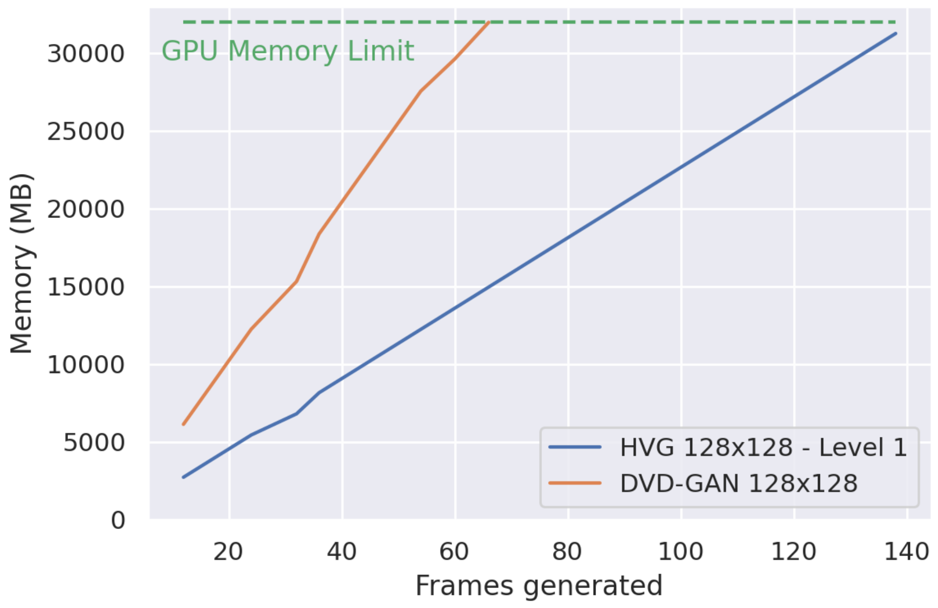


**Figure 6.3 – Scaling computational costs** We report the required GPU memory for a two-level CVG. We observe that, given the same batch size, the memory cost scales linearly with the output length. Our model scales better than a comparable non-cascaded model.

## 6.5.1. Experimental Setting

**Datasets.** We consider the Kinetics-600 (Kay et al., 2017; Carreira et al., 2018) and the UCF101 (Soomro et al., 2012) datasets for class conditional video generation. Kinetics-600 is a large scale dataset of Youtube videos depicting 600 action classes. The videos are captured in the wild and exhibit lots of variability. The amount of videos available from Kinetics-600 is constantly changing as videos become unavailable from the platform. We use a version of the dataset collected on June 2018 with around 350K videos. UCF101 contains approximately 13K videos with around 27 hours of video from 101 human action categories. Its videos

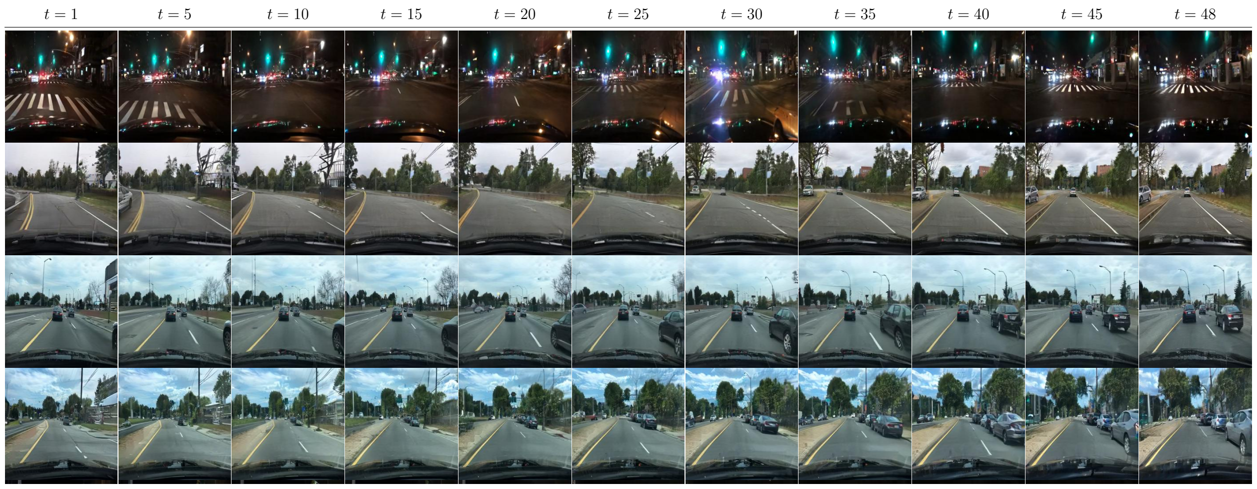


**Figure 6.4 – Random 48/256x256 BDD100K samples:** We show samples from our three-stage BDD100K model. Each row shows a different sample over time. Despite the two stages of local upsampling, the frame quality does not degrade noticeably through time.

have camera motion and cluttered backgrounds, and it is a common benchmark in the video generation community.

Additionally, we use the BDD100K dataset (Yu et al., 2018) for unconditional video generation. BDD100K contains 100k videos showing more than 1000 hours of driving under different conditions. We use the training set split of 70K videos.

For the rest of the section we denote video dimensions by their output resolution DxD and number of frames F as F/DxD.

**Evaluation metrics.** Defining evaluation metrics for video generation is an open research area. We use metrics from the image generation literature adapted to video. On Kinetics, we report three metrics: i) Inception Score (IS) given by an I3D model (Carreira & Zisserman, 2017) trained on Kinetics-400, ii) Frechet Inception Distance on logits from the same I3D network, also known as Frechet Video Distance (FVD) (Unterthiner et al., 2018), and iii) Frechet Inception Distance on the last layer activations of an I3D network trained on Kinetics-600 (FID). On BDD100K we report FVD and FID as described before, but we omit IS scores as they are not applicable since there are no classes. On UCF101 we report IS scores following the standard setup in the literature.

**Implementation details.** All CVG models are trained with a batch size of 512 and using up to 4 nVidia DGX-1. Levels for Kinetics-600 are trained for 300k iterations, while levels for BDD100K and UCF101 are trained for 100k iterations, all with early stopping when evaluation metrics stop improving. We use PyTorch and distribute training across multiple machines using data parallelism. We synchronize the batch norm statistics across workers.

We employ the Adam (Kingma & Ba, 2014) optimizer to train all levels with a fixed learning rate of $1 \times 10^{-4}$ for G and $5 \times 10^{-4}$ for D. We use orthogonal initializations for all the weights in our model as well as spectral norm in the generator and the discriminator. More details can be found in the appendix.

**Baselines.** As baselines we consider DVD-GAN (Clark et al., 2019), TGANv2 (Saito et al., 2017; Saito & Saito, 2018), VideoGPT (Yan et al., 2021) and MoCoGAN Tulyakov et al. (2018). Comparisons are mostly against DVD-GAN, as the current state-of-the-art model for class-conditional video generation and the only approach that can generate realistic samples on Kinetics-600.

### 6.5.2. Kinetics-600

We first evaluate the performance of CVG on the Kinetics-600 dataset with complex natural videos.

We train a two-level CVG on Kinetics-600 that generates either 12/128x128 or 48/128x128 videos, to compare to the previous state-of-the-art. The first level of CVG generates 24/32x32 videos with a temporal subsampling of 4 frames. The second level upsamples the first level output using a factor of 2 for the temporal resolution and a factor 4 for the spatial resolution, producing 48/128x128 videos with a temporal subsampling of 2 frames. Since these generations are large, we employ partial views on first level outputs to train the second level, which takes as input windows of 6/32x32 frames and is trained to generate 12/128x128 video snippets (4x lower dimensional than the final output). As a result, this level has approximately the same training cost than a model generating videos of size 12/128x128. At inference time we run the second level convolutionally over all the 24 first level frames to generate 128x128 videos with 48 frames. We also use the same model to generate 12/128x128 videos by using random 6 frame windows from the first level output (i.e. we use the same training and inference setup). We compare to DVD-GAN for both 12/128x128 and 48/128x128 videos with temporal subsampling 2, as the current state-of-the-art in this dataset. Note that DVD-GAN outperforms by a large margin other approaches on Kinetics-600 such as (Weissenborn et al., 2020).

We report the scores obtained by our model in Table 6.1. For 12/128x128 videos, our model achieves higher IS and comparable FID to DVD-GAN, validating that both models perform comparably when using a similar amount of computational resources. Additionally, CVG outperforms a 48/128x128 DVD-GAN model in FID score and reaches a similar IS score, despite only being trained on reduced views of the data. Qualitatively, the generations of both

**Table 6.2 – UCF101 16/128x128 comparison:** We compare CVG to previous video generation approaches on UCF101. Our method obtains significantly higher Inception Score.

| **Model** | **IS(↑)** |
|---|---|
| MoCoGAN (Tulyakov et al., 2018) | 12.42 |
| VideoGPT (Yan et al., 2021) | 24.69 |
| TGANv2 (Saito & Saito, 2018) | 28.87 |
| DVD-GAN (Clark et al., 2019) | 32.97 |
| CVG (ours) | **53.72** |

models are similar - they do not degrade noticeable in appearance through time although both have some temporal inconsistencies. For CVG temporal inconsistencies are often due to a poor first level generation. For DVD-GAN we hypothesize that inconsistencies are due to the reduced temporal field of view of its discriminator, which is of less than 10 frames while the model has to generate 48 frames.

### 6.5.3. UCF101

We further evaluate our approach on the smaller UCF101 dataset. We train a two-level CVG to generate 16/128x128 videos, as commonly done in the literature. The first level generates 8/64x64 videos with a temporal subsampling of 2 frames. The second level upscales the output of the first level to 16/128x128 videos, without temporal subsampling to match the literature. Since these are small videos, we do not use temporal windows to train the second upscaling level.

We report the scores obtained by our model in Table 6.2. Our model obtains state-of-the-art results, outperforming previous approaches by a large margin. Qualitatively, CVG generates coherent samples with high fidelity details, with small temporal inconsistencies. Samples for the UCF101 dataset can be found in the appendix.

### 6.5.4. Scaling up CVG on BDD100K

To show that our model scales well with the video dimensionality, we train a three-stage model on the BDD dataset to generate 48/256x256 videos. A training iteration with a single 48/256x256 video for a similarly sized non-cascaded model requires more than 32GB of GPU memory. Such model is therefore not trainable on most current GPUs without techniques like gradient checkpointing which add a significant overhead to the training time. Instead, with our cascaded model we can fit 4 examples per GPU without any engineering tricks.

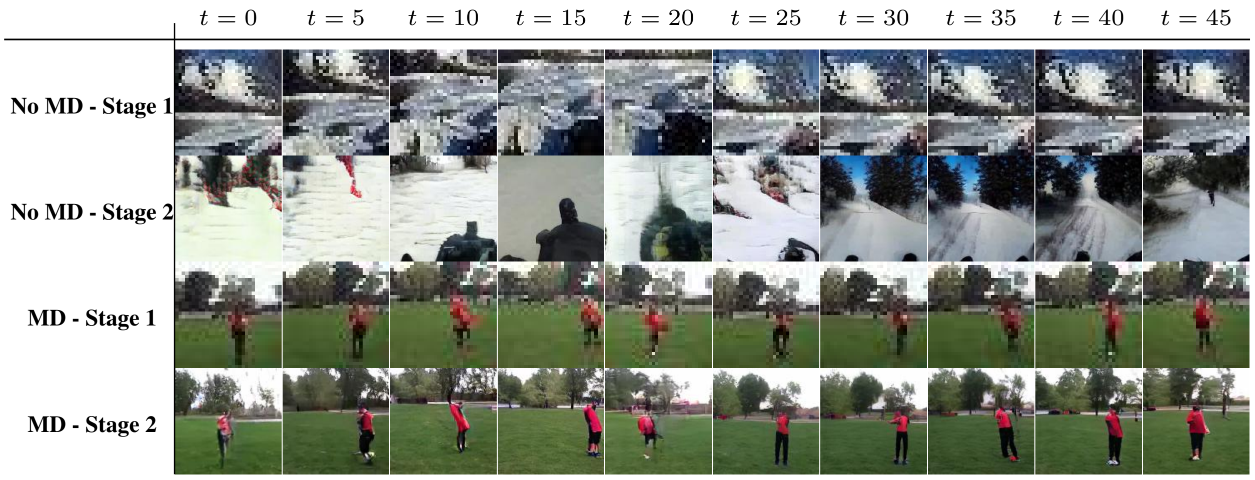


**Figure 6.5 – Matching discriminator samples** We show a random sample from our two-level model on Kinetics-600 with the matching discriminator and without the matching discriminator (No MD). For each sample we show the output of the first level and the corresponding second level output. While the No MD model generates plausible local snippets at level 2, it does not remain temporally coherent. Our model with the matching discriminator is temporally consistent because it is grounded in the low resolution input.

We train the first level to output 12 frames at 64x64 resolution with a temporal subsampling of 8 frames. The second level upsamples 12 frame windows at 128x128 resolution with temporal subsampling of 4 frames (since we are doubling the framerate of the first level). The third level is trained to upscale 12 frame windows at 256x256 resolution for a final temporal subsampling of 2 frames. Figure 6.4 shows samples from this model. The videos appear crisp, show multiple settings and do not degrade through time.

To further illustrate the scaling capabilities of our model, we report the memory requirements for a two-level 128x128 CVG as a function of the number of output frames in Figure 6.3. The first level generates half the output frames at 64x64, while the second level is trained to upscale windows of 6 frames into 12 128x128 frames, regardless of the first level output length. Compared to a non-cascaded 128x128 model, our first level scales better due to the lower resolution and reduced number of frames, and the second level has a fixed memory cost of 10290MB since it is trained on 6 frames windows. Given the same GPU memory budget, our model can generate sequences of up to 140 frames, more than double the frames of a non-cascaded model.

## 6.5.5. Model Ablations

**Matching Discriminator Ablation.** To assess the importance of the matching discriminator, we compare two-level CVG models with and without the matching discriminator (we

**Table 6.3 – Matching discriminator comparison** We report metrics on Kinetics-600 and BDD100K for our model with and without the matching discriminator. Both models perform similarly for 6 frames, corresponding to the training video length. However, the model without the matching discriminator produces incoherent generations when applied over the full first level input because it can ignore it.

| | | 6 Frames | | | 50 Frames | | |
|---|---|---|---|---|---|---|---|
| Dataset | Model | IS (↑) | FID (↓) | FVD (↓) | IS (↑) | FID (↓) | FVD (↓) |
| Kinetics-600 | CVG (No MD) | **50.31** | 1.62 | 594.99 | 37.81 | 42.29 | 1037.79 |
| | CVG | 48.44 | **1.06** | **565.95** | **49.44** | **31.87** | **790.97** |
| BDD100K | CVG (No MD) | N/A | 1.36 | 211.69 | N/A | 26.52 | 575.51 |
| | CVG | N/A | **1.07** | **144.96** | N/A | **18.73** | **326.78** |

**Table 6.4 – Temporal Window Ablation** We compare two-level models trained with different window sizes for the upscaling levels. The 6-frame model has higher computational requirements but outperforms the 3-frame model, confirming a trade-off between computational savings and final performance when selecting the temporal window size.

| | Kinetics-600 48 Frames | | |
|---|---|---|---|
| Window | IS (↑) | FID (↓) | FVD (↓) |
| 3-frame | 58.21 | 31.59 | 714.74 |
| 6-frame | **77.36** | **14.00** | **517.21** |

refer to the latter as No MD). As in Section 6.5.2, we generate 48 frames with a two-level model on Kinetics-600. We train the upscaling level on 3-frame windows of the first level to generate 6 frames. We expect the No MD model to generate inconsistent full-length videos when applied over the full first level since its outputs are not necessarily valid upscalings of the inputs. On 6 frame generations (i.e. the training setup), CVG and CVG No MD obtain similar scores as reported in Table 6.3. While the No MD model ignores its previous level inputs, it still learns to generate plausible 6 frame videos at 128x128. However, when we use the models to generate full-length 48 frame videos, CVG No MD only generates valid local snippets and is inconsistent through time. Figure 6.5 shows an example of a full length No MD generation in which this effect is observable. In contrast, our model (MD) stays grounded to the input and remains consistent through time. This is reflected in the reported metrics in Table 6.3, where the No MD model has worse scores. This ablation shows the need of a matching discriminator to ground upsampling level outputs to their inputs.

**Temporal Window Ablation.** One modelling choice in CVG is the temporal window length used in the upsampling levels. Shorter inputs provide less context to upsample frames, while longer inputs require more compute. To assess the impact of the window length, we compare two-level models trained on Kinetics-600 128x128: one trained on first level windows of 6 frames (same setup as in Section 6.5.2) and one trained on windows of only 3 frames. The 6-frame level requires approximately 2x GPU memory than the 3-frame level during training, but we expect it to perform better due to the larger context available for upscaling. We compare their performance to generate 48 frames in Table 6.4. We conclude that the window size defines a trade-off between computational resources and sample quality. We refer the reader to the appendix for an additional ablations and experiments.

## 6.6. Conclusions

We propose CVG, a cascaded video generator that divides the generative process into simpler steps. Our model is competitive with state-of-the-art approaches in terms of sample quality, while requiring significantly less computational resources due to the cascaded approach. Higher capacity models and larger outputs are key aspects in improving video generation, and CVG is a step in that direction with better scaling properties than previous approaches.

**Acknowledgments.** We acknowledge the support from the FAIR and Mila teams. L.C. was supported by an IVADO PhD fellowship.

## 6.A. Appendix - Additional Implementation Details

We use the Adam optimizer (Kingma & Ba, 2014) with learning rate $\lambda_G = 1 \times 10^{-4}$ and $\lambda_D = 5 \times 10^{-4}$ for the generator and the discriminator, respectively. The discriminator is updated twice for each generator update.

We use orthogonal initialization for all the weights in our model and use spectral normalization both in the generator and the discriminator. We only use the first singular value to normalize the weights. We do not use weight moving averages nor orthogonal penalties.

Conditional batch normalization layers use the input noise as the condition, concatenated with the class label when applicable. Features are normalized with a per-frame mean and standard deviation.

To unroll a generator beyond its training temporal horizon, we apply it convolutionally over longer input sequences. We perform 200 “dummy” forward passes to recompute the per-timestep batch normalization statistics at test time.

All convolutions in our models use 3x3 or 3x3x3 filters with padding=1 and stride=1, for 2D or 3D convolutions respectively. All models were implemented in PyTorch.

For the rest of the appendix, we use B to denote the batch size, T for the number of frames or timesteps, C for the number of channels, H for the height of the frame and W is the width of the frame.

### 6.A.1. Model Architecture - First level

**Generator.** The generator is composed by a stack of units where each unit is comprised of a ConvGRU layer and two 2D-ResNet upsampling blocks. We describe our network using a base number of channels *ch* and the channel multipliers associated with each unit. Our first level generator is formed by 4 units with channel multipliers $[8, 8, 4, 2]$. The base number of channel is 128.

The first input of this network is of size BxTx(8x*ch*)x4x4. This input is obtained by first embedding the class label onto a 128 dimensional space, then concatenating the embedding to a 128 dimensional noise vector. This concatenation is mapped to a Bx(8x*ch*)x4x4 tensor with a linear layer and a reshape, and then the final tensor is obtained by replicating the output of the linear layer T times.

The ConvGRU layer (Ballas et al., 2015) follows the ConvGRU implementation of (Clark et al., 2019) and uses a ReLU non-linearity to compute the ConvGRU update.

The 2D ResNet blocks are of the norm-act-conv-norm-act-conv style. We use conditional batch normalization layers, ReLU activations and standard 2D convolutions. Before the first convolution operation and after the first normalization and activation, there is an optional upsampling operation when increasing the resolution of the tensor. We use standard nearest neighbor upsampling. Except for the last unit, all units perform this upsampling operation. The conditional batch normalization layers receive the embedded class label (if applicable) and the input noise as a condition and map it to the corresponding gain and bias term of the normalization layer using a learned linear transformation. The 2D ResNet blocks process all frames independently by reshaping their input to be (B*T)xCxHxW.

The output of the last stack goes through a final norm-relu-conv-tanh block that maps the output tensor to RGB space with values in the [-1, 1] range.

**Discriminator.** There are two discriminators, a 2D spatial discriminator and a 3D temporal discriminator. The 2D discriminator is composed of 2D ResNet blocks. Each ResNet block is formed by a sequence of relu-conv-relu-conv layers. There are no normalization layers in the discriminator. After the last conv in each block there is an optional downsampling operation, which is implemented with average pooling layers. The 2D discriminator receives as input 8 randomly sampled frames from real or generated samples.

The 3D discriminator is equal to the 2D discriminator except that its first two layers are 3D ResNet blocks, implemented by replacing 2D convolutions with regular 3D convolutions. The 3D discriminator receives as input a spatially downsampled (by a factor of two) real or generated sample. The 2D blocks process different timesteps independently.

We concatenate the output of both discriminators and use a geometric hinge loss. The loss is averaged over samples and outputs.

We use 128 as base number of channel for both discriminators, with the following channel multipliers for each ResNet block: $[16, 16, 8, 4, 2]$

## 6.A.2. Model Architecture - Upsampling levels

The upsampling levels follow the same architecture as the first level but with the following modifications.

**Generator.** The generator units replace the ConvGRU layers with a Separable 3D convolution. We first convolve over the temporal dimension with a 1D temporal kernel of size 3 and then convolve over the spatial dimension with 2D 3x3 kernel. We empirically compare generators with ConvGRU and separable convolutions in Section 6.B, showing that the 3D convolution performs as well as the ConvGRU but it can be run in parallel.

We add residual connections at the end of each 3D and 2D ResNet block to an appropriately resized version of $\mathbf{x}^{l-1}$. We use nearest neighbor spatial downsampling for this operation, and we use nearest neighbor temporal interpolation to increase the number of frames of $\mathbf{x}^{l-1}$. We then map the residual to the appropriate number of channels using a linear 1x1 convolution. We do not add $\mathbf{x}^{l-1}$ residual connections to feature maps with spatial resolutions (HxW) greater than the resolution of $\mathbf{x}^{l-1}$.

**Discriminator.** We reuse the same 2D and 3D discriminators as for the first stage. Additionally, we add a matching discriminator that discriminates $(\mathbf{x}^l, \mathbf{x}^{l-1})$ pairs. The matching discriminator utilizes the same architecture as the 3D discriminator. It receives as input a concatenation of $\mathbf{x}^{l-1}$ and a downsampled version of $\mathbf{x}^l$ to match the resolution of $\mathbf{x}^{l-1}$. We concatenate the outputs of all three networks and use a geometric hinge loss, as done for the first level discriminator. The overall loss is averaged over samples and output locations.

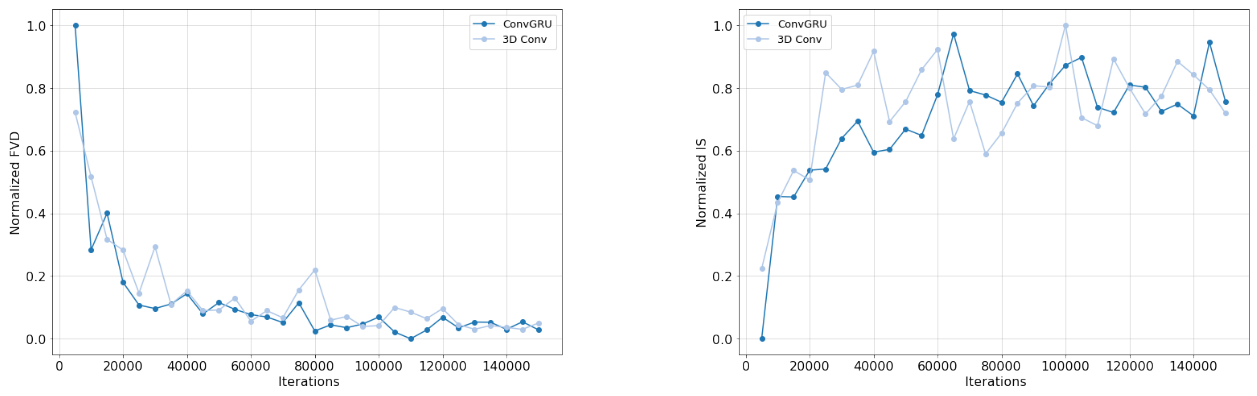


**Figure 6.6 – Comparison of recurrent layers** We compare two variants of the same generator, one with a single ConvGRU layer per generator block and one with a separable 3D convolution per generator block. On the left we show the evolution of the FVD score during training, and on the right we show the Inception Score. Both scores are normalized to the [0, 1] range where 1 is the highest score obtained by these models and 0 the lowest. Both models have similar behaviour and computational costs, but the 3D convolution processes inputs in parallel.

For 128x128 generations on Kinetics, the generator uses 128 as base number of filters with the following channel multipliers $[8, 8, 4, 2, 1]$. All discriminators have 96 base channels and the following channel multipliers $[1, 2, 4, 8, 16, 16]$. All our Kinetics models at 128x128 are two-level models. We train models to upsample inputs of sizes 3/32x32 or 6/32x32 to 6/128x128 or 12/128x128, respectively. Since we train our first level for 24/32x32 outputs, our two-level models can generate 48/128x128 outputs when unrolled.

For 128x128 generations on BDD100K, the generator uses 96 as the base number of channels with channel multipliers $[8, 8, 4, 2, 1]$. All discriminators have 96 base channels and channel multipliers $[1, 2, 4, 8, 16, 16]$. Our BDD100K 128x128 models upsample 6/64x64 inputs to 12/128x128, and can generate outputs of up to 24/128x128.

For 256x256 generations on BDD100K, the generator uses 96 as the base number of channels with channel multipliers $[8, 4, 4, 4, 2, 1]$. All discriminators have 96 base channels and channel multipliers $[1, 2, 4, 8, 8, 16, 16]$. Our 256x256 model upsamples 6/128x128 inputs to 12/256x256, and can generate outputs of up to 48/256x256.

## 6.B. Comparison of Recurrent Layers

In this section we justify the change of the ConvGRU for Separable 3D convolutions in upsampling levels. In Figure 6.6 we compare the evolution of two metrics (IS and FVD) during training for two variants of the same two-stage model, one using ConvGRUs and one using separable 3D convolutions. Both models show similar behavior during training and achieve similar final metrics. However, ConvGRUs perform sequential operations over time whereas 3D convolutions can be parallelized.

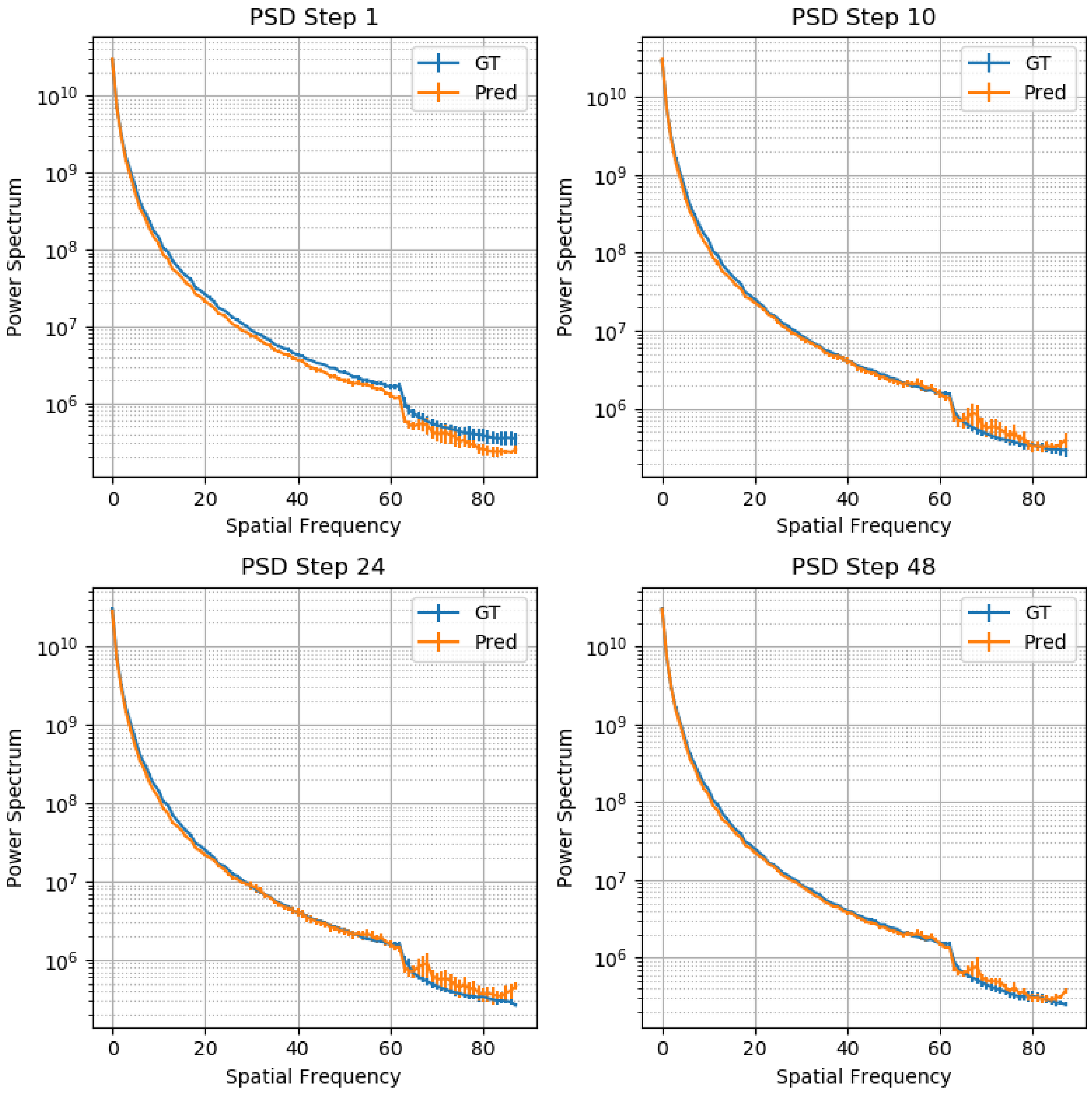


**Figure 6.7 – Power Spectrum Density (PSD) plots for different time steps** We show a comparison of the PSD between the original data and our generations at different steps in the predictions. We observe that our generations have a similar PSD to that of the original data, even at the end of the generation, indicating that the generations do not blur over time significantly.

## 6.C. Power Spectrum Density

Some video generation models produce blurry results over time. As an additional evaluation, we generate Power Spectrum Density (PSD) plots to assess whether our generations become blurrier over time, following (Ayzel et al., 2020).

We conduct this experiment on Kinetics for videos of 48 frames at 128x128 resolution. We use our model with the first level trained on 24/32x32 sequences and the second level trained to generate 12/128x128 video snippets from 6/32x32 windows, and unrolled after training to produce 48/128x128 videos. We took 1800 random videos from the ground truth data (GT) and 1800 generations from our model. We compute the PSD at frames 1, 10, 24, and 48 of each video. For each set of 600 videos, we compute the average PSD across videos, on a per frame basis. Finally, we use the three sets of 600 videos to compute the standard deviation and mean for the average PSD of the original data and our generations. Figure 6.7 shows the plots for different frame indices. Our generations have a very similar PSD to that of GT in all video frames. This indicates that our generations, while they might not be accurate, have very similar frequency statistics as the ground-truth data. We do not observe any significant blurring over time, which is confirmed by the plots - they show that even for frame 48 high frequencies are very similar between the original data and our generations.

## 6.D. Influence of Motion on the Results

In this section we analyze whether our model has different performance for categories with different motion characteristics as an additional analysis.

We conduct this experiment for CVG trained on Kinetics-600 to generate 24/32x32 videos in the first level and then to upscale 6/32x326 videos to 12/128x128 videos for the second level. The second level is unrolled over the full first level generation to obtain 48/128x128 videos.

We randomly select five categories of videos with high motion content (bungee jumping, capoeira, cheerleading, kitesurfing and skydiving) and five categories with less dynamic videos (doing nails, cooking egg, crying, reading book and yawning). We generate 1000 samples from CVG for each category, and use all available samples in the dataset to compute IS and FID scores per category.

Table 6.5 shows the IS and FID scores of each class, while Figure 6.8 and Figure 6.9 show samples from high motion and low motion categories, respectively. We do not observe a trend that indicates that CVG produces worse generations for high motion classes. Some

| | | 48 frames | | |
|---|---|---|---|---|
| | Class | IS (↑) | FID (↓) | # Videos |
| **High Motion** | Bungee Jumping | 11.66 | 82.21 | 799 |
| | Capoeira | 9.48 | 84.57 | 816 |
| | Cheerleading | 12.10 | 116.84 | 982 |
| | Kitesurfing | 11.36 | 108.81 | 648 |
| | Skydiving | 5.99 | 90.25 | 983 |
| **Low Motion** | Doing Nails | 13.32 | 91.67 | 537 |
| | Cooking Egg | 7.60 | 111.63 | 441 |
| | Crying | 8.92 | 70.74 | 627 |
| | Reading Book | 11.13 | 64.97 | 793 |
| | Yawning | 9.71. | 79.08 | 530 |

**Table 6.5 – Per category scores for classes with different amounts of motion (Kinetics-600)** We report per-class IS and FID scores for 5 randomly selected categories with high motion and 5 categories with low motion. We observe that there is a high variability in FID scores, with some classes with low motion having high scores as well as some high motion classes. In IS scores there are few differences between the two groups, with the high motion group having a slightly higher mean score.

low motion categories have high FID scores similar to the highest scores for the high motion categories, while on average the IS scores for the high motion categories are slightly better. We do not notice a qualitative difference. Instead, we believe there might be other factors - amount of structure present in a scene for example - that have greater impact on the output quality.

## 6.E. Cascaded Training Objective

In this section we describe our training objective more formally. For a CVG model with $L$ levels, our goal is to model the joint probability distribution $p_d(\mathbf{x}^1, ..., \mathbf{x}^L) = p_g(\mathbf{x}^1, ..., \mathbf{x}^L) = p_{g_L}(\mathbf{x}^L|\mathbf{x}^{L-1})..p_{g_1}(x^1)$, where each $p_{g_l}$ is defined by a level $l$ in our model.

**Training Level 1.** We consider the distribution $p_{g_1}$ and solve a min-max game with the following value function:

$$V_1(G_1, D_1) =$$
$$\mathbb{E}_{\mathbf{x}^1 \sim p_d}[\log(D_1(\mathbf{x}^1))] + \mathbb{E}_{\mathbf{z}_1 \sim p_{z_1}}[\log(1 - D_1(G_1(\mathbf{z}_1)))],$$

where $G_1$ and $D_1$ are the generator/discriminator associated with the first stage and $p_{z_1}$ is a noise distribution.

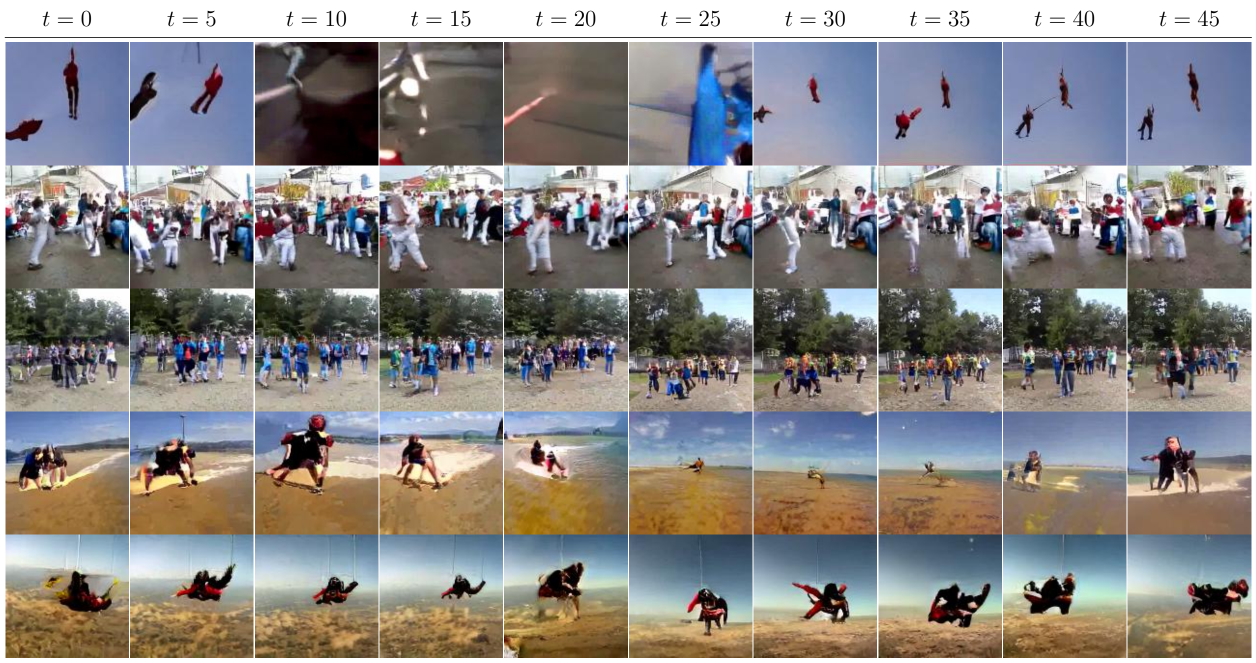


**Figure 6.8 – Samples from Kinetics-600 classes with high motion content**

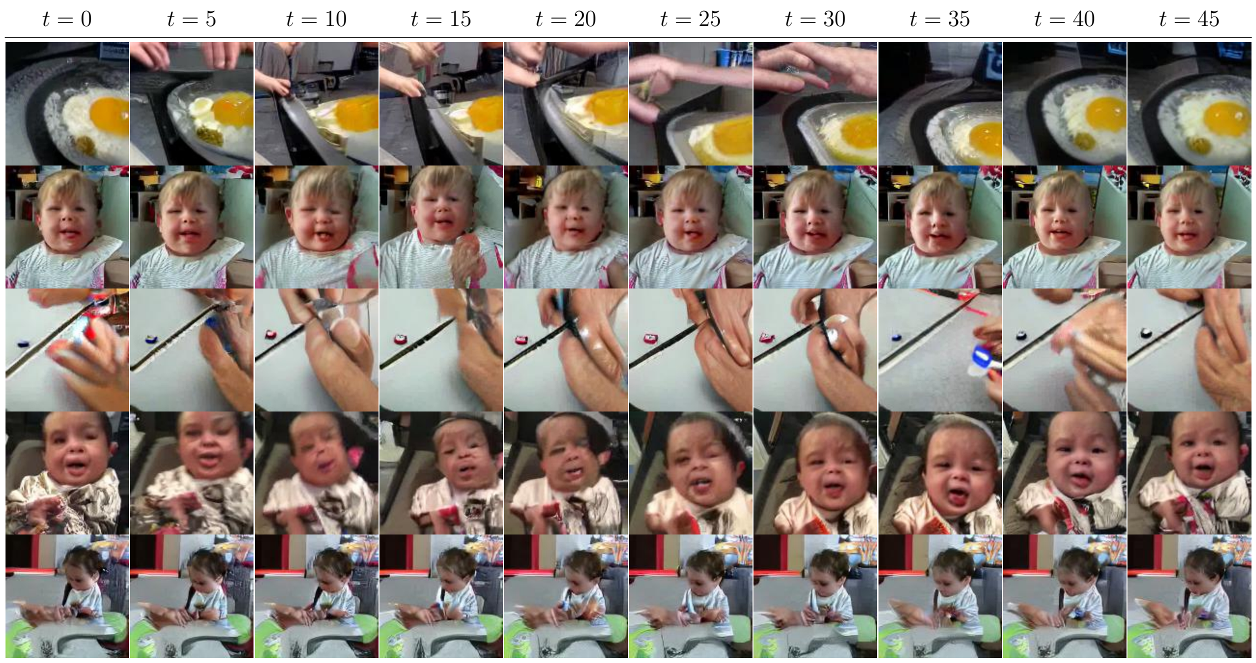


**Figure 6.9 – Samples from Kinetics-600 classes with low motion content**

This is the standard GAN objective. As shown in (Goodfellow et al., 2014), the min-max game $\min_{G_1} \max_{D_1} V_1(G_1, D_1)$ has a global minimum when $p_{g_1}(\mathbf{x}^1) = p_d(\mathbf{x}^1)$.

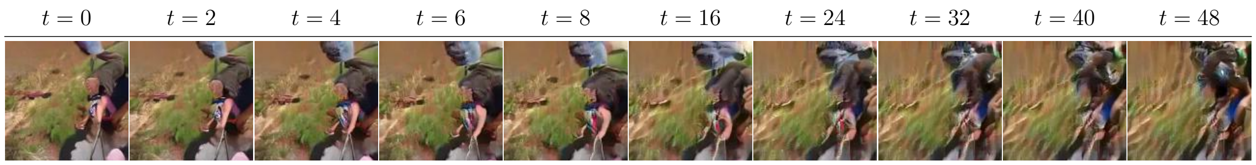


**Figure 6.10 – DVD-GAN fails to generate samples beyond its training horizon** These samples were obtained by changing the spatial dimensions of the latent in a 6/128x128 DVD-GAN model to produce 48/128x128 videos. The samples quickly degrade after the first few frames and become motionless.

**Training upsampling levels.** For each upscaling level $l > 1$ we formulate a min-max game with the following value function:

$$V_l(G_l, D_l) =$$
$$\mathbb{E}_{\mathbf{x}^{l-1},...,\mathbf{x}^1 \sim p_d} \mathbb{E}_{\mathbf{x}^l \sim p_d(.|\mathbf{x}^{l-1},...,\mathbf{x}^1)}[\log(D_l(\mathbf{x}^l, \mathbf{x}^{l-1}))] +$$
$$\mathbb{E}_{\hat{\mathbf{x}}^{l-1} \sim p_{g_{l-1}}} \mathbb{E}_{\mathbf{z}_l \sim p_{z_l}}[\log(1 - D_l(G_l(\mathbf{z}_l, \hat{\mathbf{x}}^{l-1}), \hat{\mathbf{x}}^{l-1}))],$$

where $G_l$, $D_l$ are the generator and discriminator of the current level and $p_{g_{l-1}}$ is the generative distribution of the level $l - 1$.

The min-max game $\min_{G_l} \max_{D_l} V_l(G_l, D_l)$ has a global minimum when the two joint distributions are equal, $p_d(\mathbf{x}, ..., \mathbf{x}^l) = p_{g_l}(\mathbf{x}^l|\mathbf{x}^{l-1})..p_{g_1}(\mathbf{x}^1)$ Dumoulin et al. (2016); Donahue et al. (2016). It follows that $p_d(\mathbf{x}^l|\mathbf{x}^{l-1}) = p_{g_l}(\mathbf{x}^l|\mathbf{x}^{l-1})$ when $p_{g_{l-1}}(\mathbf{x}^{l-1}|\mathbf{x}^{l-2})..p_{g_1}(\mathbf{x}^1) = p_d(\mathbf{x}^{l-1}, ..., \mathbf{x}^1)$. Level $l$ only learns the parameters associated with the distribution $p_{g_l}$, as all $p_{g_i}, 1 \leq i \leq l - 1$ levels are trained previously and their training objectives admit a global minimum when they match the data distribution. However, even if the distribution $p_{g_{l-1}}$ does not match exactly the marginal data distribution, our model still aims at learning a distribution $p_{g_l}$ such that $p_{g_l}(\mathbf{x}^l|\mathbf{x}^{l-1})..p_{g_1}(\mathbf{x}^1)$ approximates the joint data distribution.

## 6.F. DVD-GAN unrolling

One of the main characteristics of CVG is the ability to change the training and inference setup for upscaling levels. Since DVD-GAN is mostly a convolutional model, we investigate whether it can generate videos of longer duration than those it is trained on. The number of frames in that model is controlled by the RNN that receives as input the latent variable sample and outputs as many tensors as frames to be generated. We adjust the number of steps for this RNN and recompute batch normalization statistics to account for the extended amount of frames. A prototypical example from an extended DVD-GAN model generation can be found in Figure 6.10. We observe that for this model videos are coherent up until

|  | Iteration time | Total time |
|---|---|---|
| CVG Level 1 | 3.45s | 4 days |
| CVG Level 2 | 3.20s | 3.7 days |
| DVD-GAN | 11.90s | 13.8 days |

**Table 6.6 – Training time comparison**

the training length, but then they visibly degrade and quickly become motionless, producing implausible generations.

## 6.G. Time Complexity

In Table 6.6 we compare a two-level CVG and a DVD-GAN model with similar capacity trained on 48/128x128 Kinetics-600 videos with a batch size of 512 and 64 GPUs. In addition to the memory savings, the total training time of CVG (7.7 days) is significantly shorter. Our approach uses smaller networks at each level and produces smaller outputs. Therefore, each level can have a reduced training iteration time. Using an additional third level to generate larger outputs increases the CVG training time. However, we cannot train the equivalent DVD-GAN baseline for the three-level model due to memory constraints, as it does not fit in GPU memory even with a batch size of 1 example.

## 6.H. BDD100K metrics

|  |  | Evaluated on 12 frames |  | Evaluated on 48 frames |  |
|---|---|---|---|---|---|
| Model | Trained on | FID (↓) | FVD (↓) | FID (↓) | FVD (↓) |
| 3-Level CVG | 12/256x256 | 3.66 | 541.37 | 21.38 | 391.69 |

**Table 6.7 – BDD100K 256x256 Metrics** We report the FID and FVD scores for our three-level CVG trained on BDD100K. The model is trained to generate 12/256x256 videos and at inference it produces 48/256x256 videos.

We report the metrics for our three-level BDD100K model for help future comparison to CVG in Table 6.7.

**Qualitative comparison with DVD-GAN.** As a point of comparison we provide samples from the official DVD-GAN 48/128x128 trained on Kinetics-600 and released by the authors. Samples can be download at this URL: `https://drive.google.com/file/d/1P8SsWEGP6tEGPPNPH-iVycOlN6vpIgE8/view?usp=sharing`. We observe that the samples from DVD-GAN and our CVG model are of similar quality.

## 6.I. Additional Samples

### 6.I.1. Upsampling Visualizations

In this section we show some examples of level 1 generations on Kinetics-600 for a 24/32x32 model, as well as the corresponding 48/128x128 generations from level 2 trained to upscale 6/32x32 windows to 12/128x128 and unrolled over the whole first level generation. Examples are shown in Figure 6.14, in which, for each example, we show the level 1 generation on the top row and the corresponding level 2 generation in the lower row. We observe that the second level adds details and refines the low resolution generation beyond simple upsampling, but at the same time keeps the overall structure of the low resolution generation and is properly grounded.

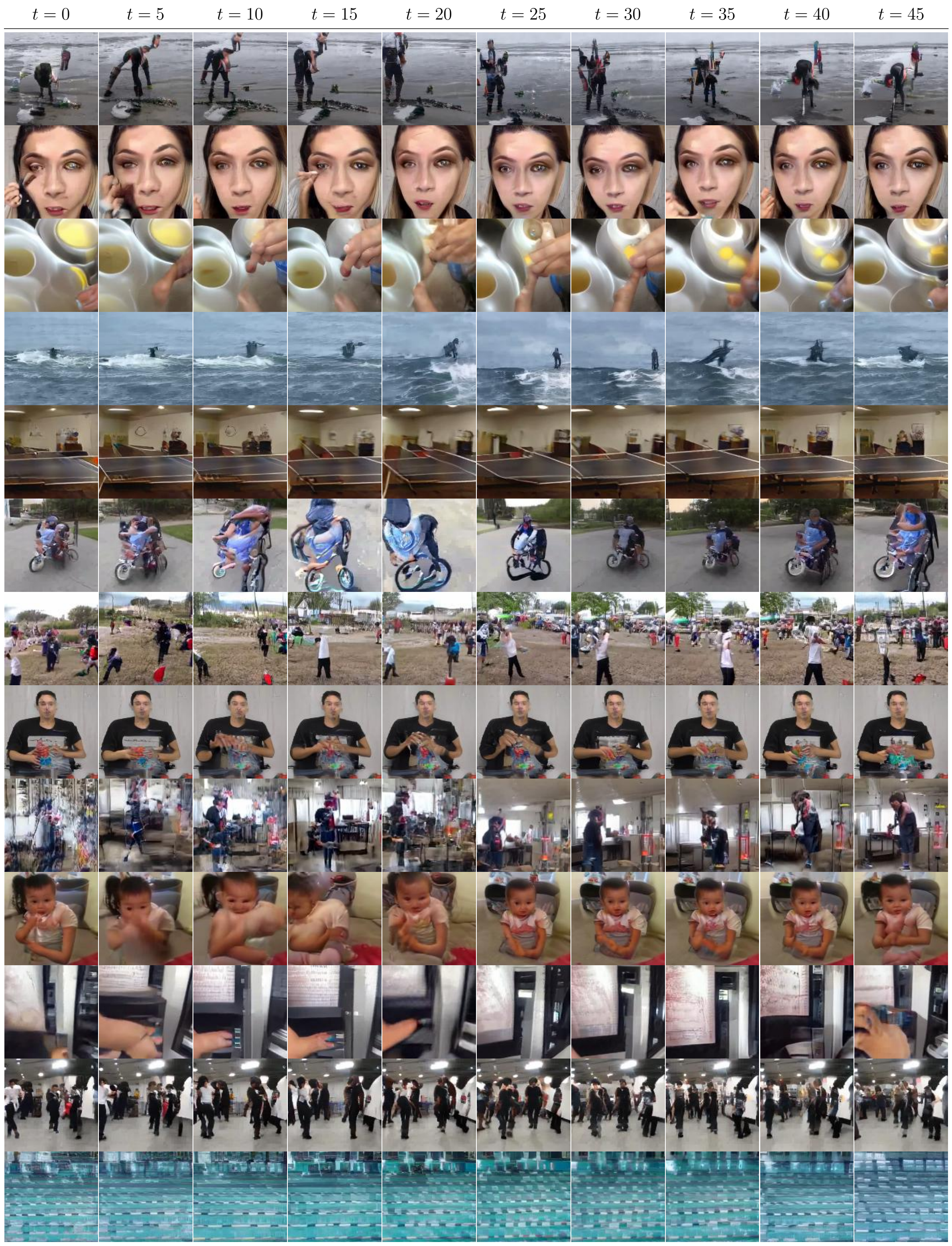


**Figure 6.11 – Additional samples for Kinetics 12/128x128** We show additional samples from our two-level Kinetics 12/128x128 model unrolled to generate 48/128x128 videos.

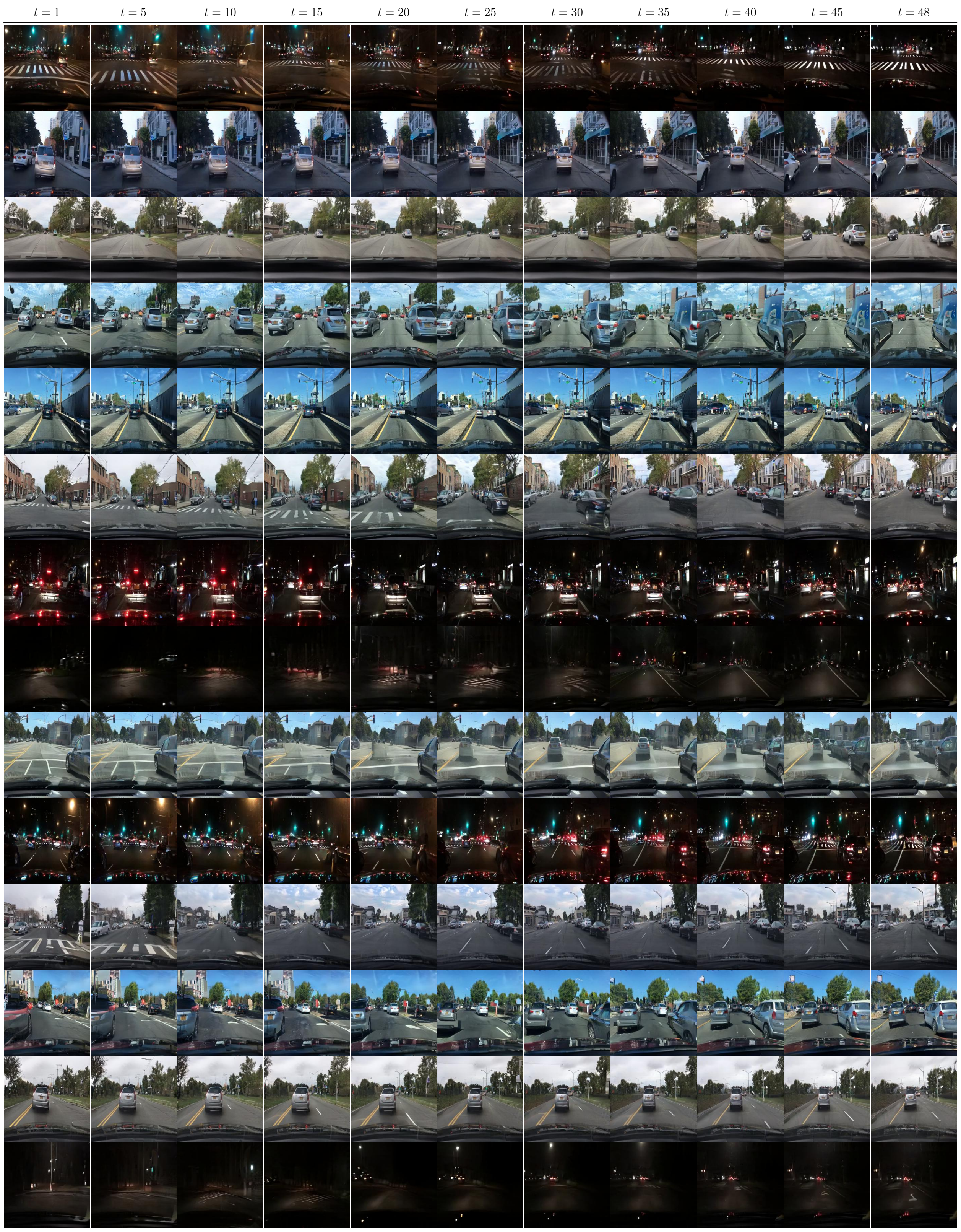


**Figure 6.12 – Additional samples for BDD100K 48/256x256** We show additional samples from our three-level BDD100K 48/256x256 model.

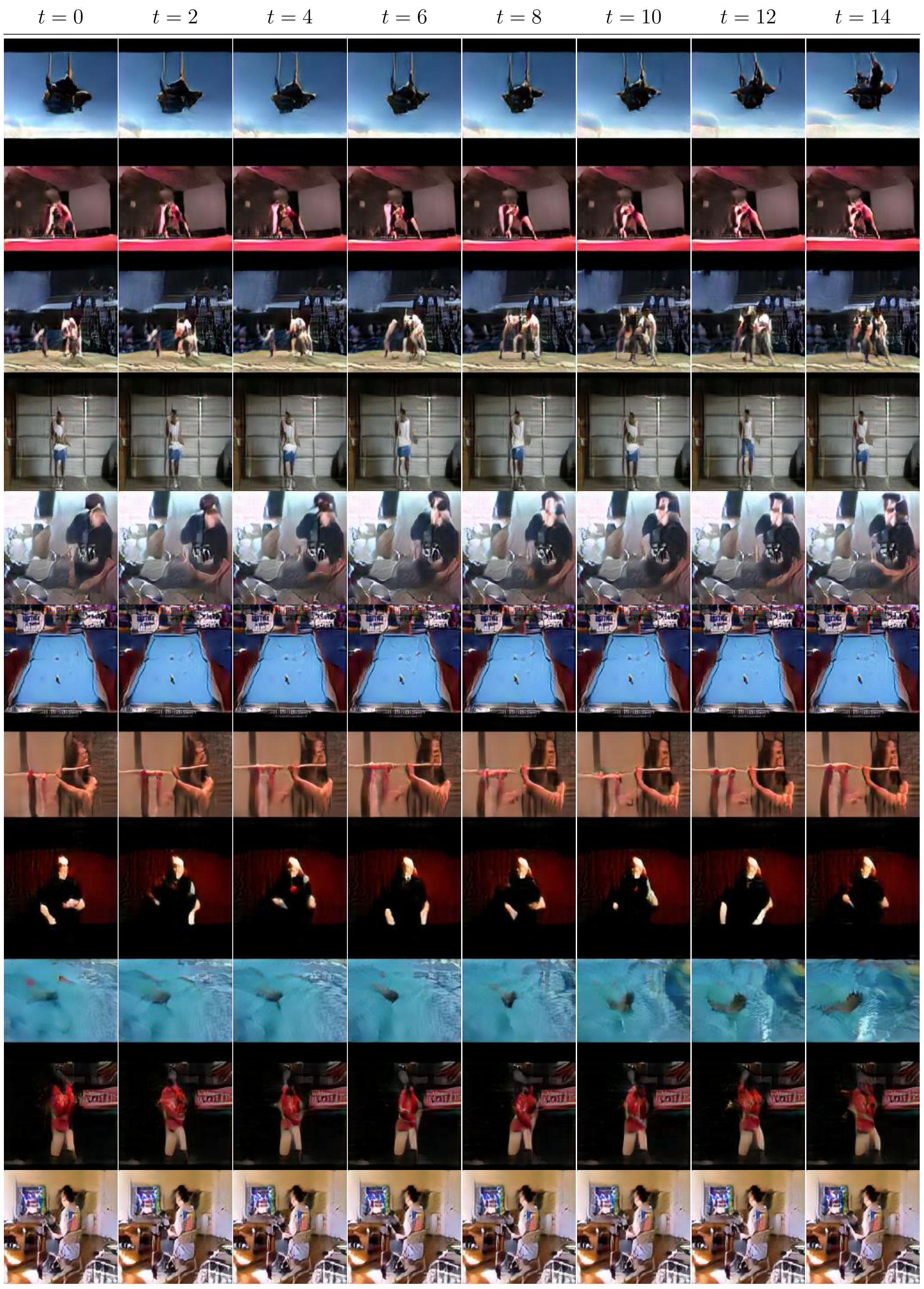


**Figure 6.13 – Samples from our model trained on UCF101** We show samples from our 16/128x128 model trained on UCF101.

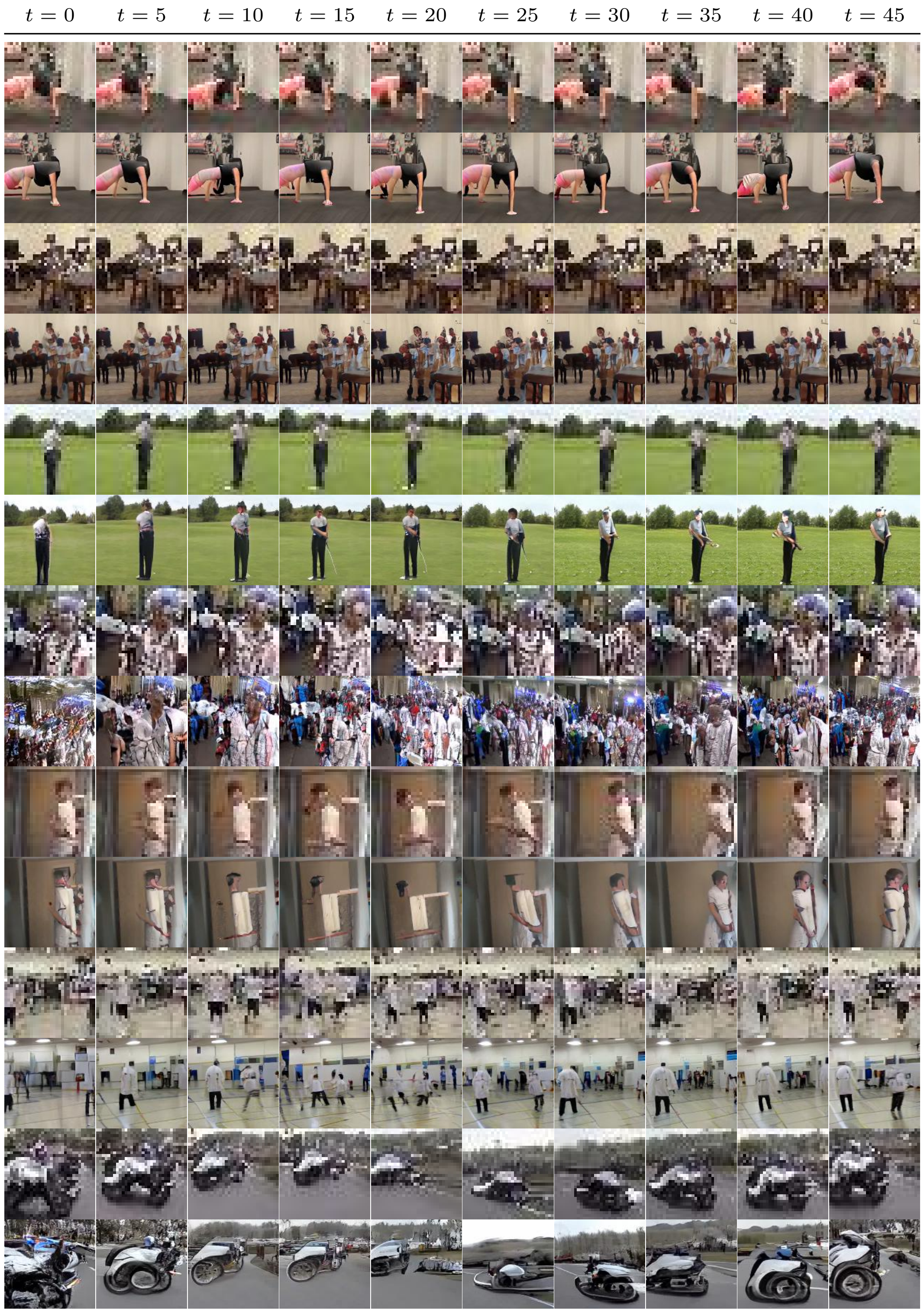


**Figure 6.14 – Pairs of samples from stage 1 and their corresponding stage 2 output** We show a few examples from our 12/128x128 two-level model trained on Kinetics-600 and unrolled to generate 48/128x128 videos. For each example, we show the first level low resolution generation and the corresponding level 2 upsampling. Level 2 outputs refine the details of the first level generations but retain the overall scene structure.

# Chapter 7

# Prologue to the Third Article

## 7.1. Article Details

**VIM: Variational Independent Modules for Video Prediction.** Rim Assouel[1], Lluis Castrejon[1], Nicolas Ballas, Aaron Courville and Yoshua Bengio. In *Proceedings of the Conference on Causal Learning and Reasoning (CLeaR 2022).*

**Authors contributions.** I came up with ideas to improve the model, implemented later versions of the model, ran some of the experiments and co-wrote the paper. Rim Assouel implemented the first version of the model, ran some of the experiments and co-wrote the paper. Nicolas Ballas came up with the model idea and provided support with the experiments and writing. Aaron Courville and Yoshua Bengio supervised the project.

## 7.2. Context

While video prediction and generation had benefited from advances in generative modeling, it seemed we had hit a ceiling in terms of the kind of dataset and generations we were able to produce. High capacity models were still not capable of generating detailed videos at high resolutions, and GANs and autoregressive models required extensive computational resources. Due to this observation we began to look at building better inductive biases into video prediction models. In particular, for this project we wanted to design video prediction models that consider objects independently and learn dynamics rules that are shared among objects.

1. Equal Contribution

## 7.3. Contributions

We propose a model for video prediction that automatically learns to decompose a scene into entities or objects. Our model learns a set of dynamics rules and selects one of the rules to update object states at each time step. These transition functions are shared among objects, following the intuition that dynamics rules are common to all physical entities. We show that the transition functions learned by our model are interpretable, and we show that our model decomposition allows it to generalize to out-of-distribution settings not seen during training.

## 7.4. Recent Developments

This article has been published recently, and therefore it is early to analyze its impact. Our method follows previous work (Goyal et al., 2021, 2020, 2019) that factorizes dynamic scenes into entities and transitions. Contrary to these works, our model is able to handle stochasticity due to its ability to choose different dynamics transitions for each object at each time step. Further, our model is also inspired by the literature on unsupervised scene decomposition (Locatello et al., 2020; Burgess et al., 2019a), although our model is centered on dynamic scenes. Recently there has been follow-up work on Slot-Attention that extends it to video (Kipf et al., 2021). We hope that our work will inspire researchers to build video prediction models with better inductive biases.

# Chapter 8

# VIM: Variational Independent Modules for Video Prediction

**Abstract.** We introduce a variational inference model called VIM, for Variational Independent Modules, for sequential data that learns and infers latent representations as a set of objects and discovers modular causal mechanisms over these objects. These mechanisms - which we call modules - are independently parametrized, define the stochastic transitions of entities and are shared across entities. At each time step, our model infers from a low-level input sequence a high-level sequence of categorical latent variables to select which transition modules to apply to which high-level object. We evaluate this model in video prediction tasks where the goal is to predict multi-modal future events given previous observations. We demonstrate empirically that VIM can model 2D visual sequences in an interpretable way and is able to identify the underlying dynamically instantiated mechanisms of the generation process. We additionally show that the learnt modules can be composed at test time to generalize to out-of-distribution observations.

## 8.1. Introduction

Predicting future events is believed to be a fundamental function of the human brain (Clark, 2013; Mullally & Maguire, 2014) having implications for representation learning, planning or counterfactual reasoning. Humans have the ability to decompose a visual scene into abstract objects and predict their changes far into the future (Kahneman et al., 1992; Spelke et al., 1993). Most interestingly, humans can adapt to a new situation quickly by re-using relevant prior knowledge about objects and their dynamics. In particular, they are good at generalizing in a compositional way because they represent knowledge as re-usable components that can be further composed to explain new observations.

Recent work (Eslami et al., 2016; Burgess et al., 2019b; Greff et al., 2019b; Crawford & Pineau, 2019; Locatello et al., 2020) have made significant progress in learning unsupervised entity-centric representations of images and a few of them (Lin et al., 2020; Kossen et al., 2019; Jiang et al., 2019a; Crawford & Pineau, 2020; Kosiorek et al., 2018) have extended those models to generate both deterministic and stochastic videos. However, except for SCOFF (Goyal et al., 2020), to the best of our knowledge none of them have considered architectures that disentangle the underlying dynamic rules of the generation process and rather model the transition part of their models with a shared monolithic module that is applied at each time step. We extend the architectural inductive biases from SCOFF and incorporate them in a probabilistic generative model of videos.

In this work we propose a variational framework to learn both entity-centric representations and entity-centric transition modules. We show that in a simple 2D stochastic environment we are able to identify the exact generating factors of variation both in terms of declarative (e.g. constitutive objects of the visual scene) and procedural knowledge (e.g. transitions rules that govern the stochastic dynamics of the objects). We argue that having those 2 layers of structure in a generative model is an essential first step towards generalizing out-of distribution, particularly when the new unseen data results from a composition of seen dynamic rules.

Our model called Variational Independent Modules (VIM), is a probabilistic generative model where both latent states and transitions functions over these latents have an entity-centric inductive bias. It thus learns a latent state composed of a set of abstract entities, or slots (Locatello et al., 2020) and a set of stochastic transition functions over the entities. These transitions functions, which we call modules, are independently parametrized and are shared across entities, following the principle of reusable independent causal mechanisms (Peters et al., 2017; Goyal et al., 2019, 2020). At each time step $t$, our model infers a set of categorical latent variables (called *selection variables* $\mathbf{r}^t$) to select which transition modules to apply to each represented entity in the latent set $\mathbf{z}^t$. We train VIM using variational inference over both the entity-centric set of $K$ latents $\{\mathbf{z}_k\}_{k=1,,K}$ and their respective selection variables $\{\mathbf{r}_k\}_{k=1,,K}$. In particular, the distribution of the categorical selection variables is implemented with a key-query attention mechanism (Bahdanau et al., 2014; Vaswani et al., 2017) in which all the possible modules compete (Goyal et al., 2019, 2020, 2021; Locatello et al., 2020) to explain future states and they are sampled according to their attention importance weights.

One of the key assumptions behind our framework is that the abstract entities are evolving mostly independently and only interact sparsely with each other (Pearl, 2009; Goyal et al., 2019). Consequently, the causal graph is sparse (Bengio, 2017; Goyal & Bengio, 2020) and a module can only consider a handful of slots as input argument. However in this work we will mostly consider unary modules and leave the exploration of n-ary modules to model interactions between multiple entities for future work.

We evaluate VIM in a 2D stochastic setting where the goal is to predict multi-modal future events given previous observations and we make the following contributions:

— We propose a simple stochastic environment in which we know the rules of the dynamics, corresponding to the different modes in the transition distribution, to enable checking for correctly identified solutions (Locatello et al., 2018).

— We show via simulations that in this environment, our framework is able to identify the ground truth dynamics rules that govern the data generation process.

— We show that VIM is able to generalize in a compositional and interpretable way in a simple out-of-distribution (OOD) tracking task.

## 8.2. Background : Recurrent State Space Models

Given a set of $D$ observed frames $\mathbf{c} = (c_1, ..., c_D)$ and the $T$ following future frames $\mathbf{x} = (x_1, ..., x_T)$, our goal is to learn a generative model that maximizes the probability $p(\mathbf{x}|\mathbf{c})$. To solve this task, our proposed model builds upon the Recurrent State-Space Model (RSSM) (Hafner et al., 2019).

RSSMs define a variational framework to model sequential data. They introduce a sequence of latent variables $\mathbf{z} = (z_1, z_2, ..., z_T)$ to capture the stochasticity of the observation at each time step such that the joint distribution is factorized as:

$$p(\mathbf{x}, \mathbf{z}|\mathbf{c}) = \prod_{t=1}^{T} p(x_t|\mathbf{z_{\leq t}})p(z_t|\mathbf{z_{<t}})p(z_0|\mathbf{c}). \tag{8.1}$$

where $p(x_t|\mathbf{z_{\leq t}})$ is the likelihood model, $p(z_t|\mathbf{z_{<t}})$ is the prior transition model, $c$ is some given context and $p(z_0|\mathbf{c})$ the discovery model that maps the observed context to a distribution over the initial latent state. The main advantage of RSSMs over previous autoregressive models is computational: it can make multi-step future predictions without having to render/encode observed frames at each time step.

RSSMs are trained using variational inference (Jordan et al., 1999), using an amortized approximate posterior $q(\mathbf{z}|\mathbf{x}, \mathbf{c}) = \prod_{t=1}^{T} q(z_t|\mathbf{z_{<t}}, \mathbf{x_t})p(z_0|\mathbf{c})$ where $q(z_t|\mathbf{z_{<t}}, \mathbf{x_t})$ is called the posterior transition model and $q(\mathbf{z}|\mathbf{x}, \mathbf{c})$ approximates the true posterior distribution $p(\mathbf{z}|\mathbf{x},\mathbf{c})$. Training is done end-to-end by maximizing the Evidence Lower Bound (ELBO) (Kingma & Welling, 2013; Rezende et al., 2014).

## 8.3. VIM: Variational Independent Modules

Similar to (Lin et al., 2020; Kossen et al., 2019; Jiang et al., 2019a; Crawford & Pineau, 2020; Kosiorek et al., 2018), VIM is a probabilistic RSSM where the latent space is structured as a set of $K$ hidden vectors $h = \{h_k\}_{k=1..K}$ (e.g. slots (Locatello et al., 2020)) where each slot is supposed to represent an *abstract* entity of the input (e.g. visual objects). In VIM we add an additional layer of structure to account for the fact that only a few rules (Goyal et al., 2020) govern the dynamics of objects in the world and that these rules (that we call *modules*) are shared amongst entities. Our transition distribution $p_\theta(\mathbf{z}_t|\mathbf{z}_{<t})$ is thus parametrized with $M$ independent modules that compete against each other to explain future observations. The modules operate over entities and can have a predefined number of arguments. For our experiments, we only use unary modules that operate independently on a single entity. The high-level computation steps of our model's transition function are the following:

— For each entity $k$ we first compute all the $M$ possible next distribution given its current latent state $h_t^k$ defined by the $M$ independent transition modules.

— We then score each candidate with a learned key-query attention mechanism to define a categorical distribution over these possible futures represented by an entity-centric selection variable $\mathbf{r}_t^k$.

— Finally we sample the module index according to $\mathbf{r}_t^k$ and select the corresponding future candidate as the next step entity state $h_{t+1}^k$.

VIM can thus be interpreted as a RSSM with 2 latent variables, $\mathbf{r} = \{\mathbf{r}_t\}_{t=1..T}$ and $\mathbf{z} = \{\mathbf{z}_t\}_{t=1..T}$ where $\mathbf{r}$ is a module selection variable and $\mathbf{z}$ is a Gaussian additive update to hidden slot representation $h$. Its generative model is factorized as

$$p_\theta(\mathbf{x}, \mathbf{r}, \mathbf{z}|\mathbf{c}) = \underbrace{p_\theta(\mathbf{z}_0|\mathbf{c})}_{\text{discovery model}} \prod_{t=1}^{T} \underbrace{p_\theta(\mathbf{x}_t|h_{\leq t}^k)}_{\text{observation model}} \prod_{k=1}^{K} \underbrace{\underbrace{p_\theta(\mathbf{z}_t^k|h_{<t}^k, \mathbf{r}_t^k))}_{\text{latent update}} \; \underbrace{p_\theta(\mathbf{r}_t^k|h_{<t}^k)}_{\text{module selection}}}_{\text{transition model}} \tag{8.2}$$

where the observation model $p_\theta(\mathbf{x}_t|h_{\leq t})$ and the discovery model $p_\theta(\mathbf{z}_0|\mathbf{c})$ are both parametrized with a slot-attention network (Locatello et al., 2020). In the following we describe in more details the parametrization of both the inference and generation model of the transition part of VIM.

### 8.3.1. Generation

We denote the representation of slot $k$ at time-step $t$ as $h_t^k$. We initialize slots as $h_0 = \mathbf{z}_0$ and update them recurrently at each time-step: $h_t^k = g_\theta(h_{t-1}^k, \mathbf{z}_t^k, \mathbf{r}_t^k)$. We also introduce the set of $M$ possible future candidates for each entity $k$ and denote it $\mathbf{s}_t^k = \{f_{\theta_m}(h_{t-1}^k) =$

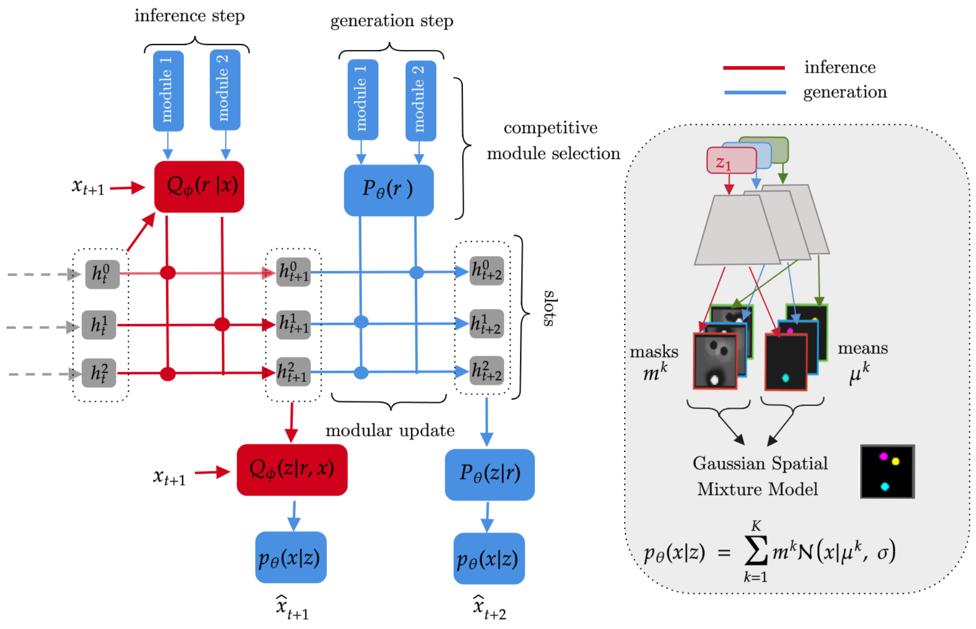


**Figure 8.1 – Overview of VIM** VIM disentangles both Objects and Dynamics with slots and transition modules. *Left*: structure of the transitions, controlled by the selection of an inferred rule $r$ and its application to the latent state $z$, and by an observation model of $x$. *Right*: generative decoder architecture mapping a set of object descriptions $z$ to an image where the different objects are rendered and composed using a Spatial Guassian Mixture Model.

$f_\theta(h^k_{t-1}, \mathbf{r}^k_t = m)\}_{m=1..M}$ where the modules are parametrized with $M$ independent MLPs whose parameters are indexed by $\{\theta_{m=1..M}\}$.

**Module Selection Prior.** In this step we describe the parametrization of the module selection prior $p(\mathbf{r}^k_{t+1}|h_{<t})$ where $\mathbf{r}^k_{t+1}$ defines a categorical variable that indexes which module to apply to entity $k$ at time step $t$. The probability of the categorical distribution from which $\mathbf{r}^k_{t+1}$ is obtained with a key-query attention mechanism where the keys are extracted from the $M$ candidates $\mathbf{s}^k_t$ and the query from a function of the current hidden state $h^k_t$. We then sample the module $m$ to apply with Gumbel softmax (Jang et al., 2016).

**Latent Update Prior.** The latent update is the result of 2 additive updates followed by a layer normalization such that at each time step $t$ we recurrently update slot $k$ : $h^k_t = LayerNorm(h^k_{t-1} + \mathbf{z}^k_t + f_\theta(h^k_{t-1}, \mathbf{r}^k_t)$ where $\mathbf{z}^k_t$ is sampled from a gaussian prior whose mean and variance are computed as follows: $\mu^t_k, \sigma^k_t = \text{MLP}(f_\theta(h^k_{t-1}, \mathbf{r}^k_t))$ and $\mathbf{z^k_t} \sim \mathcal{N}(\mu^t_k, \sigma^k_t)$.

## 8.3.2. Inference

In this section we describe the parametrization of our inference model, which is factorized as:

$$q(\mathbf{z},\mathbf{r}|\mathbf{x}) = \prod_{t=1}^{T}\prod_{k=1}^{K} q_\phi(\mathbf{z}_t^k|f_\theta(\mathbf{z}_{<t}^k,\mathbf{r}_t^k),\mathbf{x}_t)q_\phi(\mathbf{r}_t^k|\mathbf{z}_{<t},\mathbf{x}_t). \tag{8.3}$$

During inference we need to encode information about the target frame to infer the updates to apply to each entity slot $k$. However not all the visual information is needed for all the abstracted entities. Each entity needs to attend to a specific perceptual grouping of the target frame. To that end, we propose to use a slot-attention mechanism where the grouping is learned through a key-query attention mechanism. We denote $\mathbf{x}_t^k$ the perceptual grouping for entity $k$ obtained at time step $t$ such that :

$$\hat{\mathbf{x}}_k^t = \sum_i \beta_{k,i}^t V_i^t \text{ with } \beta^t = \text{softmax}(\frac{KQ^T}{\sqrt{D}}, \text{dim="slots"}) \tag{8.4}$$

where the keys $K$ are extracted from the current states $\{h_t^k\}_{k=1..K}$ and the queries and values are extracted from the target input $\mathbf{x}_t$, encoded with a size-preserving CNN backbone and a positional encoding.

**Module Selection Posterior.** We describe the rule selection posterior $q_\phi(\mathbf{r}_t^k|h_{<t},\mathbf{x}_t)$ for each entity $k$. Like with the module selection prior in generation, the idea is to score the set of candidates with a key-query attention mechanism where this time the query for entity $k$ is obtained from the entity-centric target encoding $\mathbf{x}_t^k$. The intuition behind this design choice is to select the update that best explains the current observation. We sample the module $m$ to apply with a Gumbel softmax trick.

**Latent Update Posterior.** The latent update posterior is similar to prior, where this time the Gaussian parameters of $\mathbf{z_t^k}$ are computed as a function of an entity-centric target encoding $\mathbf{x}_t^k$ such that: $\hat{\mu}_k^t, \hat{\sigma}_t^k = \text{MLP}([f_\theta(h_{t-1}^k,\mathbf{r}_t^k);\mathbf{x}_t^k])$ and $\mathbf{z_t^k} \sim \mathcal{N}(\hat{\mu}_k^t, \hat{\sigma}_t^k)$

## 8.3.3. Training

Training is done using variational inference maximizing the following evidence lower bound:

$$\begin{aligned}\mathcal{L}(\theta,\phi) = &\sum_{t=1}^{T} \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}_t|\mathbf{z}_t,\mathbf{z}_{<t})] - \underbrace{KL(q_\phi(\mathbf{z}_t|\mathbf{r}_t,\mathbf{x}_t,\mathbf{z}_{<t})||p_\theta(\mathbf{z}_t|\mathbf{r}_t,\mathbf{z}_{<t}))}_{\text{gaussian KL}} \\ &- \underbrace{KL(q_\phi(\mathbf{r}_t|\mathbf{x}_t,\mathbf{z}_{<t})||p_\theta(\mathbf{r}_t|\mathbf{z}_{<t}))}_{\text{module KL}}\end{aligned}$$

where there is a tension between explaining the multi-modality of the future outcomes either using the Gaussian distribution of a single module (but this would mean trying to fit a unimodal Gaussian distribution to multimodal observations and would result in a high noise KL) or using several modules indexed by $\mathbf{r}$ to explain the different modes. The module KL also prevents the model to learn duplicate transition modules and we verify this experimentally by training the model with more modules than actual modes in the distribution. In that case, additional modules are simply rarely selected during inference.

## 8.4. Related Work

**Video Prediction.** Since the initial models inspired by language modelling (Ranzato et al., 2014; Srivastava et al., 2015a), video prediction has seen great progress leveraging advances in generative models and deep learing architectures. The temporal and spatial dependencies between pixels is typically captured via autoregressive models (Larochelle & Murray, 2011; Dinh et al., 2016; Kalchbrenner et al., 2017; Reed et al., 2017; Weissenborn et al., 2020) or latent variables models such as the VAE (Kingma & Welling, 2013; Chung et al., 2015; Denton & Fergus, 2018; Lee et al., 2018; Castrejon et al., 2019; Villegas et al., 2017a,b) or GAN (Goodfellow et al., 2014; Vondrick et al., 2016a; Mathieu et al., 2015). Our approach uses variational inference for training, like other VAE-like models. Most of those previous models, however, rely on fixed-size unstructured vectorial representations of the state and monolithic prediction models while we explore the use of structured latent space (as a set of slots) and modular architectures (for the mechanisms) to explain multimodality in the transitions.

**Unsupervised Object Discovery.** A recent research direction explores unsupervised object-centric representation learning from visual inputs. The main motivation behind this line of work is to disentangle a latent representation in terms of objects composing the visual scene. They can be divided into two types of models: on one hand, the spatial mixture models (Locatello et al., 2020; Burgess et al., 2019b; Greff et al., 2019b) learn a set of unstructured latents that are decoded into a pixel-wise mean and mask such that each pixel location defines a Gaussian mixture model weighted by the slot masks. On the other hand, spatial-transformer based models propose to further disentangle each slot latent representation into several variables (e.g. content, location, presence, depth) where the location variable parametrizes the input to a spatial transformer that fills a canvas, as in (Eslami et al., 2016; Crawford & Pineau, 2019; Stelzner et al., 2019; Lin et al., 2020). The contribution of our work is orthogonal and we want to show the benefits of disentangling the mechanisms that process the slots in the transition module of a sequential generative model when the target distribution has multimodal uncertainty. Lin et al. (2020) argues that a careful design of the prior is needed to account for multimodal future trajectories and propose to do so with a slot-wise hierarchical Gaussian prior model. In this work we propose an *interpretable* alternative to account for several modes in the target distribution and show that a modular approach is capable of capturing the underlying factors (e.g. modes in our case) of the generating process. In terms of architectural components, we extend Locatello et al. (2020)'s slot attention module to a sequential setting where the slot attention module is used to encode a meaningful part of the target frame in a slot-wise manner. We show that by using independently parametrized modules in the recurrent transition function, our approach is able to discover object-centric dynamical rules in an unsupervised manner and that the slot attention inference machinery is able to select compositions of those dynamical rules at test time.

**Independent Mechanisms.** Recent approaches have explored architectures composed of a set of independently parametrized modules which compete with each other to communicate and attend or process an input (Goyal et al., 2019, 2020, 2021). Those architectures are inspired by the notion of independent mechanisms (Pearl, 2009; Goyal et al., 2019), which suggests that a set of independently parametrized modules capturing causal mechanisms should remain robust in case of distribution shift due to an intervention, as adapting one should not require adapting the other modules. The recurrent independent mechanisms architectures (Goyal et al., 2019, 2020, 2021) however are not probabilistic models; they cannot capture the uncertainty inherent with future predictions. In this work we formulate the same intuition of mechanisms separation in a variational inference framework where the selection of the the mechanisms is expressed as a categorical random variable whose posterior distribution the model must infer. We further showcase an interpretability advantage in the case of a simple 2D stochastic environment.

## 8.5. Experiments

In this section we describe experiments designed to showcase three main properties of VIM: i) object-centric *multi-modality* of the predictions in stochastic environments, ii) *interpretability* of the learned modules, and iii) *generalization* to compositional out-of-distribution settings in an interpretable way.

### 8.5.1. Interpretability

In this section we show that the modular architecture of VIM is able to capture the underlying generating factors of the target distribution in an interpretable way. Through an ablation study on the number of available modules we also show that with enough modules to capture the different distribution modes precisely, VIM performs better than models that rely on a unimodal prior.

**Dataset.** We evaluate VIM on a dataset built to showcase interpretability of the modules. This dataset, called the *Random Walk Dataset*, consists of 2D image sequences of multiple colored-balls that evolve on a black background. At each time step each ball can move randomly in one of the 4 cardinal directions. The resolution of the videos is $32 \times 32$ and the model is trained on up to 15 consecutive frames.

**Setup.** We train VIM with $N$ modules and $K = 4$ slots on trajectories that contain 3 balls. During training, the selection variable $\mathbf{r}$ is sampled using a Gumbel-softmax (Jang et al., 2016) with a fixed temperature of 1 and we use the hard version during testing so that the variables are categorical. For evaluation we match each ground truth ball of the visual scene to a latent slot explaining it. Note that Spatial-Transformer based models (Jang et al., 2016; Crawford & Pineau, 2019) directly exploit the slot bounding box center coordinate to compute this matching. Our model does not consider object bounding boxes. Instead, we use the slot-wise masks outputted by the observation model to match them to objects. More specifically, to determine the location $(x_k, y_k)$ of the object explained by the $k$-th slot we sample coordinates uniformly over the 2D space covering the whole frame and then we average these coordinates weighted by the value of the slot mask value at each location $m_k^{x,y}$, such that:

$$x_k, y_k = \frac{\sum_{(x,y)\in[1,H]\times[1,W]}(x,y) * m_k^{x,y}}{m_k^{x,y}}.$$

Once we have those slot-wise coordinates we compare them against ground truth balls positions and match a slot with the closest ball position in the first frame and keep the same matching for the rest of the sequence. We then use these matched pairings to compute ball position errors for tracking purposes or to compare the slot-wise module selection variable to the ground-truth action transition.

**Results.** Our objective is to verify that the learnt modules can directly be interpreted as causal dynamics rules underlying the generation process for the Random Walk dataset. We show that each module correspond to a particular mode of the generating distribution (e.g. moving in one of the cardinal directions). To do so, we evaluate our inference model on some test sequences of length 20 and store the selection variables of all the slots $((\mathbf{r}_k^t)_{k=1..K})_{t=1..20}$.

Using the first set of slots of the sequence we match each ground truth ball with its explaining slot (whose location has been obtained with its decoded mask) by choosing the closest one. We denote the slot associated with the $d$-th ball by $i_d$. We compare at each time step $t$ the ground truth transitions of each ball $d$ with the index of the module that slot $i_d$ has selected and report the proportion of the correct correspondences. The percentage in Figure 8.2 shows that each cardinal transition can be explained by one of the modules. We further visualize the repeated effect of each module on a single ball to confirm that correspondence. Figure 8.2 shows that four of the modules move a ball in one of the cardinal direction exclusively. When the model is trained with more modules than actually needed (e.g. more modules than modes in the distribution), then additional modules are not used. This behaviour is in part encouraged by the module KL term of the loss which will be larger if 2 modules are identical: given the way we compute the module selection weights with a key-query attention machinery, modules that explain the same mechanisms would have the same selection probability. We show in Figure 8.2 the correspondence between selected modules and ground truth actions for VIM trained with 5 modules. In this case, only 4 modules appear to be selected by the slots that contain balls.

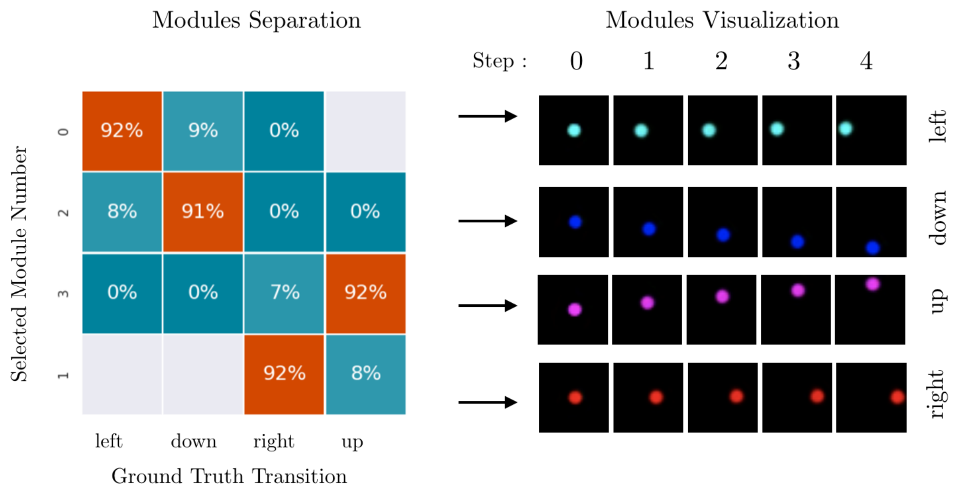


**Figure 8.2 – Discovery of Dynamics Rules** The proposed VIM model learns to match modules to ground truth entity transitions. *Left:* For each rule, we show the distribution over selected modules for a 4-module model. Modules specialize to implement one rule, with additional modules not being used. *Right:* We show the effects of repeatedly applying a module to a slot. Module transitions are interpretable and correspond to ground-truth transitions.

**Effect of the number of Modules.** In this section we study the effect of the number of available modules to model the sequences in terms of generation abilities. The model is trained on the Random Walk dataset with a varying number of modules. Larger per-module capacity is given to models with less modules to factor out the effect of total number of parameters in the modeling ability. In Figure 8.3 we identify three regimes: monolithic transition module (e.g. 1 Module does the work), transition function with less modules than actual modes in the distribution (e.g. 2 modules share the work), and a transition function with more modules than actual modes in the distribution (5 and 7 modules). The monolithic transition model can only rely on the unimodal Gaussian distribution to capture the multimodal stochasticity of the transitions, which results in lesser ability to capture multimodality. In the low modules regime, the model separates the modules to explain the modes in a hierarchical way, but some modes are shared between modules which results in lower modeling capacity than the model with enough modules to explain all the modes. In the case of 5 modules, module number 1 is never selected and thus doesn't appear in the color map. When VIM has more modules than actual modes in the distribution, additional (and non useful) modules are almost never selected. We show the resulting modules specialization and mode-module matching in Figure 8.3.

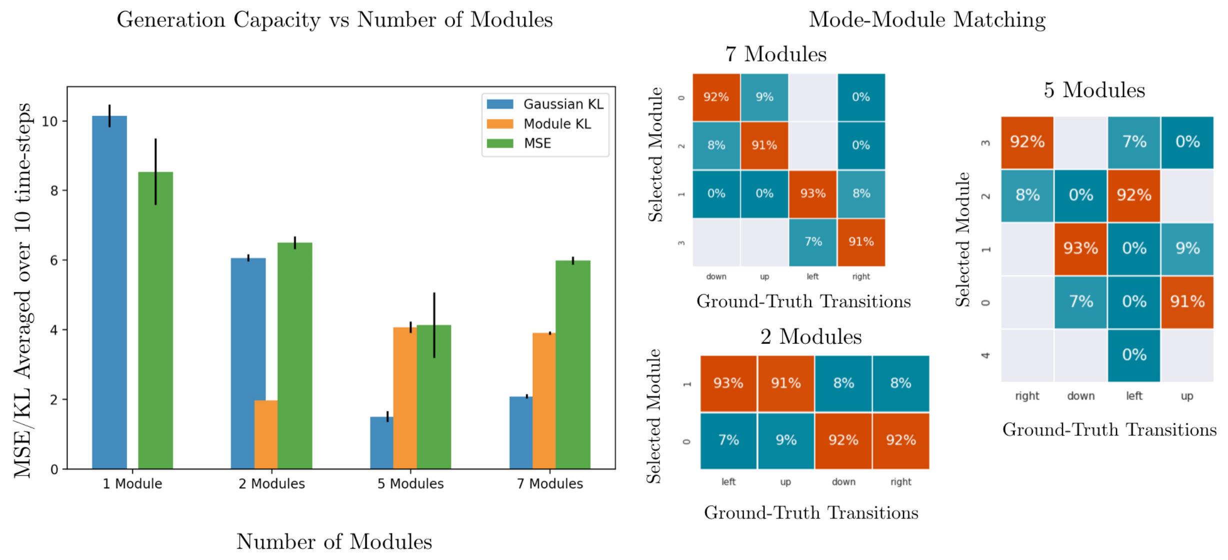


**Figure 8.3 – Ablation of the number of modules** *Left:* MSE/KL of the inference model averaged over 10 time steps for VIM with a varying number of modules *Right:* Modules specialize to each explain a mode in the distribution when enough modules are available.

## 8.5.2. Compositionality

**Dataset.** In this section we introduce a variant of the Random Walk dataset where the coloured balls dynamics at each step is a composition (e.g. left-up-up-left) of the previous atomic cardinal transitions. Between two consecutive frames, we sample the sequence of modules that composes the transition of each ball at random. When trained on this dataset, the model never sees the the atomic transitions that compose the random walk of each ball. We call it the Compositional Random Walk.

**Setup.** We introduce a variant of VIM where at each time step we allow the model to perform multiple selections and update steps before rendering. The rationale behind this variant is that for sequences whose transition dynamics rules are compositional we want to check whether VIM is able to compose some atomic building blocks (e.g. the modules) to explain an observation both at training and testing time. When the model is allowed $k$ iterations before rendering we simply denote the variant VIM-$k$.

**Results.** The insights we derived from the results of these experiments are two-fold:

— When trained on the atomic setting, VIM generalizes in an interpretable manner to compositions of transitions seen during training.

— When trained on the compositional setting, the individual modules that are learned in the multi-step version of VIM are still interpretable in terms of atomic transitions that were never seen during training.

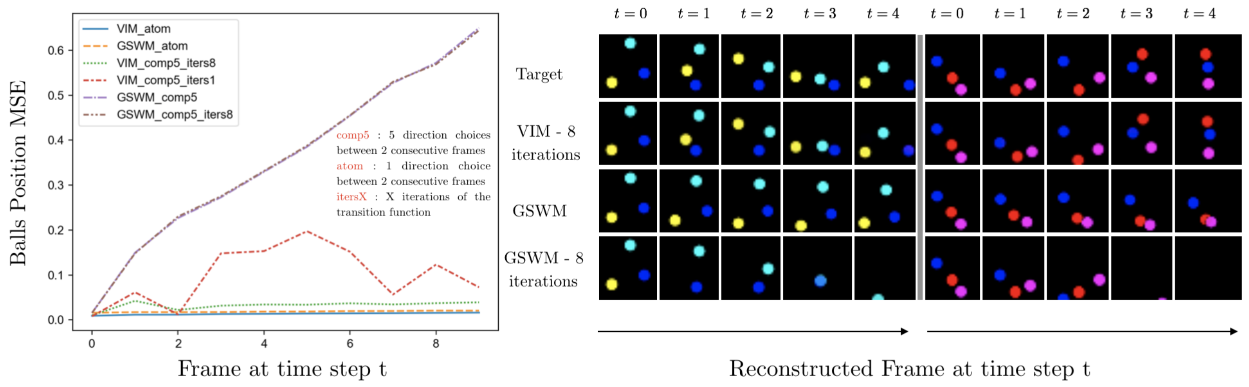


**Figure 8.4 – OOD Tracking**. VIM's ability to track to out-of-distribution composition of transitions seen during training. Compositional dynamics tracking comparison of VIM and G-SWM and their respective multi-step versions. Both trained on the atomic setting and are compared qualitatively *Right*: and quantitatively *Left* reporting their respective ball position errors.

**OOD Compositional Generalization.** We train VIM on the atomic setting and test it on out-of-distribution compositional sequences where up to 5 choices of atomic transitions are allowed for each ball between two consecutive frames.(e.g. *left-down-right-down-right*). At test time we consider a multi-step version of VIM where 8 iterations of the selection-update step are allowed in between frames. We show that the inference key-query selection part of VIM acts at each iteration as a kind of planner in the space of modules and selects the module that best explains the current observation. In Figure 8.5 we render the resulting frame at each iteration and notice that VIM composes modules in an iterative manner such that each ball progressively gets closer its target position before oscillating around it. Additionally, in Figure 8.4 we compare the tracking ability of a state-of-the art object-centric video generation model, G-SWM (Lin et al., 2020), with this multi-step version of VIM . We show that both the key-query selection mechanism and the composable modules enables VIM to better track objects that have a different amplitude than the one seen during training. For a fair comparison we also consider a multi-step version of G-SWM where several iterations of the same monolithic propagation module is allowed between 2 consecutive frames.

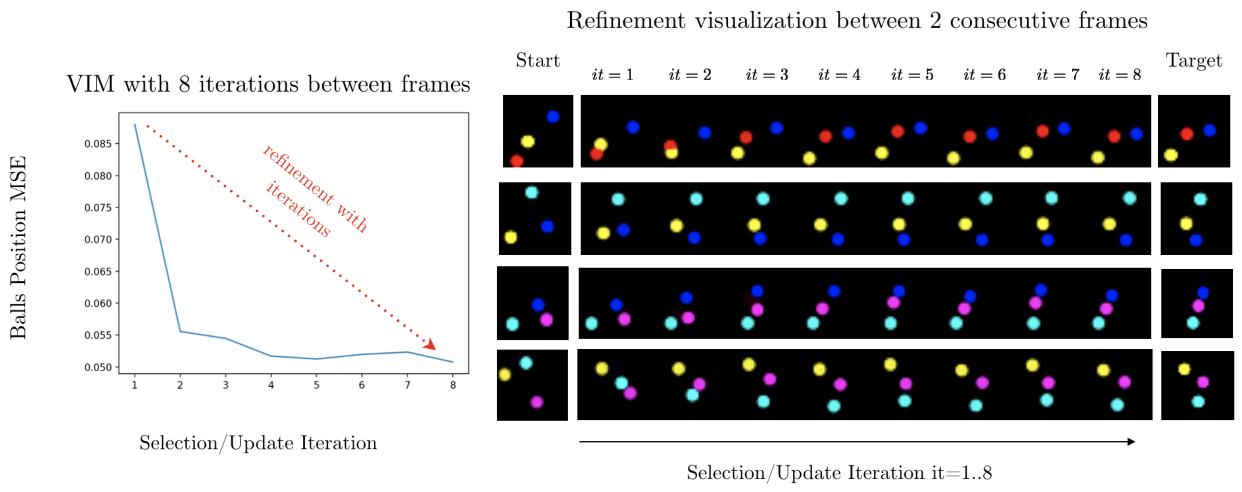


**Figure 8.5 – Iterative Refinement**. We show that VIM can model out-of-distribution compositions of transitions. *Left:* VIM trained on the Random Walk Dataset and tested on the Compositional Dataset allowing 8 iterations between frames. Between 2 consecutive frames, the ball position error decreases with the number of iterations. *Right*: Visualization of the module iterations. VIM iteratively selects modules to modify the starting observation and reconstruct the target frame.

**Multi-step Interpretability.** We train a 2-step version of VIM on the compositional setting and restrict the number of modules to 5 so that we still have more modules than atomic transitions (that are never seen) but less modules than possible compositional transitions seen during training (e.g. *left-right, up-left, down-down*, and so on). We test this model on sequences of atomic transitions to evaluate the proportion of selected modules corresponding to each ground truth transition. We show in Figure 8.6 that the same modularisation emerges as when VIM is trained on the atomic setting. In this compositional setting, when evaluated on atomic transitions we observe the same module specialization as when it was trained on atomic transitions. This shows that when trained with 2 selection/update iterations between 2 consecutive frames VIM successfully identifies the underlying atomic transformations that compose the transitions VIM has been trained on without directly observing them.

### 8.5.3. Dynamics Transfer

In this section we are interested in showing that VIM learns a factorized representation formed by entities and transition rules over these entities. In particular we would like to evaluate whether VIM is able to recognize known dynamics applied to unseen objects and

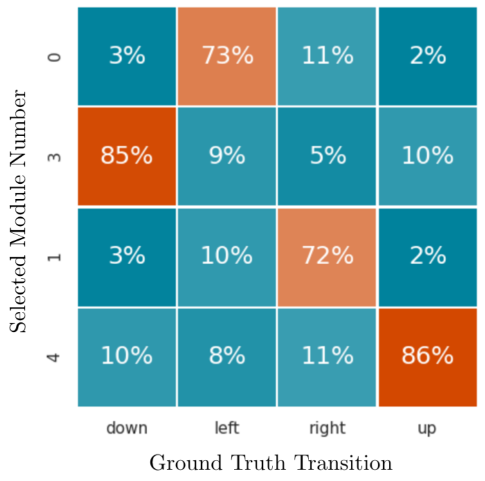


**Figure 8.6 – Rule-Module matching for compositional transitions** We show a rule-module matching histogram for a VIM model with 5 modules and 2 module iterations per frame trained on the Compositional Random Walk dataset. We observe that, despite only observing compositions of actions, the modules implement and match ground-truth atomic transitions.

that the inference machinery proposes to transfer those dynamics to known objects using the selection variables for these dynamics.

**Dataset.** In this section we test VIM on a dataset where objects have out-of-distribution attributes (e.g. shape, color, size), beyond those seen during training, but following the same Random Walk dynamics seen during training: they can go in one of the four cardinal directions between two consecutive frames and with the same amplitude.

**Results.** We train VIM on the Random Walk Dataset with coloured balls of the same size and test it on a OOD version where objects have different shape, colour and size than the balls seen during training. During testing, we run the inference model on these OOD target sequences $(\mathbf{x}_t^{target})_{t=0..T}$ and extract the selection variables at each time step $(\mathbf{r}_t^{target})_{t=1..T}$. We then consider an image $c$ where balls with known attributes are placed at the exact same location as the first frame of the target sequence and transfer the extracted target dynamics such that the rendered sequence is sampled following :

$$\{\mathbf{x}_t^{transfer}\}_{0:T} \sim \prod_{t=1}^{T} p(\mathbf{x}_t^{transfer}|\mathbf{z}_t) \prod_{k=1}^{K} p(\mathbf{z}_t^k|f(\mathbf{z}_{t-1}^k, \mathbf{r}_t^{target,k}))p(\mathbf{z}_0^k|\mathbf{c}).$$

In Figure 8.7 we show that VIM is able to recognize known dynamics of objects that have OOD attributes. VIM has successfully factorized individual objects identity and the shared dynamic rules that are applied in the same way irrespective of the objects identity.

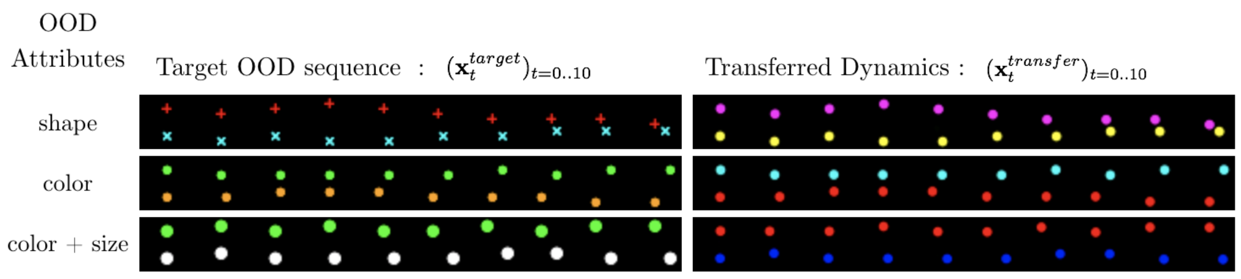

OOD Attributes
Target OOD sequence : $(\mathbf{x}_t^{target})_{t=0..10}$
Transferred Dynamics : $(\mathbf{x}_t^{transfer})_{t=0..10}$
shape
color
color + size

**Figure 8.7 – Transfer of dynamics with OOD objects**. VIM is able to recognize and transfer known dynamics of novel objects with OOD attributes. *Left*: Target trajectory from OOD objects from which we extract the module selection variables. *Right*: Transferred dynamics on known objects by applying the same selected modules as for the target sequences.

## 8.6. Conclusions

VIM introduces a variational framework to disentangle both the constituent entities (slots) and the generating factors (rules) of object dynamics in stochastic scenes. We show that the modular architecture of VIM captures different dynamics rules in an interpretable manner. Moreover, VIM modules can be composed at test time to explain OOD dynamics not seen during training. Additionally, we show that VIM is able to recognize the dynamics of objects with OOD attributes and transfer them to known objects by re-using the learned modules on different slots. The work presented here could be extended in different ways. For example, we could add interactions between slots through the use of n-ary modules. We could additionally scale up the model architecture and capacity to more complex datasets with in-the-wild 3D scenes and test its performance on downstream tasks. We plan to explore these directions in follow-up work.

**Acknowledgements.** We thank Facebook AI Research and Mila Quebec AI Institute for managing the computer clusters on which this research was conducted. This work was supported by an IVADO PhD Fellowship to L.C., an Antidote PhD Fellowship to R.A and by funding from CIFAR and NSERC.

## 8.A. Appendix - Additional Details

In this section we provide additional details. The architecture of the slot attention encoder is described in Table 8.1, the broadcast decoder is described in Table 8.2, while the architecture of the transition modules is described in Table 8.3.

**Table 8.1 – Slot Attention CNN Encoder**

| Type | Size/Channels | Activation | Other |
|---|---|---|---|
| Conv $5 \times 5$ | 64 | ReLU | stride 1 |
| Conv $5 \times 5$ | 64 | ReLU | stride 1 |
| Conv $5 \times 5$ | 64 | ReLU | stride 1 |
| Conv $5 \times 5$ | 64 | ReLU | stride 1 |
| Positional Encoding | 64 | - | LayerNorm(64) |
| Linear | 64 | ReLU | - |
| Linear | 64 | ReLU | - |

**Table 8.2 – Broadcast Decoder**

| Type | Size/Channels | Activation | Other |
|---|---|---|---|
| Spatial Broadcast | $H \times W$ | - | - |
| Positional Encoding | 64 | - | - |
| Conv $5 \times 5$ | 64 | ReLU | stride 1 |
| Conv $5 \times 5$ | 64 | ReLU | stride 1 |
| Conv $5 \times 5$ | 64 | ReLU | stride 1 |
| Conv $5 \times 5$ | 4 | ReLU | stride 1 |
| Split Channels | RGB(3) masks (1) | softmax on masks | - |
| Recombine slots | - | - | spatial mixture |

**Table 8.3 – Architecture of the Transition Modules**

| Type | Size/Channels | Activation | Other |
|---|---|---|---|
| Linear | 128 | ReLU | - |
| Linear | 128 | - | Layernorm(128) |
| Candidates | - | - | 1 module example |
| Linear | 128 | ReLU | - |
| Linear | 64 | ReLU | - |
| Stochastic update : | - | - | - |
| Linear | 64 | ReLU | - |
| Linear | 128 | - | - |
| Chunk | Mean (64) Std (64) | - | - |

### 8.A.1. Key-query Selection Mechanism

The key and query network that are used for the module selection bottleneck are implemented with simple 2 layer MLP of hidden sizes [64, 64] and a ReLU activation on the first layer. We used a fixed temperature of 1 for the Gumbel-Softmax computation.

### 8.A.2. Training Schedule

In our model, the slot attention module is first pre-trained on the first images of the videos following the same learning rate warmup suggested in (Locatello et al., 2020) for about 100 epochs to obtain initial slot separation. We found that it stabilized and speeded up training on the rest of the videos sequences. We trained the model on sequences of length 20 using a length schedule that starts at 6 frames. We increase the number of frames by two every 40 epochs.

## 8.B. Comparison with SCOFF/RIM

SCOFF (Goyal et al., 2020) proposes an architectural component to process a visual input such that it can be explained in terms $K$ object files (corresponding to slots in our model) and $M$ schemata (modules) processing those objects files. They propose to test their architecture in several downstream tasks and one of them is video prediction. To do so they train SCOFF on a next-step prediction task with an autoregressive formulation. RIM (Goyal et al., 2019) corresponds to a SCOFF with $M = 1$ schema (in other words, with a single module). At each time step $t$ the computations are conditioned on the previous frame. The model is trained using teacher forcing, and its training setup can be summarized as follows:

(1) Each object file $h^k_{t-1}$ competes to attend to a part of the previous input frame (similar to slot attention) $x^k_{t-1}$.

(2) For each object file, the $M$ possible slot candidates given by the application of each schemata are computed.

(3) Each object file selects the most plausible candidate with a key-query attention mechanism and updates its state.

(4) Each updated object file is then decoded to reconstruct the next-frame.

A key assumption in SCOFF and RIM is that the outcome is deterministic given previous frames. Similar to Babaeizadeh et al. (2018); Denton & Fergus (2018), we argue that this leads to blurry predictions in settings with stochastic dynamics, as the model is trying to capture the mean of all plausible future outcomes. To overcome this limitation we propose to include the intuition behind SCOFF's architectural inductive biases in a variational framework, where the schemata would rather correspond to the different modes/rules of the stochastic dynamics of the environment.

We show in Figure 8.8 a reconstructed sequence with a SCOFF model trained with 4 slots and 5 schemata on our Random Walk Dataset. We notice that:

— The object files capture the different objects in the scene, as slots do for VIM.

— Each object file attends to the part of the input that belongs to its represented object.

— The reconstructed frames show the mean position in all 4 possible directions, instead of capturing different directions with different schemata.

We also report in Table 8.4 the MSE averaged over 5 time-steps sequences for SCOFF (with teacher forcing at each time step) and VIM (using its inference network), both trained with 5 modules and 4 slots. We expect SCOFF to fail to produce sharp samples since it assumes the future is deterministic given the previous frame. whereas we expect VIM to be able to handle the multimodality of the future outputs.

| Model | Reconstruction MSE (↓) |
|---|---|
| RIM (Goyal et al., 2019) | $91.7 \pm 5.4$ |
| SCOFF (Goyal et al., 2020) | $81.9 \pm 1.0$ |
| VIM (Ours) | $\mathbf{4.2 \pm 0.5}$ |

**Table 8.4 – Comparison to RIM and SCOFF** We compare our model to RIM and SCOFF. All models use 4 slots/object files, and for SCOFF and VIM we use 5 modules/schemata. We report the reconstruction MSE averaged over sequences of 5 frames. The ground-truth previous frame is given to SCOFF/RIM at each time step. We observe that our model produces significantly better reconstructions of the input sequences. This is due to the sequences being stochastic, as RIM and SCOFF are deterministic models that cannot capture stochasticity in the environment dynamics. We propose VIM as a variational approach inspired by SCOFF that can multimodal future outcomes.

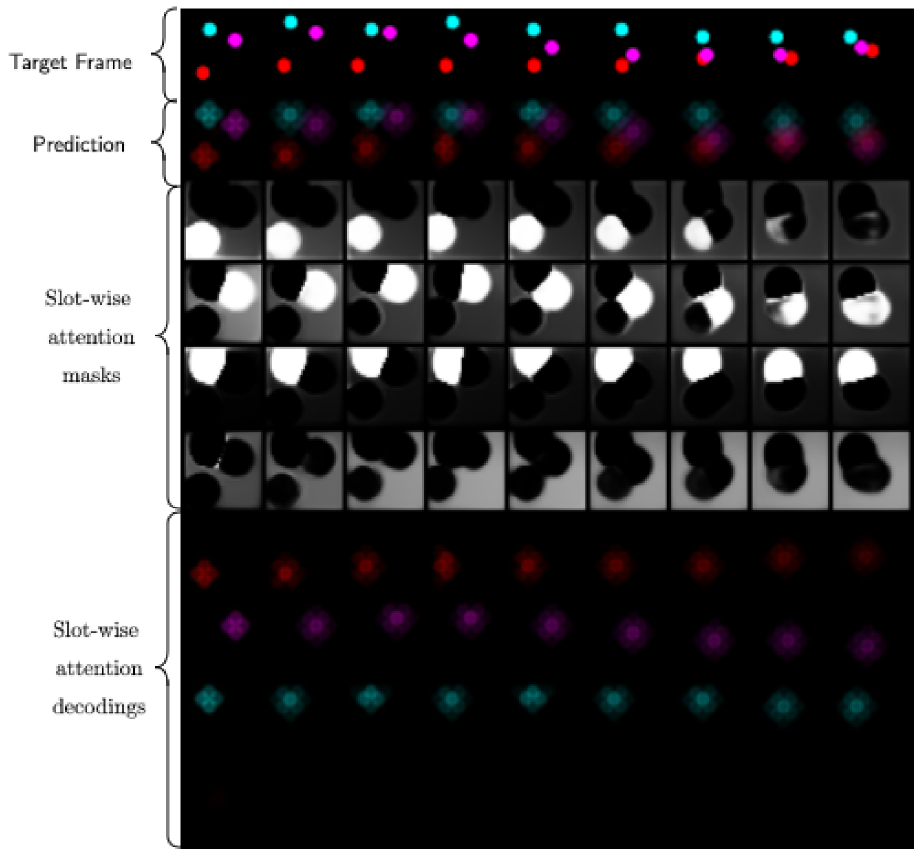


**Figure 8.8 – Sample from SCOFF** We show a sample of SCOFF in our proposed dataset. This sample illustrates the performance of RIM and SCOFF on this dataset. We observe that the slots correctly segment the input objects and model them as distinct entities. However, the model fails to produce sharp next step predictions, and instead produces the mean of all possible future outcomes from the previous frame. This is because RIM and SCOFF are deterministic models. Note that here we are showing sequences using teacher forcing.

# Chapter 9

# Prologue to the Fourth Article

## 9.1. Article Details

**INFERNO: Inferring Object-Centric 3D Scene Representations without Supervision.** Lluis Castrejon, Nicolas Ballas and Aaron Courville. Submitted to the *Conference on Lifelong Learning Agents (CoLLAs 2022)*.

**Authors contributions.** I developed and implemented the method presented in this article, ran most of the experiments and co-wrote the article. Nicolas Ballas helped with benchmarks and experiments and co-wrote the article. Aaron Courville supervised the project and revised the article.

## 9.2. Context

One of the limitations of current video prediction models is that they usually produce wrong predictions when showing object interactions. While it is not fully understood why, we argue that such events are often rare in training datasets and require generalizing beyond the 2D object appearance seen in previous frames of a video, as interactions often make objects change their 3D pose. At the same time Neural Radiance Fields (NeRFs) (Mildenhall et al., 2020) obtained really good 3D shape reconstructions from multiple views. In this project we use NeRFs to model scenes with multiple objects.

## 9.3. Contributions

Inspired by the limitations of video prediction models and the impressive reconstructions of NeRFs we propose INFERNO. INFERNO is a scene autoencoder that represents a scene as a set of objects (and a background). Each object in the scene is represented as a NeRF, and the representation explicitly captures the pose and appearance of each object. Scene representations are inferred from a single view of the scene and without requiring any annotations or ground-truth camera information. Through manipulating scene representations, we show that our model can generate out-of-distribution scenes not seen during training. Furthermore, our model is competitive with the state-of-the-art unsupervised scene decomposition methods and can be used for downstream scene understanding tasks such as the snitch localization task in CATER (Girdhar & Ramanan, 2019), which involves tracking an object in a dynamic scene.

## 9.4. Recent Developments

This article has been submitted recently, and as such it is too early to measure its impact. Previous work combined object-centric scene representations with NeRFs (Niemeyer & Geiger, 2021; Stelzner et al., 2021; Yu et al., 2021b). As opposed to our model, these methods either do not have an inference mechanism or do not explicitly model the pose of objects in the scene. We hope that INFERNO inspires work in object-centric scene representations that can be used for generating dynamic scenes and for downstream tasks. More specifically, we hope that future work can scale up our model to more realistic datasets.

# Chapter 10

# INFERNO: Inferring Object-Centric 3D Scene Representations without Supervision

**Abstract.** We propose INFERNO, a method to infer object-centric representations of visual scenes without relying on annotations. Our method learns to decompose a scene into multiple objects, with each object having a structured representation that disentangles its shape, appearance and pose. To impose this structure we rely on recent advances in neural 3D rendering. Each object representation defines a localized neural radiance field that is used to generate 2D views of the scene through a differentiable rendering process. Our model is subsequently trained by minimizing a reconstruction loss between inputs and corresponding rendered scenes. We empirically show that INFERNO discovers objects in a scene without supervision. We also validate the interpretability of the learned representations by manipulating inferred scenes and showing the corresponding effect in the rendered output. Finally, we demonstrate the usefulness of our object representations in a visual reasoning task using the CATER dataset.

## 10.1. Introduction

Inferring objects and their geometry in a scene is a fundamental ability of biological visual systems (Kahneman et al., 1992; Roelfsema et al., 1998; Spelke et al., 1993). Replicating this ability in machine is a promising step towards visual reasoning valuable to several applications involving object manipulation, navigation or forecasting.

Recent works (Jiang et al., 2019b; Locatello et al., 2020; Burgess et al., 2019a) have shown that neural networks can learn object-centric representations from low-level perceptual features. They learn to recognize the objects in a visual scene from a singe image without relying on supervision. However, most of those approaches only consider the 2D structure of images and ignore the underlying 3D geometry of the visual scenes. On the other hand, Neural Radiance Fields (NeRFs) (Mildenhall et al., 2020) have demonstrated that differentiable renderers can be combined with gradient-based optimization to learn high-fidelity 3D scene reconstructions. NeRFs have been subsequently used to learn 3D-aware generative models, including compositional scene models (Niemeyer & Geiger, 2021).

In this work, we leverage these recent advances in object-centric representation learning (Locatello et al., 2020) and 3D modelling through implicit functions (Mildenhall et al., 2020; Niemeyer & Geiger, 2021) and propose INFERNO, a model which infers a structured representation of objects and their poses from a single image. Each object is represented by latent variables characterizing its shape and appearance, together with an explicit representation of their poses (translation, scale and rotation). The object representations are then decoded using implicit functions that are localized in the scene according to the objects poses and combined together to generate a 2D output view. Our model does not need supervision and instead is fitted through minimizing a reconstruction loss, akin to an auto-encoder.

Disentangling the object appearance and pose in a scene representation allows the model to manipulate a visual scene. In particular, we demonstrate that INFERNO learns interpretable object poses, which we can modify and render to alter the pose of an object in a scene. We also validate that our approach learns meaningful representations for object discovery and visual reasoning. More specifically, we show that our approach obtains competitive performance on the CLEVR6 object discovery benchmark (Johnson et al., 2017; Greff et al., 2019a) as well as for the snitch localization visual reasoning task of CATER (Girdhar & Ramanan, 2019).

In summary, our contributions are the following:

— We propose a model able to infer and render 3D scene representations composed of multiple objects, each of them modeled by an implicit function and explicitly localized in the scene.

— We show that the representations learned by the model are interpretable and amenable to manipulations.

— We demonstrate that the inferred representations are useful for downstream tasks by showing competitive performance in object discovery and reasoning tasks.

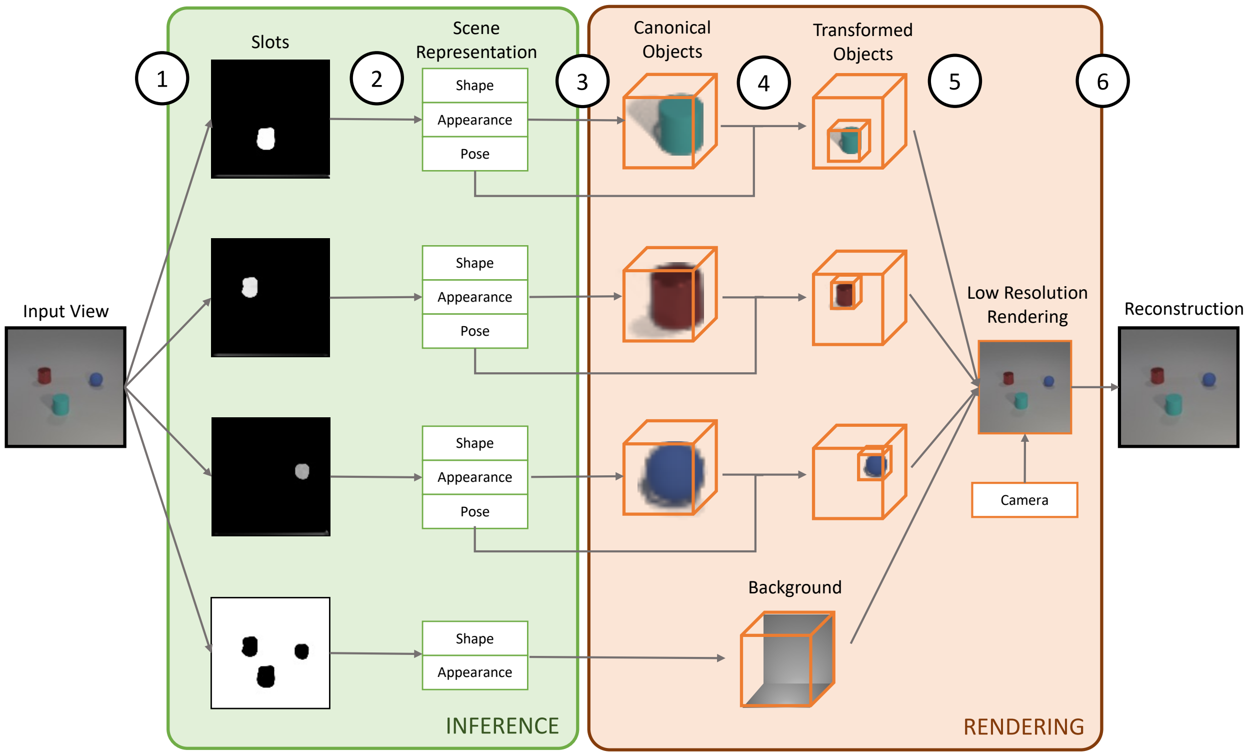


**Figure 10.1 – Model Overview:** We propose INFERNO, a model that infers and renders object-centric 3D scene representations. **1** Our model first decomposes an input observation into multiple object slots. **2** For each slot we infer a structured 3D representation. **3** The shape and appearance determine canonical objects rendered through NeRFs. **4** Objects are transformed and located in the overall scene according to their pose. **5** We combine objects and background and render a low-resolution scene given a camera location. **6** The input is reconstructed by upscaling the low resolution scene.

## 10.2. Method

We propose INFERNO (Infer NeRF Objects). The goal of our method is to infer object-centric 3D scene representations from single 2D views. Given an image $x \in \mathbb{R}^{H\times W\times 3}$, we learn an inference function $f_\theta$ that maps images to scene representations $s = f_\theta(x) = (o_1, o_2, ..., o_K, o_{bg}, c)$. Scenes are composed of $K$ objects $o_i$, a background object $o_{bg}$ and a camera location $c$.

Each object is composed of three tensors $o_i = (o_i^{shape}, o_i^{app}, o_i^{pose})$. The object shape $o_i^{shape} \in \mathbb{R}^{D_{shape}}$ and object appearance $o^{app} \in \mathbb{R}^{D_{shape}}$ are tensors that respectively describe the shape occupancy and color of an object with an implicit function. The object location $o_i^{pose} \in \mathbb{R}^{4\times 4}$ is an affine matrix that describes the object pose (i.e. scale, translation and rotation) in the scene.

The background object $o^{bg} = (bg^{shape}, bg^{app})$ only models shape and color, and its location is fixed, encompassing the *back-of-scene* cube. We also define a camera matrix $c \in \mathbb{R}^{3\times4}$, that determines the location of the scene camera and defines a 2D projection of the 3D scene.

To optimize our inference function we formulate an optimization problem in which we minimize a reconstruction loss over a dataset, similar to an auto-encoder. We define a rendering function $g_\gamma$ that takes as input a scene representation and generates a 2D view of that scene $\hat{x} = g_\gamma(f_\theta(x))$. We assume a isotropic Gaussian likelihood model with unit covariance and optimize the probability of the data under our model, which is equivalent to minimizing the mean squared error of inputs and reconstructions:

$$\begin{aligned} \gamma^*, \theta^* &= \arg\min_{\gamma,\theta} p(X|\gamma,\theta) \\ &= \arg\min_{\gamma,\theta} \frac{1}{N} \sum_{i\leq N} (x_i - g_\gamma(f_\theta(x_i)))^2 \end{aligned} \tag{10.1}$$

In the following sections we describe in more detail our inference mechanism, our rendering pipeline and their implementation.

### 10.2.1. Rendering Pipeline

We represent objects as Neural Radiance Fields (NeRFs) (Mildenhall et al., 2020) with a similar setup as that of GIRAFFE (Niemeyer & Geiger, 2021). A NeRF is a function $g_\tau$ that defines a 3D shape implicitly. It takes as input a 3D location $\mathbf{l} = (x, y, z)$ and a 2D viewing direction $\mathbf{d} = (\psi, \phi)$ and outputs an occupancy value $\sigma$ and a color value $\mathbf{a} = (r, g, b)$. NeRFs are usually implemented using fully connected neural networks. Additionally, the inputs are usually embedded into a higher-dimensional space using positional encodings $\gamma$ that embed locations and viewing directions into higher dimensional spaces $\mathbb{R}^{P_l}$ and $\mathbb{R}^{P_d}$, respectively.

$$\begin{aligned} g_\tau : \mathbb{R}^{P_l} \times \mathbb{R}^{P_d} &\to \mathbb{R}^+ \times \mathbb{R}^3; \\ (\gamma(\mathbf{l}), \gamma(\mathbf{d})) &\to (\sigma, \mathbf{a}) \end{aligned} \tag{10.2}$$

To represent multiple shapes with the same NeRF function, we can augment it with latent variables that determine which shape is being modeled (Schwarz et al., 2020). NeRFs are usually augmented with two random variables: one random variable $\mu \in \mathbb{R}^{D_{shape}}$ defines the shape of the entity being modeled, while $\upsilon \in \mathbb{R}^{D_{app}}$ models its appearance. In practice, this specialization is enforced by making the occupancy output a function of only the shape latent, while the color output is conditioned on the appearance latent.

$$\begin{aligned} g'_{\tau} : \mathbb{R}^{P_l} \times \mathbb{R}^{P_d} \times \mathbb{R}^{D_{shape}} \times \mathbb{R}^{D_{app}} \to \mathbb{R}^{+} \times \mathbb{R}^{3}; \\ (\gamma(\mathbf{l}), \gamma(\mathbf{d}), \mu, \upsilon) \to (\sigma, \mathbf{a}) \end{aligned} \tag{10.3}$$

In INFERNO, we share a single parametrization of a NeRF function across all objects. Each object specific shape and appearance is defined by the shape and appearance latent variables, which correspond to the object attributes $o^{shape}, o^{app}$. The background is defined as another NeRF with separate parameters. The background NeRF also has shape and appearance latent variables to model different backgrounds.

The pose of an object $o_i$ in the scene is determined by the affine transformation matrix $o_i^{pose}$. We denote the coordinate system of the NeRF function of an object as the object space, and the coordinate system of the scene (and the background NeRF) as scene space. Given an object pose, we can convert points from the scene space to the object space by applying the $o^{pose}$ transformation matrix on those points, and we can transform points from object space to scene space by computing the inverse of the object pose matrix.

To render a scene, we cast rays from each pixel in the 2D plane defined by a given camera to the 3D scene. We evaluate NeRFs at different points along a given ray, and integrate their occupancy and color outputs to determine pixel values. Rays might traverse multiple object NeRFs in addition to the background NeRF. To determine the occupancy and color of points described by multiple NeRFs, we first query each NeRF at those particular points. To query the object NeRFs, we first need to transform the points from scene space to the particular object space. Then, we compose the results of each NeRF with a pooling function $C$, which in our case is a weighted average:

$$C(l, \mathbf{d}) = (\sigma = \sum_{i=1}^{N} \sigma_i, \frac{1}{\sigma} \sum_{i=1}^{N} \sigma_i \mathbf{a}_i) \tag{10.4}$$

Since rendering with NeRFs as originally proposed is computationally expensive, at the beginning of training we render output views at a fixed low resolution. Low resolution scenes are then upscaled to the desired output resolution using a convolutional neural network, keeping the entire rendering pipeline differentiable. Additionally, low resolution scenes are rendered with additional channels, allowing for more detailed upscalings beyond those possible when just rendering low resolution RGB outputs. After some iterations and once the model is capable of reconstructing the input view, we remove the neural network upscaler and render with NeRFs at full resolution. This second stage of training has slower iteration times and higher memory requirements.

### 10.2.2. Inference

Given the rendering pipeline, the goal of our method is to infer representations that reconstruct a given scene. Our inference mechanism computes image features through a neural network encoder and then extracts $K$ object and a background slots. These slots are then mapped to our structured scene representation through learned neural networks.

To extract image features for each object and background slots, we use Slot Attention (Locatello et al., 2020). Slot Attention is a mechanism that maps a set of $K$ entities, called slots, to image features without annotations. It extracts image features $I \in \mathbb{R}^{H \times W \times D}$ from a given input using a resolution-preserving convolutional encoder. These features are then attended to by a set of $K$ randomly sampled slots $\pi_j \sim \mathcal{N}(\mu, \sigma)$ of dimension $D$, where $\mu$ and $\sigma$ are learnable parameters. We denote by $\pi$ the matrix concatenating all the sampled slots. Slots attend spatial chunks of the input features $I$ through soft-attention $u = TQ^T$, where $T = k(I)$ and $Q = q(\pi)$ are the embeddings of the inputs and slots respectively. The attention weights are normalized through a softmax operating on the slots axis, which makes slots compete among themselves and discourages multiple slots from attending the same input region $w = softmax(u)$. The weighted average of $I$ according to the attention weights is then computed and fed to a GRU network, to update each slot value: $\pi_j = GRU(w_j * I, \pi_j)\ \forall j \in 1, K$. Multiple rounds of soft attention are performed to iteratively refine the slots. For more details about Slot-Attention, please refer to (Locatello et al., 2020).

Object slots are unstructured tensors that result from aggregating image features. We map these slots to our structured scene representation through small fully-connected networks that operate on individual objects. More concretely, we map object slots to their 3D pose in the scene through a 2-layer MLP. To infer the object shape and appearance tensors we also use 2-layer MLPs, but we make them conditional on the predicted object pose through conditional normalization (Dumoulin et al., 2018).

## 10.3. Related Work

### 10.3.1. 3D Shape Representations

There are different ways to represent 3D geometry such as voxels or meshes (Rematas & Ferrari, 2020; Gkioxari et al., 2019). For example, the GAN models of Nguyen-Phuoc et al. (2019, 2020) successfully use voxel-based representations to render images. Voxel-based methods have trouble scaling up to high resolutions as the size of a voxel representation scales cubically with the resolution.

Recently, the use of functions that implicitly model 3D volumes has gained popularity (Park et al., 2019; Mescheder et al., 2019; Sitzmann et al., 2019, 2020b,a; Kosiorek et al., 2021; Pumarola et al., 2021; Yu et al., 2021a). Implicit representations have better scaling properties, as usually the output resolution does not directly affect the dimensionality of the learned function. NeRFs (Mildenhall et al., 2020) generate scenes by learning a function that outputs the occupancy and color of points in a scene when viewed from a particular direction. By casting rays through a plane and aggregating the output values NeRFs can generate 2D views of an implicitly modeled 3D scenes. NeRFs have obtained superior reconstructions compared to other implict methods, and our model uses NeRFs to represent multiple objects and the background of a scene.

Most methods using NeRFs represent scenes monolithically as a single entity. GIRAFFE (Niemeyer & Geiger, 2021) is a GAN-based method that represents multiple objects in a scene using NeRFs as part of their generator. Their factored representations are amenable to object manipulations. Our model uses a rendering pipeline inspired by GIRAFFE. However, we focus on recovering scene representations from existing images, while GIRAFFE does not have an inference mechanism. ObjSURF (Stelzner et al., 2021) and UORF (Yu et al., 2021b) infer scene representations composed of multiple objects, each object represented with a differently instantiated NeRF. Different from our work, they focus on novel view generation. These methods do not explicitly infer the pose of the different objects in the scene. Both methods require multiple scene views and their associated ground-truth camera locations, and ObjSURF additionally requires depth annotations.

### 10.3.2. Object-Centric Scene Models

Segmenting objects in a scene is a landmark computer vision task with an extensive literature. Recently, there is a line of work on object-centric generative models of scenes (Kosiorek et al., 2021; Burgess et al., 2019a; Locatello et al., 2020; Lin et al., 2020; Greff et al., 2019a). These models learn to generate scenes as a composition of multiple objects and a background. When equipped with an inference mechanism, these models learn to segment objects in a scene without annotations, driven by their compositional generative process. MONet (Burgess et al., 2019a) implements a multi-object VAE that segments object sequentially by infering latents corresponding over parts of the scene not yet attended to iteratively. IODINE (Greff et al., 2019a) uses a similar multi-object VAE and performs multiple rounds of inference to settle on a scene decomposition. Slot-Attention (Locatello et al., 2020) maps a set of entities, called slots, to image features through multiple rounds of soft attention. The slots compete among themselves to attend to features, making each slot attend to a region of the input image. When driven with alpha-compositing decoder, slot attention learns to segment objects in a scene. Our model uses a variant of Slot Attention to decide on which part of a 2D view should each object in our scene attend to, but we infer 3D-aware representations for each object. Object-centric scene models have also been implemented as world models, with the goal of simulating dynamics (Lin et al., 2020). By decomposing the scene into objects, these methods can simulate dynamics at an object level, which are usually simpler and shared among objects. Contrary to most previous approaches which segment 2D shapes, our method infers object-centric scene representations in 3D space. ROOTS (Chen et al., 2021) extends G-SWM (Lin et al., 2020) and GQN (Eslami et al., 2018) to learn object-centric 3D scene representations from multiple observations that can be rendered from arbitrary viewpoints. INFERNO focuses instead on using a single scene view to infer object-centric scene representations and uses differentiable rendering to generate 2D views of scene representations.

## 10.4. Experiments

In this section we showcase the capabilities of INFERNO with three main experiments. First, we demonstrate the interpretability of the scene representations through manipulating scenes and verifying the corresponding effects in the rendered outputs. Then we show that it learns to identify and segment the objects in a scene without supervision. Finally, we highlight the usefulness of such representations for downstream tasks by applying our model on the CATER snitch localization task.

### 10.4.1. Training Setup

We use the same training setup for all experiments unless otherwise mentioned. We train our models for 400K iterations using the Adam optimizer Kingma & Ba (2014) with a learning rate of $1 \times 10^{-4}$. We use a batch size of 128 and we use up to 16 nVidia V100 GPUs. We use learning rate warmup (Goyal et al., 2017), which is helpful to avoid optimization issues with Slot Attention. We use a weight decay rate $\lambda = 1 \times 10^{-6}$. We use three iterations of slot attention during training and evaluation. We remove the neural upscaler and render at full resolution after 100K iterations. We set up the number of objects in a scene as the maximum possible number of objects in a dataset, i.e. five objects for CLEVR2345, 6 for CLEVR6 and 10 for CATER. Refer to Appendix 10.B for more details on the experimental setup.

### 10.4.2. Scene Inference

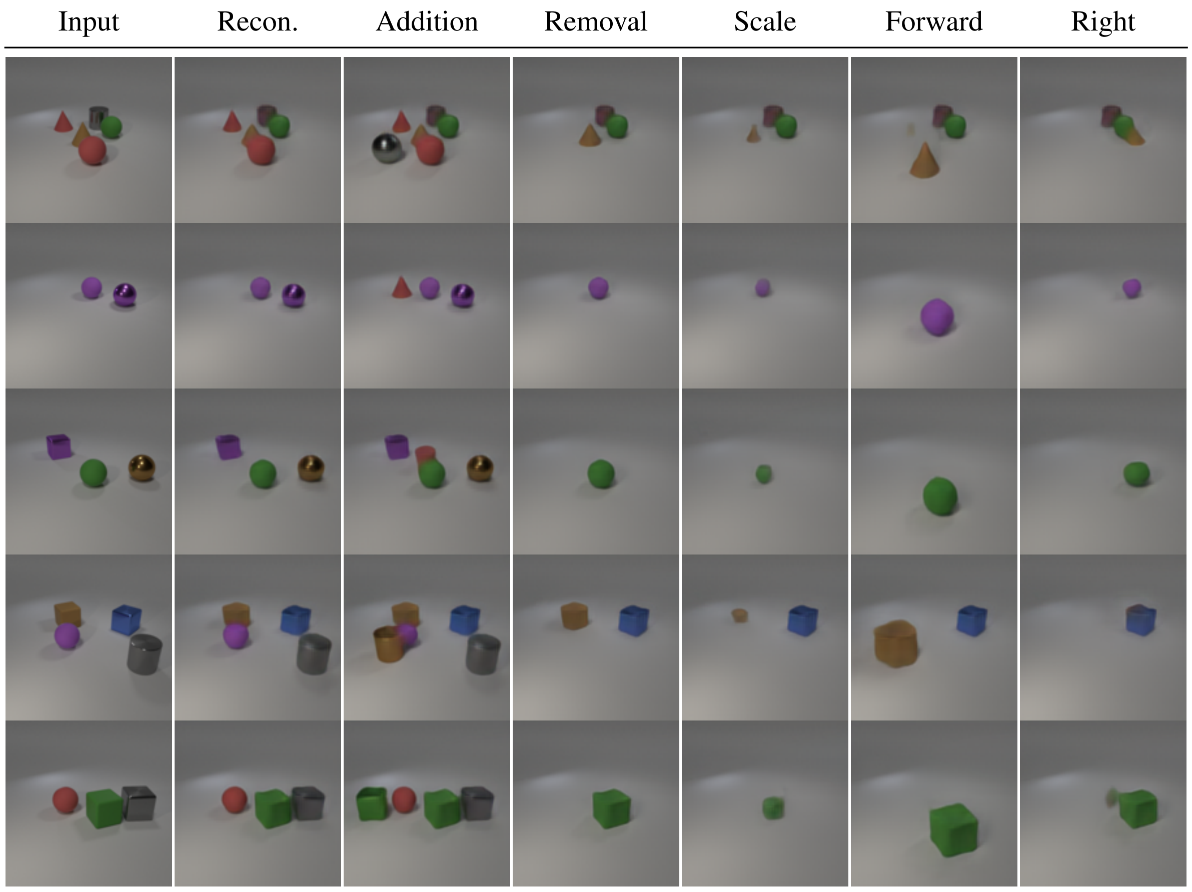


**Figure 10.2 – Manipulations on CLEVR2345**: we show some examples of the manipulations we perform to CLEVR2345 images, including object removal and addition, changing the scale of an object, and object translation. Our model can perform these transformations because it disentangles object pose and appearance.

In this section we demonstrate the properties of our scene representation. Our model infers 3D object-centric scene representations from single 2D views. These representations disentangle the appearance and pose of objects, which allows for semantic manipulations of the scene not possible otherwise. These manipulations can be validated by rendering the modified scene

**Table 10.1 – Reconstruction error on CLEVR2345** We consider an autoencoder baseline with a single NeRF object capturing the whole scene (NeRF-AE), and compare it to our model on the test set of CLEVR-2345. NeRF-AE struggles to reconstruct multiple objects accurately. In contrast, INFERNO produces better reconstructions under all metrics and allows for object identity/pose manipulations.

| Model | MSE (↓) | PSNR (↑) | SSIM (↑) | LPIPS (↓) |
|---|---|---|---|---|
| NeRF-AE | $5.14 \times 10^{-4}$ | 44.89 | 59.24 % | $168.9 \times 10^{-3}$ |
| Ours | $1.22 \times 10^{-4}$ | 52.07 | 72.4 % | $18.93 \times 10^{-3}$ |

**Table 10.2 – FID on CLEVR2345** We consider our model as a NeRF scene generator and compare it to the state-of-the-art. When reconstructing ground-truth images, our model obtains better FID than GIRAFFE. We then consider object manipulations to generate novel scenes from existing ones. While adding new objects to a scene slightly increases our FID score, when exchanging object identities across scenes we set a new state-of-the-art for image generation on CLEVR2345.

| Model | FID (↓) |
|---|---|
| GIRAFFE | 37.7 |
| Ours - Reconstruction | 23.5 |
| Ours - Remove Object | 42.4 |
| Ours - Add Object | 27.2 |
| Ours - Swap Object | **23.7** |

representations. Additionally, we verify that decomposing the scene into multiple objects leads to better reconstructions.

First, we verify the quality of INFERNO's generations by comparing them two baselines: i) a version of our model that does not consider multiple objects, and ii) the GAN method of GIRAFFE (Niemeyer & Geiger, 2021). We perform this comparison on the CLEVR-2345 dataset introduced by GIRAFFE, which contains CLEVR images with 2 to 5 objects. For reconstruction, we compare models using reconstruction metrics including mean-squared error, PSNR and SSIM. We rely on the population metric Frechet Inception Distance (FID) to evaluate the generation quality.

In INFERNO, we generate novel scenes by inferring representations for ground-truth images and then manipulating them. To compare to GIRAFFE, we manipulate scenes by adding additional objects and swapping object shapes and appearances across scenes. Manipulations are described in more detail in the Appendix. We highlight that GIRAFFE is an unconditional model, while INFERNO generates novel scenes conditioned on existing ones.

We also investigate different interpretable manipulations of scene representations and visualize the effects in the corresponding output renderings. We also conduct this experiment on the

CLEVR-2345 dataset. We validate that our model is able to render out-of-distribution scenes not corresponding to training examples, such as scenes having 1 or 6 objects, and verify that the pose manipulations have semantically coherent effects.

Table 10.1 shows the reconstruction metrics obtained by our model and baseline on the CLEVR2345 dataset. Note that GIRAFFE is a GAN-method that does not have an inference mechanism, and therefore it cannot reconstruct scenes. We observe that the NeRF-AE baseline obtains higher reconstruction error than our regular model, as our object-centric method can make better use of its capacity. In Table 10.2 we compare INFERNO with GIRAFFE using the FID metric. Our model reconstructions have better FID than the generations of the GIRAFFE. Additionally, our model can perform inference and manipulations on existing scenes. We use that capability to generate novel scenes by manipulating existing ones. Our model is able to generate novel scenes with additional objects or with altered object shapes and appearances, with better FID than GIRAFFE.

In Figure 10.2 we show some examples of the scene manipulations possible with our model. Given a scene representation, we can remove or add objects, rearrange object poses, translate the objects to new locations or change the object scales. While some of these manipulations can be performed with regular object-centric models, modifications to the scale and location of the objects are hard to implement without explicitly modeling 3D object pose.

INFERNO is able to synthesize novel views of a scene despite being trained with single scene views and without ground-truth camera locations. We refer the reader to the Appendix 10.D for additional details.

### 10.4.3. Object Discovery

**Table 10.3 – Object Discovery Metrics on CLEVR6** INFERNO, despite inferring more complex 3D object segmentations without annotations, is competitive with the current state-of-the-art 2D object discovery methods on CLEVR6.

| Model | ARI % (↑) |
|---|---|
| Slot-Attention | $98.8 \pm 0.3$ |
| IODINE | $98.8 \pm 0.0$ |
| MONet | $96.2 \pm 0.6$ |
| Slot MLP | $60.4 \pm 6.6$ |
| Ours | $96.7 \pm 0.2$ |

Unsupervised object discovery consists in learning to segment the objects in a scene without using annotations. We test our model on CLEVR6 benchmark, a variant of the CLEVR

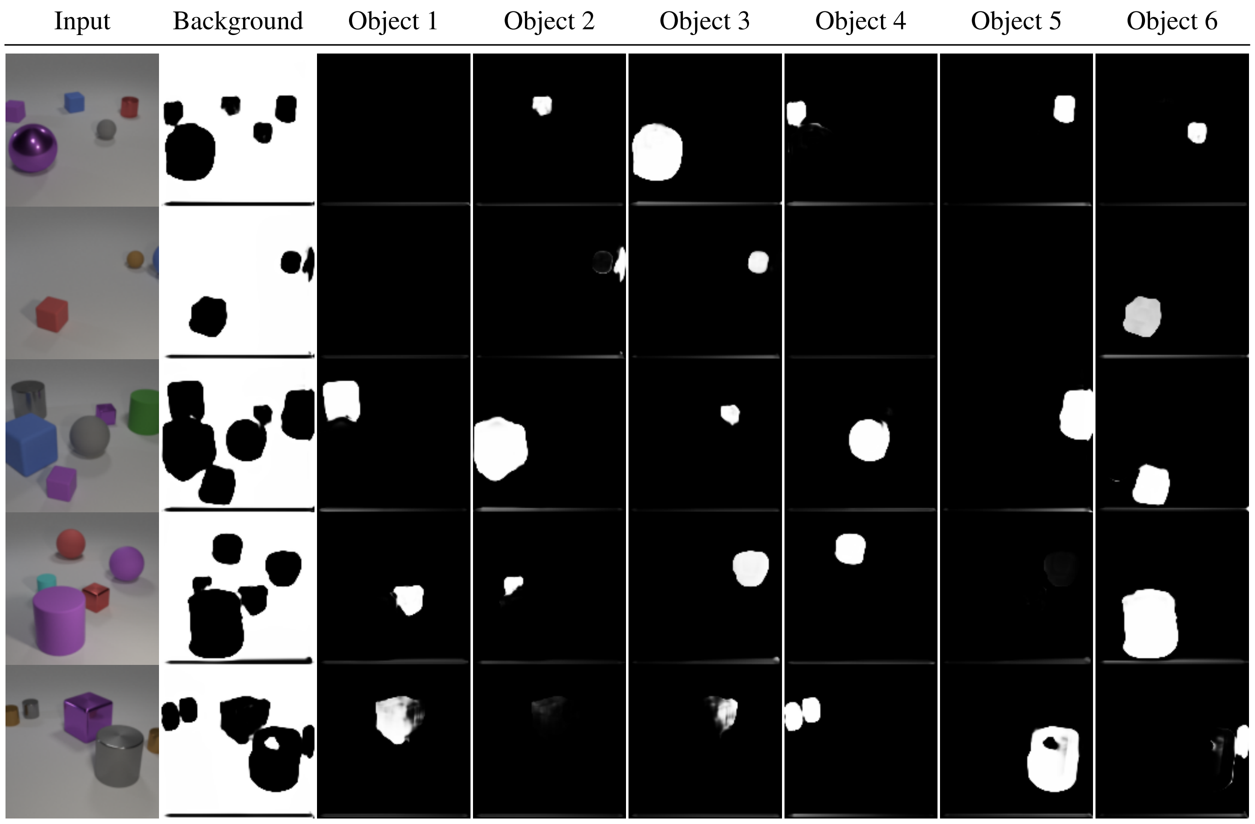


**Figure 10.3 – Object Discovery on CLEVR6**: INFERNO identifies the different objects in a scene without supervision. For each input image, we show which regions of the input are attended by each object as well as the background. We include an example of a failed segmentation in the last row, where one object slot (4) is trying to represent multiple objects at the same time.

dataset with scenes of up to 6 objects and annotated with 2D object masks. We choose this dataset to compare to previous work in unsupervised object discovery. Note that this setup evaluates 2D segmentation masks, although our model naturally provides 3D segmentations. We evaluate the quality of the segmentations using the Adjusted Random Index (ARI) metric (Rand, 1971), which is a measure of clustering similarity. In line with previous work, we compute only the foreground ARI, which does not take into account the background segmentation mask. In particular, we consider object as a different clusters and compare the cluster assignment of each foreground pixel in the original image to its prediction. To determine which pixels correspond to each object in our model we make use of the input segmentation masks predicted by our inference mechanism.

Table 10.3 reports the ARI metric of our model and different baselines in this task. We observe that INFERNO is competitive with state-of-the-art methods, surpassing MONet and having slightly lower ARI than Slot-Attention and IODINE. However, our model learns to segment objects in 3D, while the other baselines extract 2D segmentation masks. Figure 10.3 shows some examples of the discovered object masks. We can see that the model learns to segment different objects and properly discards object slots when the number of objects in the scene is lower than needed. We also include an example that our model fails to segment properly - multiple objects are segmented by the same object slot, and a single object is represented in multiple parts by different slots.

### 10.4.4. Snitch Localization

**Table 10.4 – Snitch Localization on CATER**. We report Top-1 and Top-5 accuracies for the snitch localization task. Our model outperforms the R3D LSTM and R3D NL LSTM models that learn unstructured representation. It indicates that the structured representation learned by INFERNO is useful for this task. INFERNO pretraining is also critical, showing that the pretraining and not the encoder architecture is a key component. Overall, INFERNO achieves performances close to the state-of-art approaches.

| | Model | Top-1 | Top-5 |
|---|---|---|---|
| QA Methods | R3D LSTM | 60.2 | 81.8 |
| | R3D + NL LSTM | 46.2 | 69.9 |
| | Hopper | 73.2 | 93.8 |
| | Aloe (w/out SSL loss) | 60.1 | - |
| | Aloe | **74.0** | **94.0** |
| Fine-tune | Slot-Attention | 59.1 | 88.0 |
| | Ours (w/out pretraining) | 2.91 | 12.9 |
| | Ours (w/out SSL loss) | 69.17 | 87.68 |
| | Ours | **71.7** | **88.9** |

The goal of this experiment is to show that the representation learned by our model is useful for the snitch localization task from the CATER dataset (Girdhar & Ramanan, 2019). We focus on the CATER task involving videos with a static camera. The objective is to predict the final position of an object (the snitch) in a video. The scenes show multiple objects that move over time, one being the snitch object. The snitch can be occluded and moved around simultaneously by other objects, requiring object tracking and reasoning about dynamics to solve the task.

We follow the experimental setup of Ding et al. (2020). First, we train INFERNO to reconstruct images from the CATER dataset. Once INFERNO is trained, we discard its rendering pipeline, and instead feed the scene representations to a 12-layer transformer to predict the final snitch position. Each object in our representation is given as an input element to the transformer. We add a learned positional encoding to the object representations based on their frame index. Objects in the same frame have the same positional encoding. The last output of the transformer is fed to a MLP head that predicts the logits for the 36 possible output positions.

We minimize the sum of the cross-entropy and a L1 loss between the predicted and the true snitch final position. Following (Ding et al., 2020), we optionally use anauxiliary SSL loss after pretraining. The SSL loss randomly masks one object per-frame and tries to predict its representation at the corresponding object output, through minimizing the L2 distance between the predicted object representation and the observed but masked one. The SSL loss is only backpropagated through the transformer and not the inference network.

During training, we randomly sample 40 frames from a video and predict the snitch location from these frames. At test time, we randomly sample 10 temporal crops of 40 frames each and average the model predictions to compute the final probabilities. Refer to Appendix 10.B for more details about the experimental setup.

Table 10.4 reports CATER Top-1 and Top-5 accuracies for different methods. We first compare our model pretrained to reconstruct images on CATER with a randomly initialized encoder. Using a randomly initialized encoder does not perform well, suggesting that the pretraining rather than the encoder architecture is key to learn useful representations for the task. We also observe that the additional SSL loss slightly improves the performance of our model.

We next compare our approach to Aloe (Ding et al., 2020), an object-centric baseline, R3D and R3D NL, 3D convolutional models proposed by (Girdhar & Ramanan, 2019), and Hopper (Zhou et al., 2021), that use a strong inductive bias towards object tracking. Our model outperforms the R3D and R3D NL models that rely on unstructured representation. This result suggests that the object-centric representation learned by INFERNO is useful for the visual reasoning task. Our model also outperforms Aloe, an object-centric method using 2D object representation, when both methods do not use additional SSL loss. However Aloe benefits more from the use of an additional SSL loss. Overall, INFERNO achieves performances close to the state-of-art approaches.

We finally evaluate the performance of a slot-attention baseline (Locatello et al., 2020) in Table 10.4. The slot-attention baseline first pretrains a slot-attention encoder, with a similar architecture than our model, by reconstructing CATER frames using a mask decoder. It then fine-tunes the encoder using the same procedure than our model to solve the snitch localization task. We observe that our model significantly outperforms the slot-attention model which focus on the 2D geometry of the scene. This result supports the advantage of 3D-aware representation for solving the CATER task.

### 10.4.5. Limitations

Based on the conducted experiments we highlight two main limitations of our model.

First, INFERNO has difficulty in inferring the hidden sides of objects. That is due to training our model without ground-truth camera locations and from a single view, as opposed to models that reconstruct scenes across multiple views using their corresponding ground-truth camera locations.

The other main limitation is in scaling up our model to more complex datasets. This is usually due to a wrong slot decomposition, and it is a common failure mode of Slot Attention whether its driven by a 2D mask decoder or by our rendering mechanism.

## 10.5. Conclusions

We propose INFERNO, a model for inferring object-centric 3D scene representations. Our model is able to discover objects in a scene without annotations, and the inferred scene representations are interpretable and amenable to manipulations. Further, the scene representation is useful for visual reasoning downstream tasks such as the snitch localization task in CATER.

**Acknowledgements.** We thank the Mila Quebec AI Institute for managing the computer clusters on which this research was conducted. This work was supported by an IVADO PhD Fellowship to L.C. and by funding from CIFAR.

## 10.A. Appendix - Additional Model Details

Our model is composed of five main modules: encoder, slot attention, slot to object mapping, NeRF decoder, neural network upscaler. In this section we provide additional details about each of these modules as well as describe their architecture.

**Table 10.5 – Encoder Neural Network**

| Layer Type | Size | Normalization | Activation | Other details |
|---|---|---|---|---|
| Conv $5 \times 5$ | 64 | - | ReLU | Stride 1 Pad. 1 |
| Conv $5 \times 5$ | 64 | - | ReLU | Stride 1 Pad. 1 |
| Conv $5 \times 5$ | 64 | - | ReLU | Stride 1 Pad. 1 |
| Conv $5 \times 5$ | 64 | - | ReLU | Stride 1 Pad. 1 |

**Table 10.6 – Slot Attention Neural Network**

| Name | Size | Description |
|---|---|---|
| Positional emb. | 64 | Additive embedding, same size as CNN input features |
| Flatten | - | Flattens the spatial dimensions of CNN features |
| QKV MLP | 128 | Linear layers that map slots and input features to the same dimension |
| LayerNorm | 128 | Normalizes the slots/inputs |
| MLP + GRU | 128 | The output of soft-attention goes through a linear layer + GRU |

**Table 10.7 – Slot to Object MLP details**

| MLP Name | Size | Act and Norm. | Description |
|---|---|---|---|
| Obj Pose | 7 | ReLU, LayerNorm | Slot to translation, scale and rotation |
| Obj Shape/App. | 128 | ReLU, CondLayerNorm | Slot to shape and appearance |
| BG Shape/App. | 128 | ReLU, LayerNorm | Background slot to its shape/app, fixed pose. |

**Table 10.8 – Details about the NeRF MLPs used**

| MLP Name | Layers | Size | Description |
|---|---|---|---|
| Obj MLP | 8 | 64 | ReLU activation, no norm. Skip connection with layer 4. |
| BG MLP | 4 | 16 | ReLU activation , no norm. |

**Table 10.9 – Neural Upscaler architecture**

| Layer | Size | Activation | Normalization | Other |
|---|---|---|---|---|
| Conv $3 \times 3$ | 64 | ReLU | Instance | Stride=1 Pad=1 |
| Upsample | - | - | - | Nearest Neighbors |
| Conv $3 \times 3$ | 64 | ReLU | Instance | Stride=1 Pad=1 |
| Upsample | - | - | - | Nearest Neighbors |
| (Only 128px) Conv $3 \times 3$ | 64 | ReLU | Instance | Stride=1 Pad=1 |
| (Only 128px) Upsample | - | - | - | Nearest Neighbors |
| Conv $3 \times 3$ | 3 | - | - | Stride=1 Pad=1 |

**Encoder.** The goal of the encoder is to extract image features. We use an encoder with no downsampling, as it is typically used with Slot Attention. Details about the encoder architecture are described in Table 10.5.

**Slot Attention.** We employ Slot Attention to map image features to object slots. Slots are sampled randomly from a Gaussian distribution with learned parameters. We use different distributions for the background slot and the object slots. During training we employ three iterations of slot attention to refine the image features to slot assignments.

**Slot to Object Net.** The background and object slots are mapped to scene parameters using a series of MLP. For each object slot, we first map the object to its pose parameters. We use a 2-layer MLP with LayerNorm to map a slot to its pose. The size of the hidden dimension is the same than the output dimension size. We parametrize object pose as a 7-dimensional tensor. We use three dimensions for the object location along each axis, three dimensions for the scale of the object and a single dimension to express a rotation of the object along the X axis. These parameters are then mapped to their corresponding $4 \times 4$ affine transformation matrix for each object. Note that in practice we are not modeling rotations in the experimental section.

Once we have inferred the object poses, we infer object shapes and appearances. These are inferred individually for each object using a common 2-layer MLP. The MLPs are conditional on the object pose using Conditional Layer Normalization, that makes the learned parameters of LayerNorm be a function of a condition. Specifically, we map the 7-dimensional pose tensor to LayerNorm parameters with a single linear layer with no activation or normalization. Shapes and appearances are defined by the output of the MLPs, which produce two 128-dimensional tensors.

For the background object we only infer shape and appearance, and define its pose to be that of the scene cube. The shape and appearance of the background are inferred through another 2-layer MLP with ReLU and LayerNorm. A summary of the models discussed in this section can be found in Table 10.7.

Note that our scene representation also admits a camera pose. In our experiments we fix the camera location to look at the scene from a standard location (centered and 33 degrees above the Z plane). Other concurrent approaches (Yu et al., 2021b; Stelzner et al., 2021) use ground-truth camera locations to generate novel views, while we focus on recovering scene representations without the use of ground-truth annotations.

**NeRF MLPs.** With the object shape and appearance tensors we can render them following GRAF (Schwarz et al., 2020). We use one NeRF for the objects and one NeRF MLP for the background. The details about each NeRF architecture can be found in Table 10.8. Note that, to render objects according to their pose, we query their NeRF MLP in a canonical object space by transforming input coordinates in scene space to object space using the object pose. To reduce the computational complexity of rendering with many NeRFs, we render scenes at a fixed resolution of 16x16. Instead of rendering RGB pixels, we render feature images with 128 channels. The output of the rendering is then upscaled and mapped to RGB views with a neural network upscaler.

**Neural Upscaler.** The neural upscaler takes the low resolution output of the NeRF MLPs and upscales it to the full output resolution. Additionally, it maps the rendered image to RGB space. This module is implemented using a convolutional neural network. We always render the NeRF output at 16px when using a neural network upscaler. Consequently, we add additional layers to the neural upscaler depending on the desired output resolution. For most experiments we use a resolution of 64px, while for the Scene Inference experiments on CLEVR2345 we use a resolution of 128px. The architecture of the neural upscaler can be found in Table 10.9. After some training iterations and once the model correctly reconstructs its inputs, we remove the neural upscaler and render scenes using NeRFs at full resolution.

## 10.B. Experiment Details

### 10.B.1. Scene Manipulation

**Dataset.** For generating manipulated scenes we consider the CLEVR2345 dataset (Niemeyer & Geiger, 2021). Images in the CLEVR2345 dataset contain from 2 to 5 objects. We use the original train and test splits. Images are resized to 128x128 pixels and RGB values are normalized in the [0, 1] range.

**Training.** We use a batch size of 128 and train our model for 400k iterations. We use Adam with a learning rate of $1 \times 10^{-4}$ and weight decay $1 \times 10^{-6}$. We use 5 objects and rely on the model to not use additional slots if the scene shows less than 5 objects. We use the neural upscaler with additional layers to upscale to 128px. We remove the neural upscaler and render at full resolution after 100K iterations. Additionally, for this experiment we use an additional LPIPS loss. We use the LPIPS metric computed by an AlexNet network, and we add this loss to our regular MSE loss. We weight the LPIPS loss by a factor of 100, so that it has a comparable order of magnitude to the MSE loss.

**Manipulations.** Manipulations are done as follows:

— *Substraction:* We randomly delete up to two of the object slots.

— *Addition:* We randomly add an object slot from another scene.

— *Scale:* We reduce the scale (in all XYZ axis) of one of the objects in the substraction scene.

— *Forward:* We manipulate the pose vector of one object in the substraction scene and move it forward on the Z axis.

— *Right:* We manipulate the pose vector of one object in the substraction scene and move it forward on the X axis.

Additionally, we consider the *Swap* transformation for Table 10.1. This transformations modifies a scene by replacing the object shape and appearance vectors with those of an object from another scene.

**Metrics.** To compare reconstructions we use Mean-Squared Error, Peak Signal-to-Noise Ratio (PSNR), Structural Similarity (SSIM) and the LPIPS metrics.

MSE measure the average squared difference between pixel values.

$$\mathrm{MSE}(x, x') = \frac{1}{N} \sum_{N} (x - x')^2 \tag{10.5}$$

PSNR is a metric commonly used in signal processing.

$$\mathrm{PSNR}(x, x') = -10 \log_{10}(\mathrm{MSE}(x, x')) \tag{10.6}$$

SSIM (Wang et al., 2004a) provides scores more aligned with human perception, specially under the presence of image noise. Scores are computed convolutionally by applying a kernel over images, which are then contrasted.

LPIPS (Zhang et al., 2018b) computes differences in neural network activations for two images. It is a perceptual metric that has been shown to have higher correlation to human perception than other metrics not based on neural networks.

To compare populations of generated images we use the Frechet Inception Distance (Heusel et al., 2017). The Frechet Inception Distance embeds images into a neural network space and then fits a Gaussian distribution to the generated and ground-truth activation statistics. The score is obtained by then computing the Frechet distance between the two. Note that other metrics such as Inception Score are not applicable for the CLEVR2345 since there are no well-defined classes.

### 10.B.2. Object Discovery

**Dataset.** For object discovery we consider the CLEVR6 dataset. We use the original CLEVR6 dataset and extract the images from TFRecord files available at this URL. We use the original training/test split, using the first 70% images for training and the remaining ones for test. We take a crop between pixels [29, 221] and [64, 256], for the height and width respectively, and then resize the crop to 64px. We normalize the value of the images between

[0, 1]. To generate the CLEVR6 dataset, we keep only those images that have at maximum 6 objects according to the annotation files.

**Training.** We use a batch size of 128 and train our model for 400k iterations. We use Adam with a learning rate of $1 \times 10^{-4}$ and weight decay $1 \times 10^{-6}$. We use 6 objects and rely on the model to not use additional slots when needed.

**Metrics.** We follow previous work (Greff et al., 2019a) and use the Adjusted Rand Index (ARI) (Rand, 1971) to evaluate cluster assignments in object discovery. ARI scores range from 0 (random assigment) to 1 (perfect match). As in previous works, we do not consider a segmentation mask for the background.

### 10.B.3. Visual Reasoning on CATER

For the visual reasoning task, we consider the CATER dataset and uses 5K videos that do not have camera motion. All the videos are resized to a 64x64 resolution.

**Pretraining.** We first pretrain our INFERNO to reconstruct individual frames from the CATER dataset. We bootstrap a model trained on CLEVR6 for object discovery for 400k iterations and train it on the CATER dataset for an additional 100k iterations to speed-up training, as the iteration time of a model with 10 objects is larger. We use a batch size of 128 and we use Adam with a learning rate of $1 \times 10^{-4}$ and weight decay $1 \times 10^{-6}$. When training on CLEVR6 we use 6 object slots, while when training on CATER we use 10 object slots.

**Finetuning.** After pretraining, we finetune the INFERNO encoder to the supervised task of snitch localization. We discard the rendering pipeline of our model, and instead feed the inferred object slots representations to a transformer that aims at predicting the final position of the snitch.

To predict the snitch, we consider a 12 layers transformer with the hidden dimension of 128 which takes the slot representation as input. The transformer treats each object as input element. A learned positional embedding is added each slots representation based on their frames index, i.e. the position of the objects is the same within a frame. The final output to the transformer is given to a 1 layer MLP head with an hidden dimension of 128. It outputs 36 logits that correspond to possible snitch location. We minimize the sum of the cross-entropy between the predicted position and the true target and a the l1 loss between the prediction and target.

We optionally use a SSL loss similar to Ding et al. (2020). The SSL loss randomly masks one object per-frame and tries to predict its representation at the corresponding output. The model minimizes the L2 distance between the predicted object representation and the observed one. The SSL loss is only backpropagated through the transformer and not the encoder. We weight the SSL loss by a factor of $1.0e-3$.

We use an Adam optimizer to minimize the loss. The initial learning rate is set to the $1.0e-4$ and gradually decreased to $1.0e-6$ using a cosine learning rate decay. Similarly, we use a initial weight decay of $1.0e-5$ that we increases to $1.0e-3$ using a cosine schedule. Our model is finetuned for 500 epochs. We don't make use of learning rate decay.

During training, we randomly samples 40 frames from a video and predict the snitch localization from this one crop. At test time, we randomly sample 10 temporal crop of 40 frames each and average the prediction over the 10 crops as our final prediction.

## 10.C. Additional Visualizations

We have included additional manipulations of one scene in GIF format in the supplementary material. We show: i) each object rendered individually, ii) object identity swaps with other scenes, iii) object translations along one axis, iv) translations in diagonal (two axis), and v) objects moving in a circle.

We also include more examples of reconstructions and scene manipulations where we add and remove objects in Figure 10.4 and Figure 10.5, respectively.

## 10.D. Novel View Synthesis

In this section we synthesize novel views of a scene by modifying the camera pose. For each example we show the input image, our reconstruction, then two images as a result of moving the camera $\pm 15°$ in the azimuth axis, and two images as a result of zooming in the scene. This experiment is shown in Figure 10.6. While our model is trained without ground-truth camera poses and with single views of scenes, it is able to generalize to small camera pose modifications and render novel views of a scene.

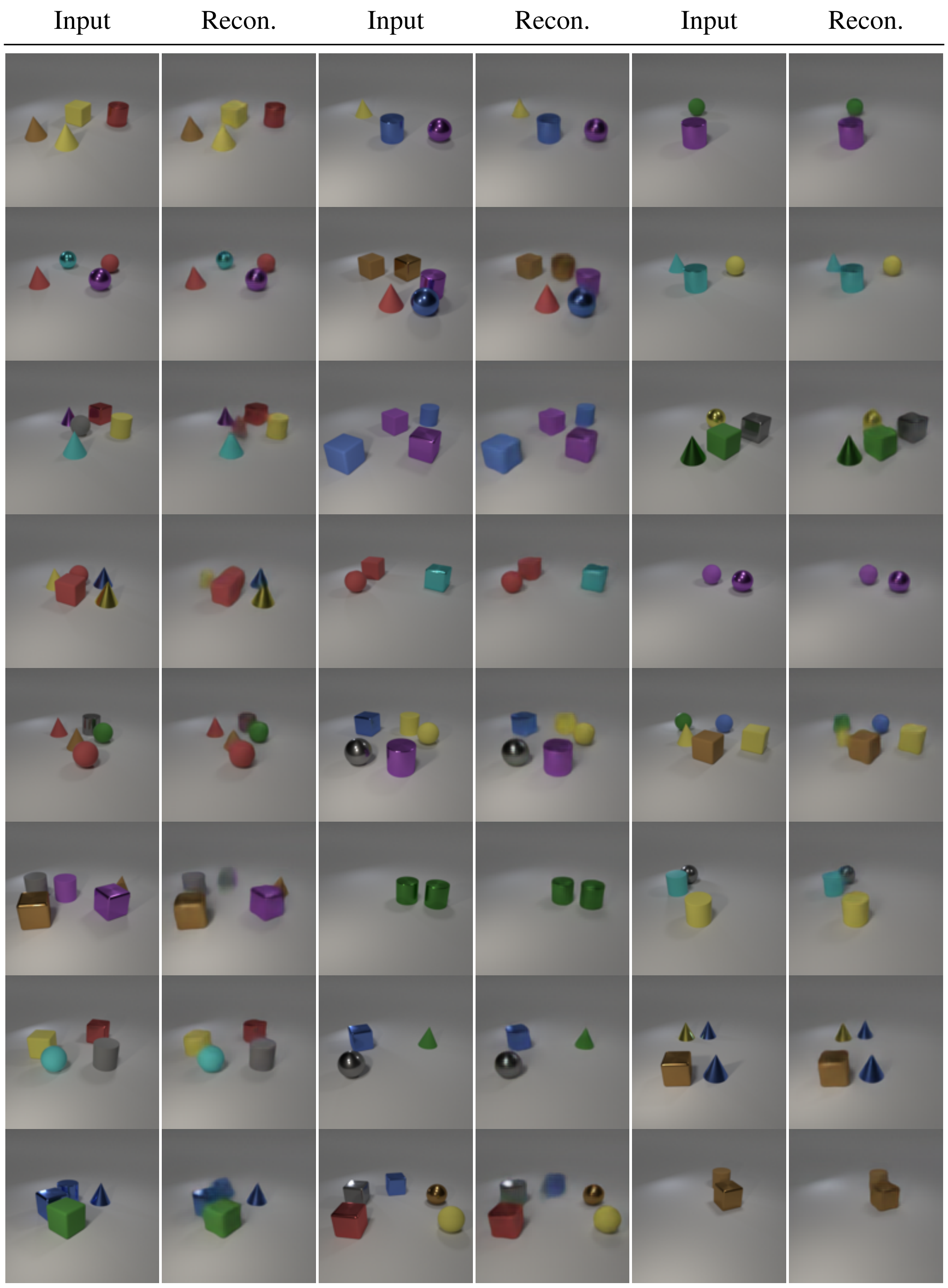


**Figure 10.4 – Additional reconstructions on CLEVR2345.**

## 10.E. NeRF output with Neural Upscaler

In this section we show the raw outputs of the NeRF function. These outputs show scene views rendered at low resolution, which are then upscaled with a neural network. While

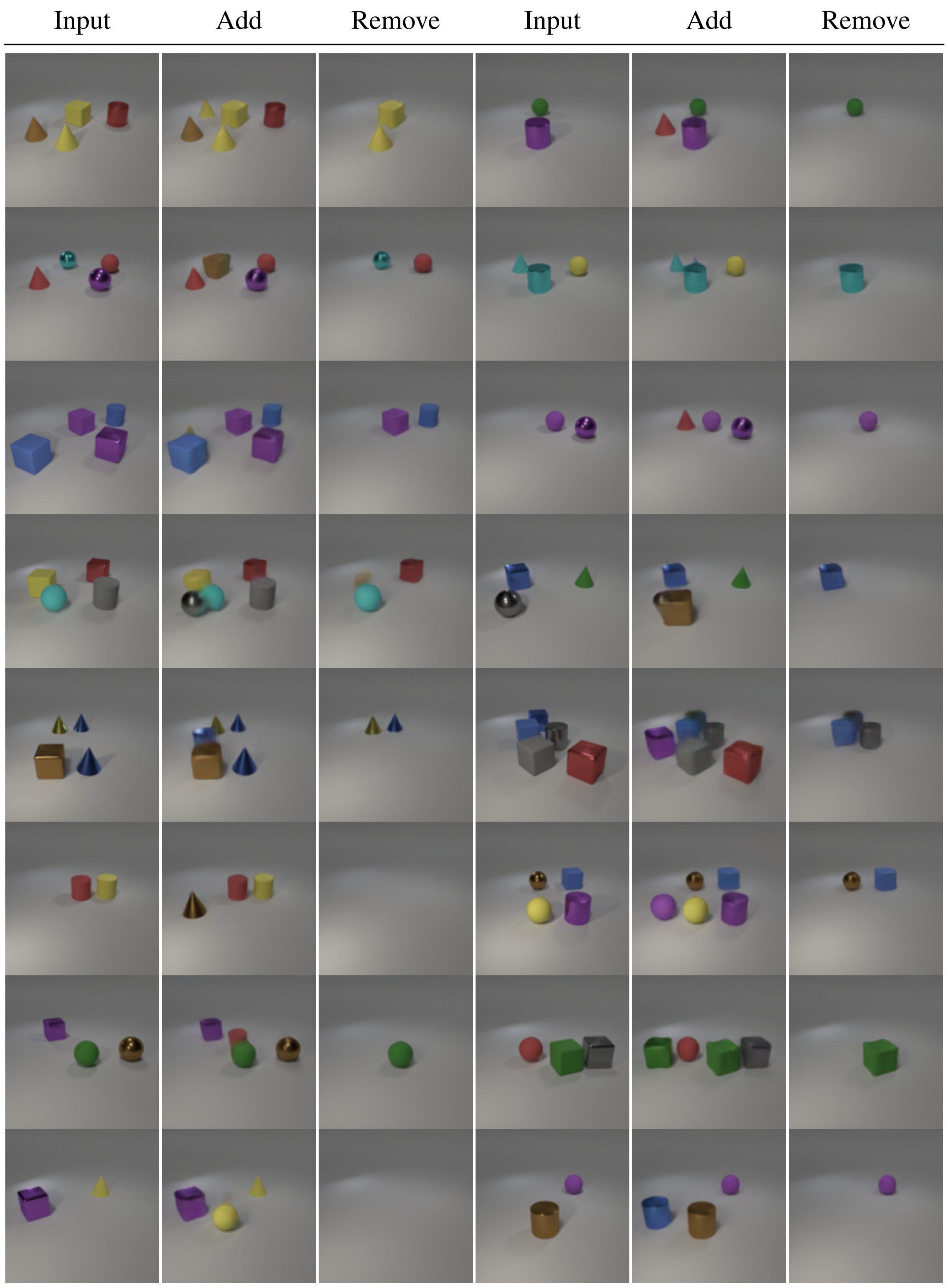


**Figure 10.5 – Additional object additions and removals on CLEVR2345.** For each scene, we show images with one randomly added object, and with 1-3 random objects removed. Some of these images show out-of-distribution samples with a number of objects not seen during training (2-5 objects).

these generations have reduced details due to their resolution, they clearly show the different objects and their location in the scene. We show these visualizations in Figure 10.7. NeRFs are rendered at low resolution to ease the computational costs, as the time and memory requirements of rendering with a NeRF are linearly correlated with the number of casted rays/pixels in the output image. After we bootstrap the model with the neural upscaler, we then remove it and render with NeRFs at full resolution.

## 10.F. Slot visualization

In this section we show the rendering of individual slots. Given an input scene, we first infer its representation. Then, for each object in the representation, we render it individually against a black background. Examples from this visualization are shown in Figure 10.8. We observe that, in general, each object slot is rendering a part of the scene corresponding to a single object instance. Additionally, for some objects their slot captures parts of its shading.

## 10.G. Ethics Discussion

INFERNO can be used to generate views of novel scenes. As with other generative models, there is a chance that similar methods as the one we propose might be used to create "deepfakes". Note however that it would require extensive follow-up work and that it would not be a direct application of our method.

At the same time, generative models have multiple positive applications. For example, novel generations can be used to train better image classifiers, and image generators can also be used to facilitate content creation for visual artists.

Overall our model would require follow-up work to enable some of the positive and negative applications described, as in its current form it operates on synthetic datasets.

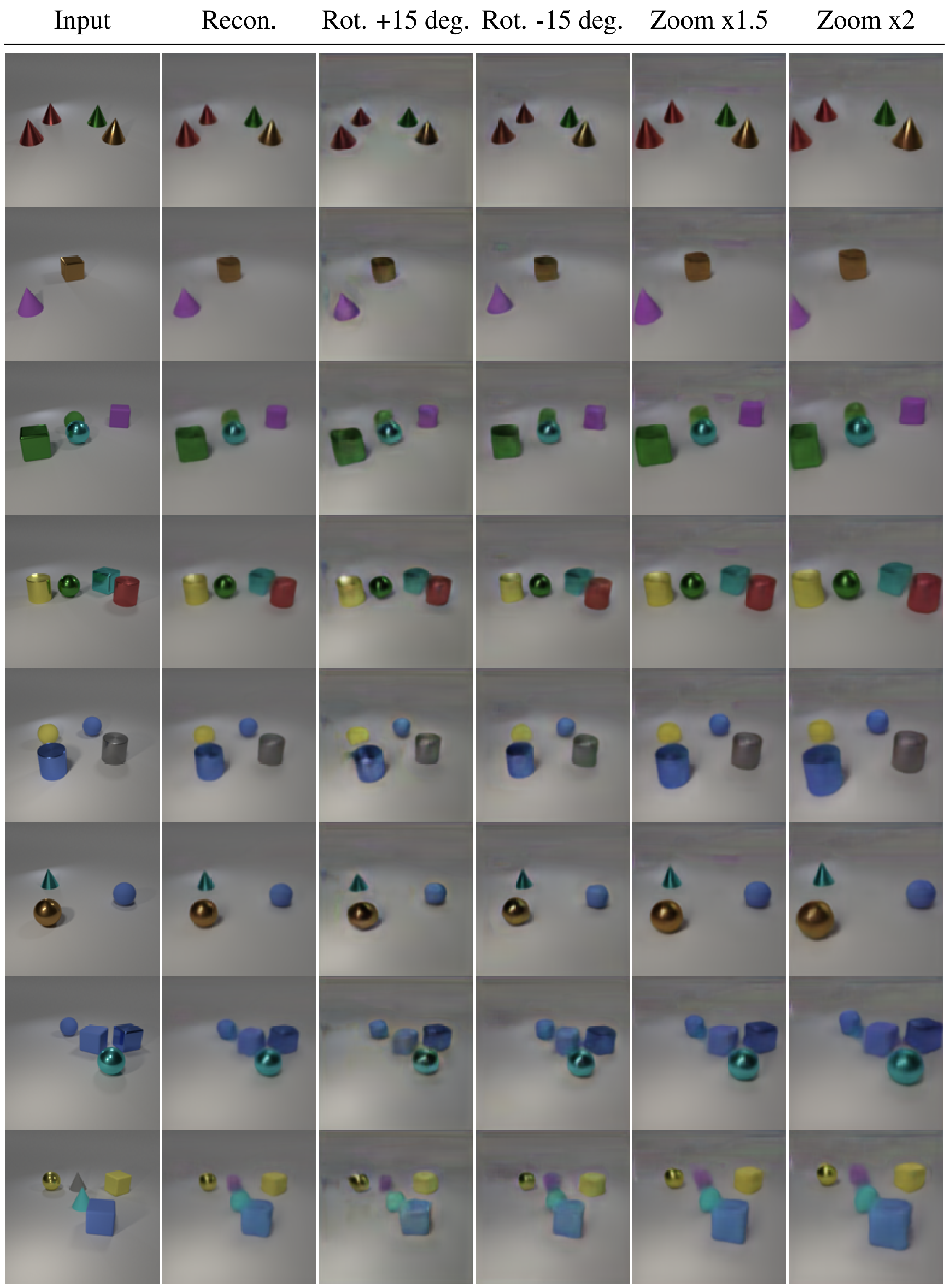


**Figure 10.6 – Novel view synthesis on CLEVR2345.** For each scene, we move the camera $\pm 15°$ on the azimuth axis. Additionally, we zoom in the scene twice. While our model is trained with a fixed default camera pose and single scene views, it is able to generalize to small camera pose modifications and render novel views of a scene.

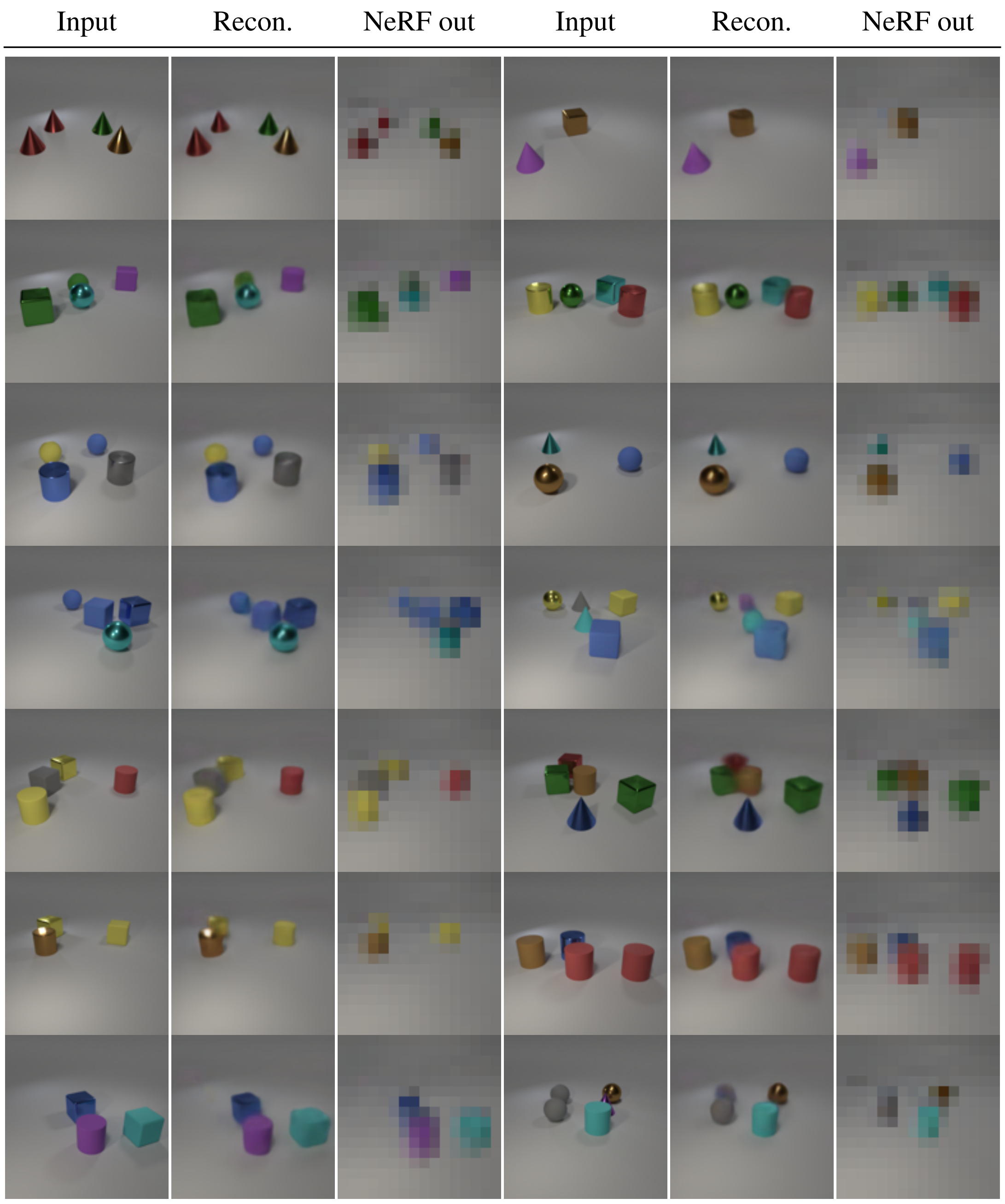


**Figure 10.7 – NeRF outputs on CLEVR2345.** For each input scene we show the reconstructed image as well as each the low resolution output of the NeRF function. This output is then upscaled with a neural network to obtain the reconstructed scene. While NeRF output lack full detail, they correctly depict each individual object and their position in the scene.

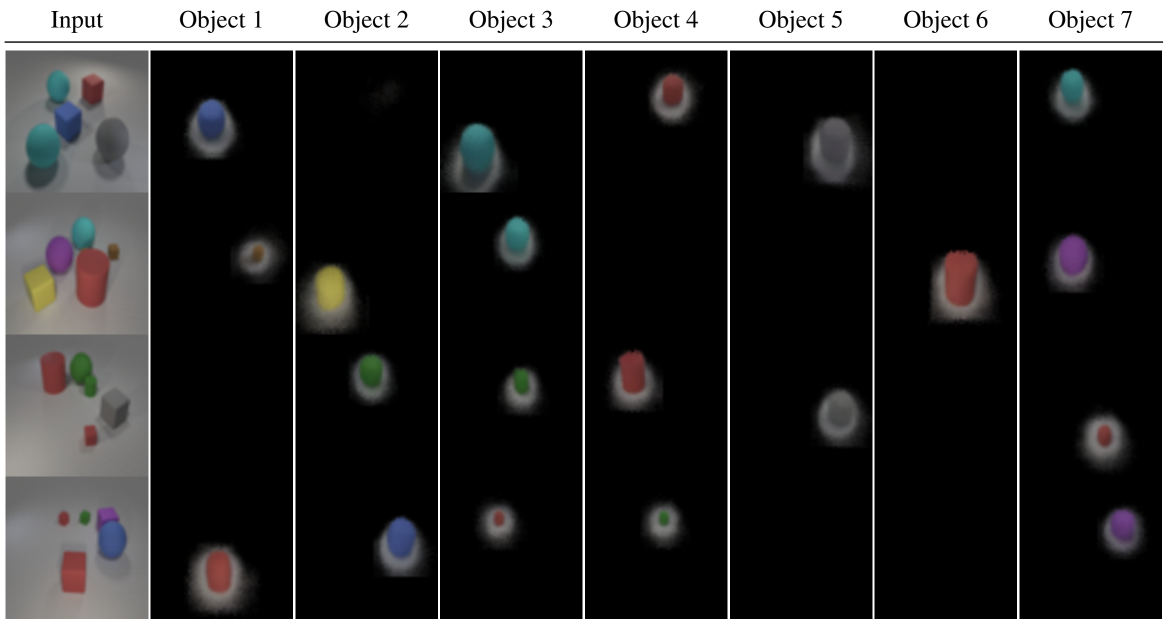


**Figure 10.8 – Object Slots rendered individually on CATER:** For each input scene we show the reconstructed image as well as each individual object slot rendered against a black background. We observe that each object slot is rendering a part of the scene corresponding to a single object instance.

# Chapter 11

# Discussion and Conclusions

The articles in this thesis present novel generative models, with a focus on applying them to dynamic scenes. The first two articles investigated different types of generative models for video, found some of their limitations, and proposed improved versions that produced better quality results or reduced the computational requirements of the generation process. On the other hand, the two last articles propose video models with inductive biases that better reflect the characteristics of real world dynamic scenes.

In the first article we sought to improve VAEs for video prediction. We first identified its different components, namely the prior and approximate posterior distributions and the likelihood model. For each component, we then analyzed some of its limitations, such as underfitting or limiting distribution families. We proposed a model incorporating mitigations for all the identified issues, including a higher capacity decoder and more expressive latent distributions and strategies to ease the optimization of these models. With these modifications, we showed that our model can generate better quality results under different metrics for two popular video prediction datasets. Furthermore, we also demonstrate that with the improvements our model can be used to predict videos on more realistic datasets than possible with previous methods.

The issues highlighted in this paper remain important for video prediction. With GANs being difficult to train and producing implausible continuations for video prediction, and autoregressive models having high computational requirements, VAEs have remained one of the most popular methods for video prediction. The current state-of-the-art in video prediction (Babaeizadeh et al., 2021) builds directly upon our finding that video prediction methods tend to underfit the training data, thus indicating that higher capacity models are needed. In fact, the major innovation in these models has been to replace the neural network architectures with higher capacity models inspired by advances in the image generation literature.

While our VAE model generated higher quality predictions than previous approaches, VAEs tend to produce worse quality outputs than GANs. In the case of videos, DVD-GAN (Clark et al., 2019) showed for the first time realistic generations from the Kinetics-600 dataset, which was not possible with any model before. However, DVD-GAN has very high computational requirements that prevented it from being widely adopted. Therefore, in the second article we investigated alternative GAN methods with reduced costs and we proposed a cascaded GAN model for video generation which consists of multiple levels that are trained sequentially and independently. Our model requires half the computational resources of DVD-GAN while retaining its generation quality due to decomposing the generation process into simpler steps and introducing an independence assumption. Furthermore, we use the better scaling capabilities of our model to generate videos of higher dimensionality than possible with previous GAN methods.

Despite our efforts, GANs for video have not been widely adopted by the video community. While for image GANs there are engineering techniques and neural architectures that alleviate the training instabilities, the equivalent best practices for training video GANs have not been found yet. In addition, even with the cost reductions proposed by our model or alternatives such as TGAN, video GANs still require large amounts of computational resources. There have been few developments in recent years, with the main development being an extension of DVD-GAN to video prediction (Luc et al., 2020).

Given the limitations of GANs and VAEs and the fact that most improvements were due to increased resources, we decided to focus on designing video models that implemented inductive biases that more accurately reflected our understanding of dynamic scenes. In the third article of this thesis we propose a model that models dynamic scenes as a set of entities that are updated independently but following the same transition rules. This reflects our understanding that the world is made of distinct objects and that they all share the same dynamics laws. While we only apply our model on simple 2D datasets, we show that our model can handle stochasticity and that it learns interpretable transition rules compared to previous approaches. Additionally, we show that our model generalizes to out-of-distribution scenes not seen during training due to its factorized representation.

While it is early to assess the impact of our model, it follows a recent line of research literature (Goyal et al., 2019, 2020) of object-centric models that implement a factorization between state and the transition function. Our model extends these previous works by adding the ability to handle multimodal future transitions and showing out-of-distribution generalization properties not seen in other models. Recent work (Lin et al., 2020; Locatello et al., 2020) has proposed different approaches towards implementing the same inductive biases, with similar limitations as our work. As a natural follow-up direction we expect the community to extend these models to more realistic scenarios and to show more applications of such models.

The last article proposes a model that combines object-centric scene representations with a differentiable 3D renderer. Our model explicitly factorizes object appearance and pose in a scene, where objects are represented as localized neural radiance fields (Mildenhall et al., 2020). Due to this scene representation, our model can generate out-of-distribution scene compositions not seen during training by adding or removing objects or modifying their poses and appearances. Furthermore, different from previous approaches we show that our model can be used to tackle downstream tasks that require tracking objects across time.

Similarly to the third article in the thesis, it is too early to assess its impact. Our work follows a line of research that incorporates differentiable rendering techniques with generative models (Niemeyer & Geiger, 2021; Stelzner et al., 2021; Yu et al., 2021b; Chen et al., 2021), and extends them with a factorized pose and appearance representation. While all these models operate on simple datasets, we expect follow-up work to extend them to more realistic scenarios and to show more downstream applications of these models. We believe that modeling 3D objects in video prediction is an important step necessary to systematically model object interactions and simulate camera motion, and we hope our model inspires future work in this direction.

While each of the articles presented in this thesis has been an improvement over previous approaches, there are still many open research directions concerning dynamic scenes. Currently, systematically solving simple tasks such as CATER and PHYRE is still an unsolved question. Specialized models designed exclusively for these tasks are not able to reach high success rates, and models based on learning representations videos are not as good as these models. As a positive note, at the time of writing this thesis, there has been progress in solving action classification tasks using general unsupervised methods. However, the limited success in tasks beyond classification, even in simple scenarios such as PHYRE, indicates that more research is needed to solve general dynamic scene understanding tasks.

In this thesis we focused on generative models of video. The main conclusion of our research is that improvements in model capacity have produced the best results so far. However, scaling traditional generative models such as GANs and VAEs has provided diminishing returns, and more research is needed to find alternatives to these models that produce better results with less computational resources. Models that implement better inductive biases, such as the models presented in our two last articles, are a promising avenue towards this goal, and is a research direction that is actively being explored by the community. Finally, at the time of writing, a novel family of generative methods called Diffusion models is gaining popularity and holds promise to improve the generations of other generative model families without requiring more computational resources.

Overall, it is an exciting time to conduct research in the field of video generation and we hope that this thesis and its findings motivate future developments in this area.

# References

Arjovsky, Martin, Chintala, Soumith, and Bottou, Léon. Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. PMLR, 2017.

Ayzel, Georgy, Scheffer, Tobias, and Heistermann, Maik. Rainnet v1. 0: a convolutional neural network for radar-based precipitation nowcasting. *Geoscientific Model Development*, 13(6):2631–2644, 2020.

Babaeizadeh, Mohammad, Finn, Chelsea, Erhan, Dumitru, Campbell, Roy H, and Levine, Sergey. Stochastic variational video prediction. In *International Conference on Learning Representations*, 2018.

Babaeizadeh, Mohammad, Saffar, Mohammad Taghi, Nair, Suraj, Levine, Sergey, Finn, Chelsea, and Erhan, Dumitru. Fitvid: Overfitting in pixel-level video prediction. *arXiv preprint arXiv:2106.13195*, 2021.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bakhtin, Anton, van der Maaten, Laurens, Johnson, Justin, Gustafson, Laura, and Girshick, Ross. Phyre: A new benchmark for physical reasoning. *Advances in Neural Information Processing Systems*, 32, 2019.

Ballas, Nicolas, Yao, Li, Pal, Chris, and Courville, Aaron. Delving deeper into convolutional networks for learning video representations. *arXiv preprint arXiv:1511.06432*, 2015.

Bengio, Yoshua. The consciousness prior. *arXiv preprint arXiv:1709.08568*, 2017.

Bishop, Christopher M and Nasrabadi, Nasser M. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Brock, Andrew, Donahue, Jeff, and Simonyan, Karen. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Brown, Tom B, Mann, Benjamin, Ryder, Nick, Subbiah, Melanie, Kaplan, Jared, Dhariwal, Prafulla, Neelakantan, Arvind, Shyam, Pranav, Sastry, Girish, Askell, Amanda, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Burgess, Christopher P, Matthey, Loic, Watters, Nicholas, Kabra, Rishabh, Higgins, Irina, Botvinick, Matt, and Lerchner, Alexander. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019a.

Burgess, Christopher P., Matthey, Loic, Watters, Nicholas, Kabra, Rishabh, Higgins, Irina, Botvinick, Matt, and Lerchner, Alexander. Monet: Unsupervised scene decomposition and representation, 2019b.

Carreira, Joao and Zisserman, Andrew. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*, pp. 6299–6308, 2017.

Carreira, Joao, Noland, Eric, Banki-Horvath, Andras, Hillier, Chloe, and Zisserman, Andrew. A short note about kinetics-600. *arXiv preprint arXiv:1808.01340*, 2018.

Castrejon, Lluis, Ballas, Nicolas, and Courville, Aaron. Improved conditional vrnns for video prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 7608–7617, 2019.

Chatterjee, Moitreya, Ahuja, Narendra, and Cherian, Anoop. A hierarchical variational neural uncertainty model for stochastic video prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9751–9761, 2021.

Chen, Chang, Deng, Fei, and Ahn, Sungjin. Roots: Object-centric representation and rendering of 3d scenes. *Journal of Machine Learning Research*, 22(259):1–36, 2021.

Cho, Kyunghyun, Van Merriënboer, Bart, Bahdanau, Dzmitry, and Bengio, Yoshua. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Chung, Junyoung, Kastner, Kyle, Dinh, Laurent, Goel, Kratarth, Courville, Aaron C, and Bengio, Yoshua. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, pp. 2980–2988, 2015.

Clark, Aidan, Donahue, Jeff, and Simonyan, Karen. Efficient video generation on complex datasets. *arXiv preprint arXiv:1907.06571*, 2019.

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

Cordts, Marius, Omran, Mohamed, Ramos, Sebastian, Rehfeld, Timo, Enzweiler, Markus, Benenson, Rodrigo, Franke, Uwe, Roth, Stefan, and Schiele, Bernt. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 3213–3223, 2016.

Crawford, Eric and Pineau, Joelle. Spatially invariant unsupervised object detection with convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33-01, pp. 3412–3420, 2019.

Crawford, Eric and Pineau, Joelle. Exploiting spatial invariance for scalable unsupervised object tracking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34-04, pp. 3684–3692, 2020.

Denton, Emily and Fergus, Rob. Stochastic video generation with a learned prior. In *International Conference on Machine Learning*, pp. 1182–1191, 2018.

Denton, Emily L, Chintala, Soumith, Fergus, Rob, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pp. 1486–1494, 2015.

Denton, Emily L et al. Unsupervised learning of disentangled representations from video. In *Advances in Neural Information Processing Systems*, pp. 4414–4423, 2017.

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Ding, David, Hill, Felix, Santoro, Adam, and Botvinick, Matt. Object-based attention for spatio-temporal reasoning: Outperforming neuro-symbolic models with flexible distributed architectures. *arXiv preprint arXiv:2012.08508*, 2020.

Dinh, Laurent, Sohl-Dickstein, Jascha, and Bengio, Samy. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Donahue, Jeff, Krähenbühl, Philipp, and Darrell, Trevor. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

Dumoulin, Vincent and Visin, Francesco. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

Dumoulin, Vincent, Belghazi, Ishmael, Poole, Ben, Mastropietro, Olivier, Lamb, Alex, Arjovsky, Martin, and Courville, Aaron. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

Dumoulin, Vincent, Perez, Ethan, Schucher, Nathan, Strub, Florian, Vries, Harm de, Courville, Aaron, and Bengio, Yoshua. Feature-wise transformations. *Distill*, 3(7):e11, 2018.

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

Eslami, SM, Heess, Nicolas, Weber, Theophane, Tassa, Yuval, Szepesvari, David, Kavukcuoglu, Koray, and Hinton, Geoffrey E. Attend, infer, repeat: Fast scene understanding with generative models. *arXiv preprint arXiv:1603.08575*, 2016.

Eslami, SM Ali, Jimenez Rezende, Danilo, Besse, Frederic, Viola, Fabio, Morcos, Ari S, Garnelo, Marta, Ruderman, Avraham, Rusu, Andrei A, Danihelka, Ivo, Gregor, Karol, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018.

Gidaris, Spyros, Singh, Praveer, and Komodakis, Nikos. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

Girdhar, Rohit and Ramanan, Deva. Cater: A diagnostic dataset for compositional actions and temporal reasoning. *arXiv preprint arXiv:1910.04744*, 2019.

Gkioxari, Georgia, Malik, Jitendra, and Johnson, Justin. Mesh r-cnn. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9785–9795, 2019.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in*

*Neural Information Processing Systems*, pp. 2672–2680, 2014.

Goodfellow, Ian, Bengio, Yoshua, Courville, Aaron, and Bengio, Yoshua. *Deep learning*, volume 1. MIT Press, 2016.

Goyal, Anirudh and Bengio, Yoshua. Inductive biases for deep learning of higher-level cognition. *arXiv preprint arXiv:2011.15091*, 2020.

Goyal, Anirudh, Lamb, Alex, Hoffmann, Jordan, Sodhani, Shagun, Levine, Sergey, Bengio, Yoshua, and Schölkopf, Bernhard. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

Goyal, Anirudh, Lamb, Alex, Gampa, Phanideep, Beaudoin, Philippe, Levine, Sergey, Blundell, Charles, Bengio, Yoshua, and Mozer, Michael. Object files and schemata: Factorizing declarative and procedural knowledge in dynamical systems. *arXiv preprint arXiv:2006.16225*, 2020.

Goyal, Anirudh, Didolkar, Aniket, Ke, Nan Rosemary, Blundell, Charles, Beaudoin, Philippe, Heess, Nicolas, Mozer, Michael, and Bengio, Yoshua. Neural production systems. *arXiv preprint arXiv:2103.01937*, 2021.

Goyal, Priya, Dollár, Piotr, Girshick, Ross, Noordhuis, Pieter, Wesolowski, Lukasz, Kyrola, Aapo, Tulloch, Andrew, Jia, Yangqing, and He, Kaiming. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Greff, Klaus, Kaufman, Raphaël Lopez, Kabra, Rishabh, Watters, Nick, Burgess, Christopher, Zoran, Daniel, Matthey, Loic, Botvinick, Matthew, and Lerchner, Alexander. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, pp. 2424–2433. PMLR, 2019a.

Greff, Klaus, Kaufman, Raphaël Lopez, Kabra, Rishabh, Watters, Nick, Burgess, Chris, Zoran, Daniel, Matthey, Loic, Botvinick, Matthew, and Lerchner, Alexander. Multi-object representation learning with iterative variational inference, 2019b.

Hafner, Danijar, Lillicrap, Timothy, Fischer, Ian, Villegas, Ruben, Ha, David, Lee, Honglak, and Davidson, James. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *CVPR*, 2016.

Heusel, Martin, Ramsauer, Hubert, Unterthiner, Thomas, Nessler, Bernhard, and Hochreiter, Sepp. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

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

Hoffman, Matthew D and Johnson, Matthew J. Elbo surgery: Yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, 2016.

Huang, Gao, Liu, Zhuang, Van Der Maaten, Laurens, and Weinberger, Kilian Q. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4700–4708, 2017.

Jang, Eric, Gu, Shixiang, and Poole, Ben. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Jiang, Jindong, Janghorbani, Sepehr, de Melo, Gerard, and Ahn, Sungjin. Scalable object-oriented sequential generative models. *Unknown Journal*, 2019a.

Jiang, Jindong, Janghorbani, Sepehr, De Melo, Gerard, and Ahn, Sungjin. Scalor: Generative world models with scalable object representations. *arXiv preprint arXiv:1910.02384*, 2019b.

Johnson, Justin, Hariharan, Bharath, Van Der Maaten, Laurens, Fei-Fei, Li, Lawrence Zitnick, C, and Girshick, Ross. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2901–2910, 2017.

Jordan, Michael I, Ghahramani, Zoubin, Jaakkola, Tommi S, and Saul, Lawrence K. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

Jumper, John, Evans, Richard, Pritzel, Alexander, Green, Tim, Figurnov, Michael, Ronneberger, Olaf, Tunyasuvunakool, Kathryn, Bates, Russ, Žídek, Augustin, Potapenko, Anna, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596 (7873):583–589, 2021.

Kahneman, Daniel, Treifsman, Anne, and Gibbs, Brian J. The reviewing of object files: Object-specific integration of information. *Cognitive psychology*, 24(2):175–219, 1992.

Kalchbrenner, Nal, van den Oord, Aäron, Simonyan, Karen, Danihelka, Ivo, Vinyals, Oriol, Graves, Alex, and Kavukcuoglu, Koray. Video pixel networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1771–1779, 2017.

Karras, Tero, Aila, Timo, Laine, Samuli, and Lehtinen, Jaakko. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

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

Kingma, Diederik P and Ba, Jimmy. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kingma, Durk P and Dhariwal, Prafulla. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10215–10224, 2018.

Kingma, Durk P, Salimans, Tim, Jozefowicz, Rafal, Chen, Xi, Sutskever, Ilya, and Welling, Max. Improved variational inference with inverse autoregressive flow. In *Advances in*

*Neural Information Processing Systems*, pp. 4743–4751, 2016.

Kipf, Thomas, Elsayed, Gamaleldin F, Mahendran, Aravindh, Stone, Austin, Sabour, Sara, Heigold, Georg, Jonschkowski, Rico, Dosovitskiy, Alexey, and Greff, Klaus. Conditional object-centric learning from video. *arXiv preprint arXiv:2111.12594*, 2021.

Kosiorek, Adam R, Kim, Hyunjik, Posner, Ingmar, and Teh, Yee Whye. Sequential attend, infer, repeat: Generative modelling of moving objects. *arXiv preprint arXiv:1806.01794*, 2018.

Kosiorek, Adam R, Strathmann, Heiko, Zoran, Daniel, Moreno, Pol, Schneider, Rosalia, Mokrá, Soňa, and Rezende, Danilo J. Nerf-vae: A geometry aware 3d scene generative model. *arXiv preprint arXiv:2104.00587*, 2021.

Kossen, Jannik, Stelzner, Karl, Hussing, Marcel, Voelcker, Claas, and Kersting, Kristian. Structured object-aware physics prediction for video modeling and planning. *arXiv preprint arXiv:1910.02425*, 2019.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Kumar, Manoj, Babaeizadeh, Mohammad, Erhan, Dumitru, Finn, Chelsea, Levine, Sergey, Dinh, Laurent, and Kingma, Durk. Videoflow: A flow-based generative model for video. *arXiv preprint arXiv:1903.01434*, 2(5), 2019.

Larochelle, Hugo and Murray, Iain. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 29–37, 2011.

Larsen, Anders Boesen Lindbo, Sønderby, Søren Kaae, Larochelle, Hugo, and Winther, Ole. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *nature*, 521(7553): 436–444, 2015.

Lee, Alex X, Zhang, Richard, Ebert, Frederik, Abbeel, Pieter, Finn, Chelsea, and Levine, Sergey. Stochastic adversarial video prediction. *arXiv preprint arXiv:1804.01523*, 2018.

Lin, Zhixuan, Wu, Yi-Fu, Peri, Skand, Fu, Bofeng, Jiang, Jindong, and Ahn, Sungjin. Improving generative imagination in object-centric world models. In *International Conference on Machine Learning*, pp. 6140–6149. PMLR, 2020.

Locatello, Francesco, Bauer, Stefan, Lucic, Mario, Rätsch, Gunnar, Gelly, Sylvain, Schölkopf, Bernhard, and Bachem, Olivier. Challenging common assumptions in the unsupervised learning of disentangled representations. *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*, 2018.

Locatello, Francesco, Weissenborn, Dirk, Unterthiner, Thomas, Mahendran, Aravindh, Heigold, Georg, Uszkoreit, Jakob, Dosovitskiy, Alexey, and Kipf, Thomas. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020.

Locher, Paul J. How does a visual artist create an artwork. *The Cambridge handbook of creativity*, pp. 131–144, 2010.

Luc, Pauline, Neverova, Natalia, Couprie, Camille, Verbeek, Jakob, and LeCun, Yann. Predicting deeper into the future of semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 648–657, 2017.

Luc, Pauline, Couprie, Camille, Lecun, Yann, and Verbeek, Jakob. Predicting future instance segmentation by forecasting convolutional features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 584–599, 2018.

Luc, Pauline, Clark, Aidan, Dieleman, Sander, Casas, Diego de Las, Doron, Yotam, Cassirer, Albin, and Simonyan, Karen. Transformation-based adversarial video prediction on large-scale data. *arXiv preprint arXiv:2003.04035*, 2020.

Lucic, Mario, Kurach, Karol, Michalski, Marcin, Gelly, Sylvain, and Bousquet, Olivier. Are gans created equal? a large-scale study. *Advances in neural information processing systems*, 31, 2018.

Maaløe, Lars, Sønderby, Casper Kaae, Sønderby, Søren Kaae, and Winther, Ole. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016.

Maaløe, Lars, Fraccaro, Marco, Liévin, Valentin, and Winther, Ole. Biva: A very deep hierarchy of latent variables for generative modeling. *arXiv preprint arXiv:1902.02102*, 2019.

Mao, Xudong, Li, Qing, Xie, Haoran, Lau, Raymond YK, Wang, Zhen, and Paul Smolley, Stephen. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2794–2802, 2017.

Mathieu, Michael, Couprie, Camille, and LeCun, Yann. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

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

Mildenhall, Ben, Srinivasan, Pratul P, Tancik, Matthew, Barron, Jonathan T, Ramamoorthi, Ravi, and Ng, Ren. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pp. 405–421. Springer, 2020.

Miyato, Takeru, Kataoka, Toshiki, Koyama, Masanori, and Yoshida, Yuichi. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mullally, Sinéad L and Maguire, Eleanor A. Memory, imagination, and predicting the future: a common brain mechanism? *The Neuroscientist*, 20(3):220–234, 2014.

Murphy, Kevin P. *Machine learning: a probabilistic perspective.* MIT press, 2012.

Murphy, Kevin P. *Probabilistic Machine Learning: Advanced Topics.* MIT Press, 2023. URL `probml.ai`.

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

Nguyen-Phuoc, Thu, Richardt, Christian, Mai, Long, Yang, Yong-Liang, and Mitra, Niloy. Blockgan: Learning 3d object-aware scene representations from unlabelled images. *arXiv preprint arXiv:2002.08988*, 2020.

Niemeyer, Michael and Geiger, Andreas. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11453–11464, 2021.

Oh, Junhyuk, Guo, Xiaoxiao, Lee, Honglak, Lewis, Richard L, and Singh, Satinder. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pp. 2863–2871, 2015.

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

Pearl, Judea. *Causality: Models, Reasoning, and Inference. 2nd edition.* Cambridge University Press, New York, NY,, 2009.

Peters, Jonas, Janzing, Dominik, and Schölkopf, Bernhard. *Elements of causal inference: foundations and learning algorithms.* The MIT Press, 2017.

Pumarola, Albert, Corona, Enric, Pons-Moll, Gerard, and Moreno-Noguer, Francesc. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10318–10327, 2021.

Qi, Haozhi, Wang, Xiaolong, Pathak, Deepak, Ma, Yi, and Malik, Jitendra. Learning long-term visual dynamics with region proposal interaction networks. *arXiv preprint arXiv:2008.02265*, 2020.

Radford, Alec, Narasimhan, Karthik, Salimans, Tim, and Sutskever, Ilya. Improving language understanding by generative pre-training. *OpenAI website*, 2018.

Rand, William M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Ranganath, Rajesh, Tran, Dustin, and Blei, David. Hierarchical variational models. In *International Conference on Machine Learning*, pp. 324–333, 2016.

Ranzato, MarcAurelio, Szlam, Arthur, Bruna, Joan, Mathieu, Michael, Collobert, Ronan, and Chopra, Sumit. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014.

Reed, Scott, van den Oord, Aäron, Kalchbrenner, Nal, Colmenarejo, Sergio Gómez, Wang, Ziyu, Chen, Yutian, Belov, Dan, and de Freitas, Nando. Parallel multiscale autoregressive density estimation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2912–2921, 2017.

Rematas, Konstantinos and Ferrari, Vittorio. Neural voxel renderer: Learning an accurate and controllable rendering tool. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5417–5427, 2020.

Rezende, Danilo Jimenez and Mohamed, Shakir. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Roelfsema, Pieter R, Lamme, Victor AF, and Spekreijse, Henk. Object-based attention in the primary visual cortex of the macaque monkey. *Nature*, 395(6700):376–381, 1998.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Saito, Masaki and Saito, Shunta. Tganv2: Efficient training of large models for video generation with multiple subsampling layers. *arXiv preprint arXiv:1811.09245*, 2018.

Saito, Masaki, Matsumoto, Eiichi, and Saito, Shunta. Temporal generative adversarial nets with singular value clipping. In *ICCV*, 2017.

Saxena, Vaibhav, Ba, Jimmy, and Hafner, Danijar. Clockwork variational autoencoders. *Advances in Neural Information Processing Systems*, 34, 2021.

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

Silver, David, Schrittwieser, Julian, Simonyan, Karen, Antonoglou, Ioannis, Huang, Aja, Guez, Arthur, Hubert, Thomas, Baker, Lucas, Lai, Matthew, Bolton, Adrian, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Sitzmann, Vincent, Zollhöfer, Michael, and Wetzstein, Gordon. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *arXiv preprint arXiv:1906.01618*, 2019.

Sitzmann, Vincent, Chan, Eric R, Tucker, Richard, Snavely, Noah, and Wetzstein, Gordon. Metasdf: Meta-learning signed distance functions. *arXiv preprint arXiv:2006.09662*, 2020a.

Sitzmann, Vincent, Martel, Julien, Bergman, Alexander, Lindell, David, and Wetzstein, Gordon. Implicit neural representations with periodic activation functions. *Advances in*

*Neural Information Processing Systems*, 33, 2020b.

Sønderby, Casper Kaae, Raiko, Tapani, Maaløe, Lars, Sønderby, Søren Kaae, and Winther, Ole. Ladder variational autoencoders. In *Advances in Neural Information Processing Systems*, pp. 3738–3746, 2016a.

Sønderby, Casper Kaae, Raiko, Tapani, Maaløe, Lars, Sønderby, Søren Kaae, and Winther, Ole. How to train deep variational autoencoders and probabilistic ladder networks. In *33rd International Conference on Machine Learning (ICML 2016) International Conference on Machine Learning*, 2016b.

Soomro, Khurram, Zamir, Amir Roshan, and Shah, Mubarak. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

Spelke, Elizabeth S, Breinlinger, Karen, Jacobson, Kristen, and Phillips, Ann. Gestalt relations and object perception: A developmental study. *Perception*, 22(12):1483–1501, 1993.

Srivastava, Nitish, Mansimov, Elman, and Salakhudinov, Ruslan. Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning*, pp. 843–852, 2015a.

Srivastava, Rupesh Kumar, Greff, Klaus, and Schmidhuber, Jürgen. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015b.

Stelzner, Karl, Peharz, Robert, and Kersting, Kristian. Faster attend-infer-repeat with tractable probabilistic models. In *International Conference on Machine Learning*, pp. 5966–5975. PMLR, 2019.

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

Tulyakov, Sergey, Liu, Ming-Yu, Yang, Xiaodong, and Kautz, Jan. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1526–1535, 2018.

Unterthiner, Thomas, van Steenkiste, Sjoerd, Kurach, Karol, Marinier, Raphael, Michalski, Marcin, and Gelly, Sylvain. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Łukasz, and Polosukhin, Illia. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Villegas, Ruben, Yang, Jimei, Hong, Seunghoon, Lin, Xunyu, and Lee, Honglak. Decomposing motion and content for natural video sequence prediction. *arXiv preprint arXiv:1706.08033*, 2017a.

Villegas, Ruben, Yang, Jimei, Zou, Yuliang, Sohn, Sungryull, Lin, Xunyu, and Lee, Honglak. Learning to generate long-term future via hierarchical prediction. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3560–3569. JMLR.

org, 2017b.

Villegas, Ruben, Pathak, Arkanath, Kannan, Harini, Erhan, Dumitru, Le, Quoc V, and Lee, Honglak. High fidelity video prediction with large stochastic recurrent neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Vondrick, Carl, Pirsiavash, Hamed, and Torralba, Antonio. Anticipating visual representations from unlabeled video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 98–106, 2016a.

Vondrick, Carl, Pirsiavash, Hamed, and Torralba, Antonio. Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*, pp. 613–621, 2016b.

Wang, Yuxuan, Skerry-Ryan, RJ, Stanton, Daisy, Wu, Yonghui, Weiss, Ron J, Jaitly, Navdeep, Yang, Zongheng, Xiao, Ying, Chen, Zhifeng, Bengio, Samy, et al. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*, 2017.

Wang, Zhou, Bovik, Alan C, Sheikh, Hamid R, and Simoncelli, Eero P. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004a.

Wang, Zhou, Bovik, Alan C, Sheikh, Hamid R, Simoncelli, Eero P, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004b.

Weissenborn, Dirk, Täckström, Oscar, and Uszkoreit, Jakob. Scaling autoregressive video models. *International Conference on Learning Representations*, 2020.

Williams, Ronald J and Zipser, David. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Wu, Yuxin and He, Kaiming. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Xingjian, SHI, Chen, Zhourong, Wang, Hao, Yeung, Dit-Yan, Wong, Wai-Kin, and Woo, Wang-chun. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pp. 802–810, 2015.

Xiong, Wei, Luo, Wenhan, Ma, Lin, Liu, Wei, and Luo, Jiebo. Learning to generate time-lapse videos using multi-stage dynamic generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2364–2373, 2018.

Xue, Tianfan, Wu, Jiajun, Bouman, Katherine, and Freeman, Bill. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Advances in neural information processing systems*, pp. 91–99, 2016.

Yan, Wilson, Zhang, Yunzhi, Abbeel, Pieter, and Srinivas, Aravind. Videogpt: Video generation using vq-vae and transformers. *arXiv preprint arXiv:2104.10157*, 2021.

Yu, Alex, Ye, Vickie, Tancik, Matthew, and Kanazawa, Angjoo. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4578–4587, 2021a.

Yu, Fisher, Xian, Wenqi, Chen, Yingying, Liu, Fangchen, Liao, Mike, Madhavan, Vashisht, and Darrell, Trevor. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

Yu, Hong-Xing, Guibas, Leonidas J, and Wu, Jiajun. Unsupervised discovery of object radiance fields. *arXiv preprint arXiv:2107.07905*, 2021b.

Zhang, Richard, Isola, Phillip, Efros, Alexei A, Shechtman, Eli, and Wang, Oliver. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 586–595, 2018a.

Zhang, Richard, Isola, Phillip, Efros, Alexei A, Shechtman, Eli, and Wang, Oliver. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018b.

Zhao, Long, Peng, Xi, Tian, Yu, Kapadia, Mubbasir, and Metaxas, Dimitris N. Towards image-to-video translation: A structure-aware approach via multi-stage generative adversarial networks. *International Journal of Computer Vision*, 2020.

Zhou, Honglu, Kadav, Asim, Lai, Farley, Niculescu-Mizil, Alexandru, Min, Martin Renqiang, Kapadia, Mubbasir, and Graf, Hans Peter. Hopper: Multi-hop transformer for spatiotemporal reasoning. *arXiv preprint arXiv:2103.10574*, 2021.