Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://cas.tudelft.nl/

CAS-2022-888

M.Sc. Thesis

Modular Neural Networks for Video Prediction

Nan Lin

Abstract

Modular neural networks have received an upsurge of attention lately owing to their unique modular design and potential capacity to decompose complex dynamics and learn interactions among causal variables. Inspired by this potential, we employ the recently introduced Recurrent Independent Mechanisms (RIMs) in the downstream video prediction task. RIMs consist of several modular recurrent units and modular hidden states which are called RIM cells. Those modules are connected by two attention mechanisms. Through experiments, we show that RIMs perform better or comparably with related baselines.

From modular recurrent units to modular image representations, we push the modularity further to explore how much the performance can benefit from it. We extend RIMs architecture on both the encoder and decoder sides to allow for object-centric (OC) feature representation learning in video prediction, resulting in an end-to-end architecture we refer to as `OC-RIMs`. Our qualitative evaluations demonstrate that every RIM cell in `OC-RIMs` now attends to a certain object within the input scene at any specific moment. As a result, `OC-RIMs` offer considerable quantitative performance improvement in video prediction over comparable baselines across two datasets.

We perform extensive ablation studies to validate the design choices of every module of RIMs. We empirically show that most modules work as expected. However, the sparse activation greatly detriments the prediction performance, which is against the claims in the paper where RIMs were proposed. On the other hand, RIM cells are expected to work near-independently. But experiments show that the use of communication mechanism leads to heavy co-adaptation between cells, i.e., RIM cells fail to make any reasonable predictions independently. Those issues have raised our concerns about the design of RIMs. Finally, we point out some future work directions to address these deficiencies. [a]

[a] Code of our experiments will later be made publicly available.

Modular Neural Networks for Video Prediction

Nan Lin

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Department of Microelectronics & Computer Engineering
Circuits and Systems Group

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Modular Neural Networks for Video Prediction"** by **Nan Lin** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated:

Chairman: _____
dr.ir. J. Dauwels

Advisor: _____
dr.ir. H. Jamali-Rad

Committee Members: _____
dr.ir. J. Dauwels

_____
dr.ir. H. Jamali-Rad

Abstract

Modular neural networks have received an upsurge of attention lately owing to their unique modular design and potential capacity to decompose complex dynamics and learn interactions among causal variables. Inspired by this potential, we employ the recently introduced Recurrent Independent Mechanisms (RIMs) in the downstream video prediction task. RIMs consist of several modular recurrent units and modular hidden states which are called RIM cells. Those modules are connected by two attention mechanisms. Through experiments, we show that RIMs perform better or comparably with related baselines.

From modular recurrent units to modular image representations, we push the modularity further to explore how much the performance can benefit from it. We extend RIMs architecture on both the encoder and decoder sides to allow for object-centric (OC) feature representation learning in video prediction, resulting in an end-to-end architecture we refer to as `OC-RIMs`. Our qualitative evaluations demonstrate that every RIM cell in `OC-RIMs` now attends to a certain object within the input scene at any specific moment. As a result, `OC-RIMs` offer considerable quantitative performance improvement in video prediction over comparable baselines across two datasets.

We perform extensive ablation studies to validate the design choices of every module of RIMs. We empirically show that most modules work as expected. However, the sparse activation greatly detriments the prediction performance, which is against the claims in the paper where RIMs were proposed. On the other hand, RIM cells are expected to work near-independently. But experiments show that the use of communication mechanism leads to heavy co-adaptation between cells, i.e., RIM cells fail to make any reasonable predictions independently. Those issues have raised our concerns about the design of RIMs. Finally, we point out some future work directions to address these deficiencies. [1]

[1] Code of our experiments will later be made publicly available.

Acknowledgments

I would like to first thank my supervisors dr.ir. J. Dauwels and dr.ir. H. Jamali-Rad for their assistance during the writing of this thesis. Without them, this would not have been possible. I grew a lot under their guidance and through working with them, both in terms of knowledge and as a person.

I would like to thank my friends at TU Delft. They gave me a great time and company while I am in the Netherlands. They have also given me invaluable support when I am low and upset. I want to thank my parents for supporting me, both financially and mentally.

I must thank all the teachers that have taught or helped me during my study. All of them have been very friendly to me.

I also want to thank the setbacks that I have faced during this two-year program. This study has not been as easy as I expected. I appreciate the challenges because only through them can I keep growing stronger.

Lastly, I want to thank myself for choosing the field of machine learning because computer science especially machine learning has always been my favorite field. I realize how much I still lack in becoming an adequate data scientist. Fortunately, I believe I still have time to catch up and pursue what I find meaningful and valuable. I hope that in the future I will stick in this direction and make unique contributions that only I can make and finally become the person that I want to be.

Nan Lin
Delft, the Netherlands

Contents

List of Figures

List of Tables

1 Introduction

Machine Learning (ML) is a type of Artificial Intelligence (AI) method that learns from data to perform specific tasks. ML algorithms build a model that needs to be trained on so-called "training data" before being applied. The tasks ML addresses span a wide range. Algorithms that work on training data with labels are referred to as supervised learning, where the goal is to predict the labels. Others that work on data without labels are called unsupervised learning. An example would be clustering, which maximizes inter-group differences and intra-group similarities. In recent years, deep learning has become the most studied research topic in the ML world owing to its extraordinary performance. Deep learning is based on artificial neural networks (ANNs). "Deep" refers to the stacking of multiple layers in these models, which results in millions or even billions of parameters. Training these models was made possible by the amount of available data which grows exponentially every year and the rapid growth of Graphics Processing Unit (GPU) computation power. power. Over the years, deep learning models have experienced significant performance improvement in various applications. In one of the most studied tasks, image recognition, their accuracy even exceeds human [5]. Other applications range from machine translation [6] to medical image analysis [7] and solving partial differential equations [8].

However, while the performance gets increasingly satisfying, other issues such as robustness, data privacy, and interpretability of deep learning models have raised more and more concerns [9]. Compared with conventional ML methods like linear regression and decision trees which are easier to explain [10], deep learning models are hard to interpret due to their complex architecture. Nonetheless, researchers have tried to build more explainable AI models from multiple perspectives [11], but these models still face issues like being hard to scale up. Meanwhile, compositional AI (composable AI) [12, 13, 14] has become a new promising way to build more robust and explainable models while maintaining excellent performance. Instead of building a single large model that deals with a complex task, compositional AI aims at building compositions of modular blocks where each of them deals with a sub-task. This type of model also lays the foundation for integrating knowledge representation into deep learning, which has huge potential of becoming the next generation AI.

Modular Neural Networks (MNNs) are a representative example of compositional deep learning models. It aims at decomposing the dynamics and interactions in complex environments and tasks. The latest MNNs [15, 16, 17, 18] process the input with compositions of modular blocks and depict causal relationships [19] using attention mechanisms [20] on the learned latent representations. However, the actual capability of such networks has not been fully explored, and we lack an understanding of how these modules work. The test bed for MNNs should be challenging enough and the results should reflect and help us understand their composability. In this study, we aim to research a typical MNN named Recurrent Independent Mechanisms (RIMs) [21].

RIMs contain a group of so-called RIM cells, which consists of a recurrent unit such as a Gated Recurrent Unit (GRU) [22] and a hidden state vector. It selectively allocates input to different RIM cells and updates the hidden states of those selected. Out of all the tasks RIMs can handle, video prediction is a challenging one since it requires learning spatio-temporal dependencies and making pixel-level predictions. For datasets like Moving MNIST [23], the input data is highly structured, which is exactly where MNNs can be applied. Therefore we apply RIMs to video prediction downstream tasks and compare their performance to non-modular neural network baselines.

Through our pilot experiments, we notice that the input allocation layer plays an important role in specializing the RIM cells. However, the default way of formulating the input is inefficient for RIMs and the model is still no more explainable than monolithic models because the role of each RIM cell cannot be interpreted. To further extend the modularity of RIMs and have more interpretability for the modular structure, we extend RIMs by introducing modularity in the input and output. We take inspiration from object-centric (OC) learning. While MNNs modularize the dynamics, OC representations modularize the representation of visual scenes. OC representation learning is built upon the idea that natural scenes can be better modeled as compositions of objects and their relations, in contrast to distributed representations [15, 16, 17, 18]. Such inductive bias can be modeled into neural networks as a set of transformations of the data into a set of vector representations, each corresponding to an individual object [16, 24, 25].

Inspired by the advent of RIMs, recently proposed variational independent mechanisms (VIMs)[26] combined the object-centric modules, namely slot attention[24] and component-wise spatial broadcast decoder (SBD)[27] with a variational counterpart of RIMs. It is shown showed that VIMs are capable of learning OC representations through object interactions [26]. However, this work focused on adapting deterministic RIMs to a variational model. Yet, the impacts of object-centric representation are not clear in terms of downstream task performance. In this study, we explore such impacts on modular networks by focusing on applying RIMs to video prediction tasks, where the representation of spatio-temporal patterns plays a central role. To tailor RIMs toward downstream video prediction tasks, we take extend RIMs architecture by two extra components: (i) a slot attention unit on the encoder side to extract object-specific features, (ii) a component-wise decoder (namely, component-wise SBD or basic component-wise decoder (BCD)). We refer to the resulting model as object-centric RIMs (`OC-RIMs`).

We study the capability of RIMs and `OC-RIMs` by studying its performance in downstream video prediction tasks. With the new modules in `OC-RIMs`, our model is able to decompose any input scene into objects. Therefore, we are able to further study the design choices of RIMs by looking at their decomposition ability, tracking consistency, input attention mechanism, etc.

We demonstrate that `OC-RIMs` decompose the input scene into modular representations, where each RIM cell now attends to a certain object at any specific moment. More importantly, when compared to other baselines, this decomposability results in significantly better performance in the downstream prediction tasks - in terms of mean squared error (MSE) as well as structural similarity (SSIM) across Moving MNIST [23] and Moving Sprites [28] datasets.

The contributions of this study can be summarized as follows:

- We propose `OC-RIMs`, an end-to-end RIM-based architecture, which is able to learn object-wise spatio-temporal patterns in an unsupervised way. `OC-RIMs` factorize object representations into different slots and decode distinct hidden states using a component-wise decoder.
- We structurally approach the problem of video prediction with RIMs and `OC-RIMs`, providing both qualitative and quantitative results on its impact on different datasets.
- We demonstrate that `OC-RIMs` improve the downstream video prediction performance considerably when compared to vanilla RIMs and a few other related baselines.
- We perform rich ablation studies and provide a detailed analysis of how each component in `OC-RIMs` works with each other, based on which we validate all the design choices of RIMs and provide suggestions for further study on this architecture and video prediction tasks.

2 Literature Review

In this chapter, we intend to review the recent works related to our research in order to provide an introduction to the field and existing research gaps. We will first introduce Convolutional Neural Networks (CNNs) as a tool to learn spatial information in images in the first section. After that, starting with a general introduction to Recurrent Neural Networks (RNNs), we will introduce this generic deep learning method for learning temporal relationships in the second section. We will also cover 3D convolution as an alternative. Before moving to modular networks, in the third section, we will introduce a core component in it, which is the attention mechanism. Next, we will talk about the recently proposed modular networks. In the fifth section, we will introduce object-centric learning, which is another core concept to extend an existing model. Finally, we will discuss the downstream video prediction task, which will be our testbed for evaluating our models.

2.1 Learning Spatial Dependencies from Images

To process perceptual visual input, i.e., images, there have emerged various deep learning methods, ranging from Multi-layer Perceptron (MLP) to CNNs and visual transformers. Dating back to [29], a primal convolution network was trained by backpropagation and employed to recognize handwritten digits. CNNs are a class of artificial neural networks designed for image data, which mainly consist of convolution layers(`conv2d`), pooling layers, and fully connected layers.

In this section, we will focus on CNNs, which is the mainstream for roughly the last ten years, since MLPs have an intrinsic disadvantage of not fully utilizing local dependencies, and visual transformers are not fully studied yet not a focus of our work.

Convolutional layers. Image convolution (correlation) is a common operation in the image processing domain. The most used convolution kernels include Gaussian filters, bilateral filters, Laplacian filters, and so on. For example, Gaussian filters are able to filter high-frequency noise in the image, and Laplacian filters can extract edges. Extending such an idea to deep learning, if we learn the weights of these filters from data, we will get a convolutional layer.

Pooling layers. They reduce the spatial dimensions of a feature map by combining local features through aggregating operations such as maximization, averaging, and so on. Such operations are also able to enlarge the receptive field of each feature.

Fully connected layers. Also known as dense layers. After convolutional layers and pooling layers, we will get a feature map of size $\texttt{Channel} \times \texttt{Height} \times \texttt{Weight}$. For classification (and sometimes object detection), we would flatten all three dimensions such that we get a vector that we feed into fully connected layers to get classification output.

The flourishing of CNNs started in [29], where back-propagation was proposed to train the networks. But it was in 2012 when AlexNet [1] was proposed, that a new epoch of CNNs started. CNNs have become widely used to extract visual features/representations from images. AlexNet represents a classic style of CNNs for image recognition/classification. It starts with five convolutional layers which transform an image into vectors. Then there follows three dense layers and a 1000-dimensional softmax output layer. AlexNet has a relatively small number of convolutional layers. An illustration of AlexNet architecture is shown in Figure 2.1. The kernel sizes are first large, and then gradually smaller. In this way, the feature maps can quickly be downsampled while the features can have a large receptive field. However, due to the GPU limitations back at the time, AlexNet has a symmetrical two-stream structure so that they can be efficiently trained on multiple GPUs. As a result, AlexNet outperformed all the prior methods in the ImageNet challenge.

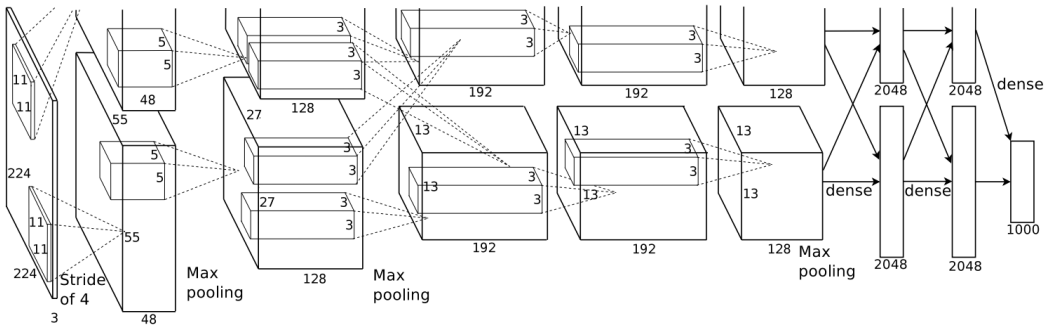

Figure 2.1: Illustration of AlexNet [1]. It has a two-stream structure and each contains five convolutional layers and two dense layers. The convolutional layers have different kernel sizes.

VGGNet [30] replaces the variable kernel size with a stack of fixed-size small kernels (3×3). The resulting feature from a stack of small kernels can have the same receptive field as from a large kernel. Therefore, VGGNets are much deeper (i.e., has more layers) than AlexNet. The proposed 16-layer and 19-layer VGGNets, VGG16 and VGG19, have become a new commonly adopted backbone for image tasks and they showed significant improvement on AlexNet.

VGGNet is an example of the trend of trying to go deeper with CNNs. Inception [2] is another attempt to build deeper CNNs. The name Inception refers to a module that comprises a complete network. Each module takes a feature map (or an image) as input and performs resolution-preserving operations. These operations include, 1×1 convolution, 3×3 convolution, 5×5 convolution, and max-pooling. They are performed independently and then concatenated channel-wise. To reduce the dimension for 3×3 and 5×5 convolution, they are preceded by 1×1 convolutions with reduced filter numbers. Figure 2.2 shows the structure of an Inception module. To be noted, having several kernel sizes within one Inception module means it has the ability to embed multi-scale information in one layer. The experiment results also showed favorable performance while Inception also enjoys the advantage of computation efficiency (with 1×1 convolution for dimension reduction). Compared with VGGNet, Inception is deeper (more convolutional layers) and wider (larger kernels).

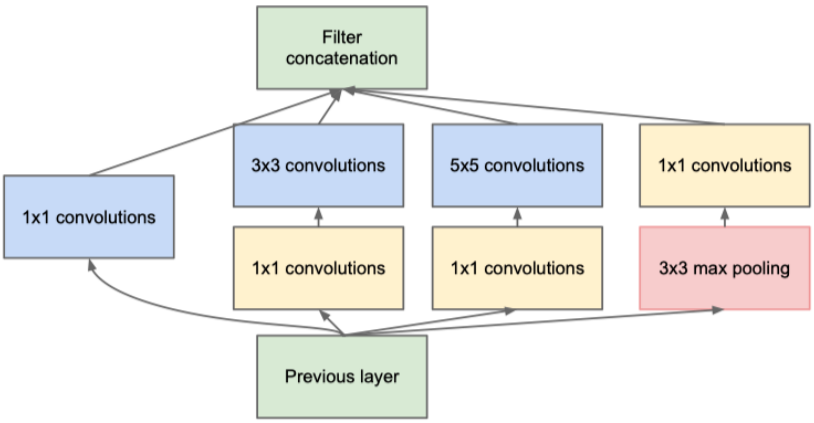

Figure 2.2: Illustration of an Inception [2] module. It has a four-stream structure and the results of each are concatenated to form the output.

The next monumental architecture is a milestone for even deeper CNNs. ResNet [3, 31] has succeeded in extending CNNs to up to 1000 layers deep by introducing the so-called "residual connection" or "skip connection". The main difficulty of making neural networks deeper is that they become much harder to train as the depth grows. One source of this problem is the vanishing/exploding gradients, which makes the models harder to converge. But it has been largely mitigated by normalization techniques, which allow models with tens of layers to be trained. However, even when these deep models converge, they would have degraded performance. Given experiment results [3], this degradation is not caused by overfitting because it also happens in the training set. If a deep network could copy the layers from a shallower network and learn to do identity mapping after these layers, it should perform no worse than the shallower one. However, experiments show that the optimizers failed to learn identity mapping. Inspired by this, the authors proposed to learn residual functions instead of the function itself. In other words, instead of learning a complex non-linear function $f(x)$, we learn $g(x) := f(x) - x$. It is obvious that when $g(x) := 0$, which should be relatively easier to learn, $f(x)$ will become identity mapping. In a neural network, this can be simply implemented by adding a "skip/shortcut connection" from the input of any parametric layer to its output. When the dimensions of the input and output are different, we can also apply linear transformation in the shortcut connection to match the dimension. Figure 2.3 is a simple illustration of such connections. The resulting CNNs are referred to as ResNets.

The architecture of ResNet consists of several "stages". Before the first stage, the input image is first passed to a convolutional layer with a large convolutional kernel (7×7). In each stage, 3×3 convolutions with stride 1 and a fixed number of filters are used, such that the dimension of input and output feature maps are the same. Shortcuts are made for every two convolutional layers. The first layer of each stage (except for the first one) has a stride of two and twice the number of filters of the previous stage. In this way, the resolution of feature maps of each stage is only half of the previous one and the channels double that of the previous stage. After the last

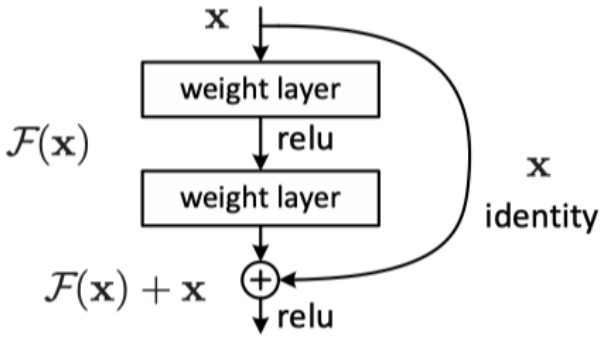

Figure 2.3: Shortcut/skip connection [3].

stage, an average pooling and a fully connected layer with softmax normalization are used to form the final output (classification on ImageNet). ResNet indeed demonstrated better performance than its plain counterpart especially when the depth goes higher. The classification accuracy of ResNet has once again broken the record on ImageNet. More importantly, the simple but effective idea of shortcut connection showed us the potential of designing much deeper networks than before and the possibility of much higher performance.

To further reduce the computation complexity of ResNets and inspired by Inception, ResNeXt has been proposed in [31]. Despite the numerous variants of Inception, the core remains to be the split-transform-merge strategy. In an Inception module, an input is split into lower-dimensional features and processed individually, and merged together afterward. ResNeXt uses this idea and proposes to split a high-dimensional feature map into groups as shown in Figure 2.4. The resulting model has the approximately same number of parameters as its ResNet counterpart but has compelling performance and much lower computation complexity.

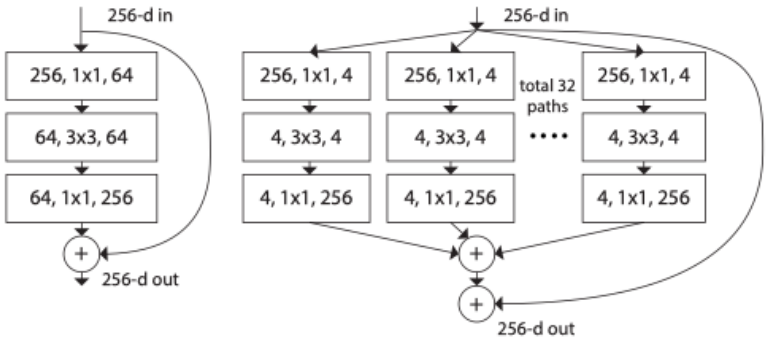

Figure 2.4: Left: A block of ResNet. Right: A block of ResNeXt with 32 groups. They have roughly the same number of parameters. A layer is shown as (# in-channels, filter size, # out-channels).

2.2 Learning Temporal Dependencies from Sequences

Recurrent Neural Networks (RNNs) are a type of neural network for processing sequences and are a common method for processing and understanding the temporal relationships in the Natural Language Processing (NLP) field. Starting from the most basic fully recurrent neural network, researchers in this field have developed various RNN architectures, the most used ones of which are Long-short Term Memory (LSTM) [32] and Gated Recurrent Unit (GRU) [22]. Despite their differences, they all have a common high-level structure. An RNN process a sequence recursively. In the beginning, it initializes a vector called a hidden state for storing all historical information. At each time step, it takes the latest input in the sequence to update the hidden state. In this way, all past information would be stored in the hidden state vector. Below we briefly introduce the three types of mainstream RNN architectures.

Elman networks. This type of RNN is also known as Simple Recurrent Networks (SRN). The update equation of Elman networks can be expressed as

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h). \tag{2.1}$$

where σ is an arbitrary activation function.

Long-short term memory. This RNN was proposed in [32] which is designed to let the neural network has long- and short-term memories. It consists of a cell, an input gate, an output gate, and a forget gate. The cell is responsible for remembering information for arbitrary time steps while the other gates control the information that flows in and out of the cell.

Gated Recurrent Unit, like LSTM, also has a forget gate but does not have an output gate. The performance of GRUs was found to be similar to LSTM on certain tasks such as speech recognition [33].

3D convolution. Meanwhile, deriving from the traditional signal processing methodology, convolution is another way of processing time sequences. Operations like smoothing, delaying, and predicting can be done through a convolutional filter. Analogous to CNNs, we can also use learned convolutional filters to process time sequences, where we perform the operation in the time dimension. When applied to time sequences of visual data (e.g., videos), the resulting operation is called 3D convolutions (`Conv3D`)[34, 35, 36].

2.3 Attention Mechanism

Attention mechanism [37] was inspired by the human brain's attention mechanism, where one only focuses on a certain (usually small) area on large visual input. Essentially, the attention mechanism does two things, compare and aggregate. It is able to compare two sets of vectors called queries and keys and give a matching score of the two. Next, it performs a weighted sum of the so-called values based on the matching score.

In [20], Transformer, an attention-based network that did not use recurrent layers nor convolutions, was proposed and showed superior performance in two machine translation tasks. Transformers were also extended for static images reconstruction [38]

and video recognition [39]. However, the core component of transformers, the attention mechanism also plays an important role in the MNNs that we are going to introduce, attention has become a central part that selects which modules to use to apply.

2.4 Modular Networks

Recently, MNNs [13, 14, 21, 40], a new class of deep learning architectures, have been proposed as multi-purpose architectures that can handle a wide range of downstream tasks. Such architectures are composed of an initial encoder that processes perceptual inputs (e.g., images) and a modular transition model (MTM) that operates on a few conceptual variables that resemble the human consciousness system that works on high-level concepts [41]. In particular, RIMs [21] is a typical instance of such MNNs, which lays the foundation for a series of follow-up works [13, 14]. In these two works, the aforementioned conceptual variables are considered as objects; therefore, the model is trained to learn interactions amongst objects. However, the potential of adapting RIMs to an object-centric model is still not fully explored yet. Recently, [26] proposed VIMs, a variational alternative to RIM, where they also experimented with slot attention and spatial broadcast decoder.

2.5 Object-Centric Learning

Object-centric representation learning is built upon the assumption that modeling natural scenes as compositions of objects and their relationships is better than using distributed representations [15]. Such representation is often formulated as a set of transformations from data in the image domain to a set of vectors in the latent space, each corresponding to an individual object. This transformation can be learned without supervision in various ways. For instance, MONet [16] models such transformations using a variational autoencoder [42], leveraging a recurrent attention mechanism that outputs objects masks. While approaching the problem using a component-wise encoder and a spatial mixture model to relate the encoded latent variables, GENESIS [18] used an autoregressive prior to capture the relationships between different components. In contrast, IODINE [43] also models the scene using a spatial mixture model but decodes the latent variables using a component-wise SBD [27]. On the other hand, SPACE [25] explicitly provides factorized object representation for both foreground object and background segments, defining an evidence lower bound (ELBO) that takes into account both foreground and background variables. Finally, Slot Attention (SA) [24] computes slots (i.e., object representations) using a recursive attention mechanism that iteratively maps the encoded input into a set of slots which are then processed by a GRU. Despite being a simple module, slot attention is shown to effectively extract object-centric features. Although these models share a few common points with our method, none of them can handle videos or data sequences, making them unable to capture object dynamics or handle downstream tasks such as video prediction.

2.6 Video Prediction

Video prediction, a fundamental function of the human brain to predict near-future visual scenes [44, 45], has received great interest in the computer vision community in recent years [23, 46]. Essentially, this is because video prediction requires learning representation of spatio-temporal relationships, which play a central role in numerous applications and studies. Direct video prediction algorithms have been applied in robotics [47], autonomous driving [48], etc. It can also serve as a generative pre-training strategy for video representation learning in other downstream tasks since videos provide a great amount of visual information [35], especially action recognition [49] and activity early recognition [50]. Besides predicting the complete future frames, we can only extract and predict some abstract lower-dimensional information from videos, such as human pose [51] and [52]. But the problem of future frame prediction has always been the most challenging one, because of the high-dimension nature of the prediction target.

We have witnessed extensive study and progress in the field of future frame prediction. The major progress in this field is in how to represent images and how to design the model architecture to process spatial and temporal information. Earlier works tend to borrow ideas from Natural Language Processing (NLP), where the processing target is sequences of vectors, analogous to video prediction. [53] proposed a Video Language Modelling algorithm to quantize frame patches into one-hot vectors. It first divides a complete frame into smaller patches, such that in each patch there are fewer variations. Then quantize each 8×8 8-bit image patch to a 10000-dim one-hot vector. These quantized image patches are then processed by a recurrent layer to get predictions. To consider the spatial dependencies between local patches, it is proposed to use a convolution layer on each patch and its neighbors before feeding to the recurrent layers. [23] continued on the idea of using image patches to encode frames but also proposed an alternative to operate on image percept, i.e., using a vector to encode an image with a convolutional encoder. It also proposed an autoencoder-like architecture, consisting of an encoding LSTM that incorporates all past information up to the current step and a decoding LSTM that predicts the next step conditioned on the current hidden states. While encoding and decoding, it also used convolutions as a core operation. Such convolution + RNN method has been pushed further by [54], where the authors extended LSTM by replacing the linear transformation on the input and hidden states with convolutions. In these two works, the encoder and decoder extract both the spatial and temporal dependencies. Alternative to this, an "encoder + transition model + decoder" architecture has been proposed to disentangle the spatial and temporal processing in the network. [55] used such an architecture and added action as external input to the transition model to achieve action-conditioned future frame generation. This architecture gradually became the most popular choice in recent years. The models such as CrevNet [35] and SimpVP [56] both adopted this structure and achieved state-of-the-art performance on several datasets.

One of the core problems in video prediction is to use what kind of operation to characterize spatio-temporal relationships. In summary, there are three types of methods.

1. Stack of RNNs with convolutional layers [23, 53, 57]
2. 3D convolution [34, 35, 36]
3. Pure CNN [56]

The first kinds are able to process variant-length sequences because RNNs have a recursive structure and convolution is time-invariant. For pure CNN transition model [56], it cannot process flexible length sequence directly since it has a multi-step-to-multi-step structure.

Perpendicular to the advance in high-level structure, several works have focused on designing models to get sharp predictions. These designs can be roughly divided into three aspects. (i) Multi-scale architecture, (ii) resolution preserving blocks, (iii) training tricks. Inspired by Laplacian Pyramid in the image processing field, [58, 59, 60] have proposed multi-scale image/video generation models. In particular, [58] used a Laplacian pyramid framework composed of convolutional layers and generated images with Generative Adversarial Networks (GANs). Extending such work to the video domain, [59] proposed similar models for future frame generation. This work has also pointed out that GANs can prevent the networks from generating blurry predictions when there exist multiple modes in the next frame. [60] proposed a model to generate possible future frames conditioned on only one static image.

Current models [53, 61] tend to adopt a convolutional encoder and decoder to capture the spatial dependencies while using either an autoregressive model [62], a recursive layers such as gated recurrent unit (GRU) [22, 42, 63, 64], or transformers [20, 65] to learn the temporal patterns. The focus of recent works like [35] has been resolution-preserving block design in order to achieve sharp prediction, or loss function design [66]. Meanwhile, since objects are basic components of a video, the object-centric perspective has also been explored by a few works [17, 26, 46, 47, 67]. Such as in [17], the authors used an attention mechanism to capture object-centric features, but they did not explicitly consider learning object-centric representations. However, object-centric representation learning still has not been fully studied in video prediction literature.

Methodology

3

In this chapter, we are going through the design details of RIMs and their extension: `OC-RIMs`. We start with a high-level architecture of RIMs tailored toward video prediction as in Figure 3.1. Overall, the network takes one frame at each time step and outputs a prediction for the next step. This is done by a forward pass through three parts:

1. An `Encoder` that converts an image to a (set of) vector(s) that can represent the input frame,
2. A **recurrent module**, in this case, RIMs, that convert the input vector(s) to another (set of) vectors that represent the next frame,
3. A `Decoder` that decodes the vector(s) produced by the recurrent module into an image, i.e., the next frame prediction.

3.1 Convolutional Encoder

We use a convolutional encoder to extract visual information from each frame. The input tensor is of shape $C \times H \times W$, where C is channel size, H is height and W is width. We call the tensors in a convolutional neural network features maps. Abstractly, for the input feature map T^{in} of size $c_{in} \times h_{in} \times w_{in}$. Each convolutional layer consists of multiple filters, each of which has a kernel of size $c_{in} \times d_f \times d_f$ (assuming d_f is an odd number). The output feature map T^{out} can be represented as

$$T^{out}_{k,m,n} = \sum_{i}^{d_f} \sum_{j}^{d_f} T^{in}_{:,m+\frac{d_f-1}{2}+i,n+\frac{d_f-1}{2}+j} \cdot w^{k}_{:,i,j}, \tag{3.1}$$

where w^k denotes the k-th filter kernel and $\cdot$ denotes the dot product between vectors. After convolutions, the resulting feature maps are usually passed to a non-linear activation function σ:

$$\tilde{T}^{out} = \sigma(T^{out}), \tag{3.2}$$

where σ is performed element-wise. We use a common choice for σ called Rectified Linear Unit (ReLU) activation function

$$\sigma(x) = \texttt{ReLU}(x) = \begin{cases} x, & x > 0 \\ 0, & x <= 0 \end{cases} \tag{3.3}$$

where x is a real-valued scalar. Feature maps after convolution and activation are passed to pooling layers. Again, for an input feature map T^{in} of size $c_{in} \times h_{in} \times w_{in}$,

the output T^{out} can be represented as

$$T^{out}_{k,m,n} = \texttt{pool}(T^{in}_{:,m+\frac{l-1}{2}:m+\frac{l-1}{2}+l,n+\frac{l-1}{2}:n+\frac{l-1}{2}+l}), \tag{3.4}$$

where l is a small number defining the neighborhood size around location (m, n) and `pool` can be arbitrary function that aggregates multiple values to one such as `max` (taking the maximum, max pooling) or `avg` (taking the average, average pooling). We stack multiple convolutional layers (including activations) and pooling layers together and gradually reduce the height and width dimensions. At the end of a convolutional encoder, we flatten the feature map into a vector and pass it to a few fully connected layers. Overall, we have

$$x = \texttt{Encoder}(I), \tag{3.5}$$

where I is an image of size $C \times H \times W$ and x is a vector of size d_{in} that embeds this image.

3.2 Gated Recurrent Unit

A recurrent neural network has a recursive structure that takes an input x_t at time step t and updates its internal hidden state h_{t-1} to h_t. In our case, we use GRU, where the recursive process can be described as

$$z_t = \sigma_g(W_{zx}x_t + W_{zh}h_{t-1} + b_z) \tag{3.6}$$
$$r_t = \sigma_g(W_{rx}x_t + W_{rh}h_{t-1} + b_r) \tag{3.7}$$
$$\tilde{h}_t = \sigma_h(W_{hx}x_t + W_{hh}(r_t \circ h_{t-1}) + b_h) \tag{3.8}$$
$$h_t = z_t \circ \tilde{h}_t + (1 - z_t) \circ h_{t-1}, \tag{3.9}$$

where z_t is referred to as the update gate, and r_t is the reset gate. σ_g is the sigmoid function and σ_h is the hyperbolic tangent function as

$$\sigma_g(x) = \frac{1}{1 + e^{-x}} \tag{3.10}$$
$$\sigma_h(x) = \tanh(x). \tag{3.11}$$

For the initial step $t = 0$, we set $h_0 = \mathbf{0}$, where $\mathbf{0}$ is a full-zero vector.

3.3 Recurrent Independent Mechanisms (RIMs)

In this section, we are going through the design details of RIMs. RIMs are a recurrent neural network architecture in which multiple parallel recurrent cells operate with nearly independent transition dynamics [21]. The architecture is composed of K cells (or modules), each containing a recurrent neural unit (such as GRU [22]) with its own hidden state vector: $\boldsymbol{h}_{t,k}$. When applied to video prediction, the overall architecture (as is shown in Figure 3.1,) consists of a convolutional encoder `Encoder(.)`, RIMs as a transition model, and a convolutional decoder. At each time step t, a frame $\boldsymbol{I}_t$, is first encoded to a vector as

$$\boldsymbol{x}_t = \texttt{Encoder}(\boldsymbol{I}_t). \tag{3.12}$$

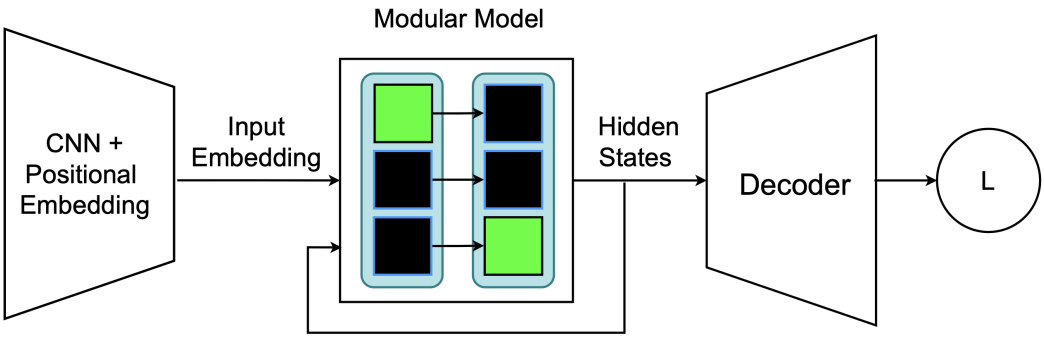

Figure 3.1: A modular network architecture tailored towards video prediction.

The input matrix $\boldsymbol{X}$ is then formed by concatenating $\boldsymbol{x}$ with a full-zero vector, as

$$\boldsymbol{X}_t^{\mathrm{T}} = \begin{bmatrix} \boldsymbol{x}_t^{\mathrm{T}} & \boldsymbol{0}^{\mathrm{T}} \end{bmatrix}. \tag{3.13}$$

The idea is to feed $\boldsymbol{x}_t$ to cells that need to be updated, and $\boldsymbol{0}$ to cells that do not need to be updated. We will refer to the full-zero vector as *null input*, as it is part of the input matrix and contains no information.

Input attention. The input attention module matches an input matrix $\boldsymbol{X}$ with the most relevant RIM cell via a key-value attention mechanism to generate per-cell GRU input:

$$\boldsymbol{a}_{t,k}^{(in)} = \texttt{softmax}\left(\frac{\boldsymbol{h}_{t,k}\boldsymbol{W}_k^q(\boldsymbol{X}\boldsymbol{W}^e)^T}{\sqrt{d_e}}\right)\boldsymbol{X}\boldsymbol{W}^v, \tag{3.14}$$

where $\boldsymbol{W}_k^q, \boldsymbol{W}^e, \boldsymbol{W}^v$ are the query, key, and value projection matrices respectively, and d_e is the dimension of keys.

Remarks. In the actual implementation, the set of matrices $\{\boldsymbol{W}_k^q\}_{k=1}^K$ are usually set to the same, which we denote as $\boldsymbol{W}^q$. We will explore this design choice in the following experiment chapter.

RIM cell activation. Activation is an important concept in RIMs. The motivation is that we only want to update those RIM cells that are relevant to the current input; the rest should remain unchanged. The activated RIM cells form the set $\mathcal{S}_t$ as

$$\mathcal{S}_t = \{k | r_{t,k} \text{ is the top-}k_a \text{ largest among all } k.\}, \tag{3.15}$$

where $r_{t,k}$ denotes a relevance score indicating how likely we should activate RIM cell k at time step t. This relevance score is calculated by normalizing the matching score between $h_{t,k}$ and the null input, as in

$$r_{t,k} = 1 - \texttt{softmax}\left(\frac{\boldsymbol{h}_{t,k}\boldsymbol{W}_k^q(\boldsymbol{X}\boldsymbol{W}^e)^T}{\sqrt{d_e}}\right)_{1,-1}, \tag{3.16}$$

where (1,-1) indexes the first row and last column.

Per-cell dynamics. The per-cell GRU inputs are passed to the transition model of each RIM cell as follows:

$$\tilde{\boldsymbol{h}}_{t,k} = \begin{cases} \texttt{GRU}_{\boldsymbol{\theta}_k}(\boldsymbol{h}_{t,k}, \boldsymbol{a}_{t,k}^{(in)}) & k \in \mathcal{S}_t \\ \boldsymbol{h}_{t,k} & k \notin \mathcal{S}_t \end{cases}, \tag{3.17}$$

where $\texttt{GRU}_{\boldsymbol{\theta}_k}$ is a GRU parameterized by $\boldsymbol{\theta}_k$.

Communication between RIMs. RIM cells operate independently by default. But to allow them for reading contextual information from each other, a communication block is designed for information sharing. Here the communication is modeled by an attention mechanism between updated

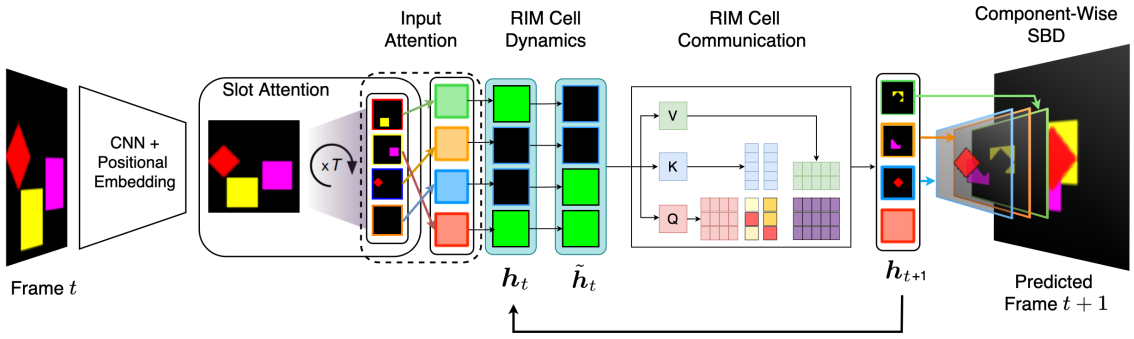

Figure 3.2: Illustration of `OC-RIMs` for video prediction. The original RIMs is extended by a slot attention module and a component-wise decoder.

hidden states. Using parameters $\theta_k^{(c)} = (\tilde{\boldsymbol{W}}_k^q, \tilde{\boldsymbol{W}}_k^e, \tilde{\boldsymbol{W}}_k^v)$, such communication block can be expressed as:

$$\boldsymbol{Q}_{t,k} = \tilde{\boldsymbol{W}}_k^q \tilde{\boldsymbol{h}}_{t,k}, \forall k \in \mathcal{S} \qquad \boldsymbol{K}_{t,k} = \tilde{\boldsymbol{W}}_k^e \tilde{\boldsymbol{h}}_{t,k}, \forall k \qquad \boldsymbol{V}_{t,k} = \tilde{\boldsymbol{W}}_k^v \tilde{\boldsymbol{h}}_{t,k}, \forall k \tag{3.18}$$

$$\boldsymbol{h}_{t+1,k} = \texttt{softmax}\left(\frac{\boldsymbol{Q}_{t,k}\boldsymbol{K}_{t,:}^{\mathrm{T}}}{\sqrt{d_e}}\right)\boldsymbol{V}_{t,:} + \tilde{\boldsymbol{h}}_{t,k} \qquad \forall k \in \mathcal{S}.$$

Concatenation decoder. To make a prediction for the next frame, RIMs first concatenate the hidden states of all RIM cells, then decode the concatenated hidden state with an upsampling convolutional decoder. We can represent such a concatenation decoder as `CAT`,

$$\hat{\boldsymbol{I}}_{t+1} = \texttt{CAT}(\boldsymbol{h}_{t+1,1}, \ldots, \boldsymbol{h}_{t+1,K}). \tag{3.19}$$

3.4 Object-Centric RIMs

RIMs attention mechanism and modular structure are designed for selective utilization of the most relevant RIM cells depending on the input data. This selection largely depends on the input attention in (3.14) and (3.13). However, if we expand these two equations, we can get $\boldsymbol{a}_{t,k}^{in} = \lambda_{t,k}\boldsymbol{x}_t\boldsymbol{W}^v, \forall k$, where $\lambda_{t,k}$ is a scalar depending on the attention matching score. This means the inputs to all RIM cells are formulated from the same vector, but only scaled differently. We hypothesize that this is inefficient for RIMs. Inspired by recent studies in [13, 14], we feed the RIMs with object-centric features (i.e., slots) such that each RIM cell can selectively attend to one object. To this aim, as in [26] we extend RIMs' architecture with two extra components: (i) slot attention [24] on the encoder side to generate object-centric representations, and (ii) component-wise decoder [16] to make compositional predictions. We refer to our extended architecture as `OC-RIMs`. This is illustrated in Figure 3.2. The architectural differences between `OC-RIMs` and several related models are summarized in Table 3.1. In the following, we first explain these two extra modules in more detail; we then discuss the overall end-to-end architecture of `OC-RIM`.

Slot attention. It is a recursive attention mechanism that maps a feature map out of a convolutional encoder to a set of vectors (called slots). The feature map is first augmented with positional embedding and then spatially flattened to a few equivariant vectors as input to a recursive slot refinement process. During the refinement process, a scaled dot-product attention between the projected input vectors (i.e., keys and values) and the projected slot vectors (i.e., queries) is used to aggregate spatial inputs. The matching score is then normalized over slots in order to introduce competition between the slots. Such competition enforces different slots to specialize and bind to

Model	Input	Per-frame Slot Refinement	Dynamics	Decoder
GRU [22]	Image Embedding	✗	Monolithic GRU	`CAT`
RIMs [21]	Image Embedding or Feature Map	✗	Modular GRU	`CAT`
VIMs [26]	Feature Map	✗	Modular MLP and LayerNorm	`SBD`
`OC-RIMs` (**ours**)	Slots	✓	Modular GRU	`SBD/BCD`

Table 3.1: Comparisons of the architectures of `OC-RIMs` and related baselines. Out of the four models above, only VIMs is a variational model. LayerNorm refers to the technique proposed in [4].

distinct objects in the input. Finally, the slots are updated through a recurrent layer such as GRU [24]. A similar approach is adopted in VIMs [26], where the input attention is adjusted such that the hidden states are used as slots and updated in a similar fashion as slot attention. In VIMs, there is only one GRU unit that updates once every time step. On the other hand, we use two GRU units, one for slot refinement that iterates multiple times for each frame, and one for the temporal prediction that updates once per time step.

Component-wise decoder. In the original RIMs architecture, [21], the hidden states of each RIM cell are first concatenated as one, and then decoded altogether. However, in the context of scene decomposition (such as in MONet [16]), where multiple latent embeddings for one image are extracted, the latent vectors are decoded separately using a shared decoder. In this setting, each latent vector describes a component within the image just like objects in our object-centric perspective. A recent proposition for such a decoder is SBD, which shows superior reconstruction performance on several datasets [27]. We, thus, employ a component-wise SBD for the proposed network. In order to understand whether the performance improvement depends on the broadcasting operation or the component-wise decoding, we also consider a basic component-wise upsampling convolutional decoder (BCD) as an alternative decoder in `OC-RIMs`.

`OC-RIMs`: an end-to-end overview. As is depicted in Figure 3.2, at each time step, the input frame is first passed through a CNN encoder (`Encoder(.)`) to generate a feature map. We then flatten it and augment it with positional embedding. This is then fed into a slot attention unit where multiple refined slots are created through a recursive process. Next, these slots are fed into RIMs through the input attention process, where each RIM cell selectively attends to these slots by matching between its hidden state and the slots as follows:

$$\boldsymbol{V}_t = \texttt{SlotAttention}(\texttt{Encoder}(\boldsymbol{I}_t)), \tag{3.20}$$

$$\boldsymbol{a}_{t,k}^{(in)} = \texttt{softmax}\left(\frac{\boldsymbol{h}_{t,k}\boldsymbol{W}_k^q(\boldsymbol{V}_t\boldsymbol{W}^e)^T}{\sqrt{d_e}}\right)\boldsymbol{V}_t\boldsymbol{W}^v, \tag{3.21}$$

where each row V is a slot produced by slot attention. Each RIM cell then updates itself and passes its hidden state to the component-wise decoder, resulting in multiple cell-wise future frame predictions and alpha masks (i.e., 100% alpha when fully opaque and 0% alpha when fully transparent). Finally, we fuse all these cell-wise predictions with normalized alpha masks as the prediction

$$\hat{\boldsymbol{i}}_{t+1,k},\ \hat{\boldsymbol{m}}_{t+1,k} = \texttt{Decoder}(\boldsymbol{h}_{t+1,k}), \qquad k = 1, ..., K, \tag{3.22}$$

$$\hat{\boldsymbol{I}}_{t+1} = \sum_{k=1}^{K} \hat{\boldsymbol{i}}_{t+1,k} \circ \hat{\boldsymbol{m}}_{t+1,k}. \tag{3.23}$$

3.5 SCOFF: Permutation Equivariant RIMs

If we observe the formation of `OC-RIMs`, we can see one intrinsic weakness of it is that the number of cells is fixed. Since we are expecting each RIM cell to bind to one object, it becomes tricky when we have more objects than RIM cells. In [14], SCOFF model was proposed based on RIMs. SCOFF also has a set of hidden states (called object files, like each cell's hidden state) and a set of recurrent units (called schemata, like each cell's recurrent unit). The difference is, however, that the recurrent units are shared among all object files. For example, one object file can be updated by the first recurrent unit, or the second, depending on their matching. This shows the possibility of having a varying number of objects. If we have more objects than object files, we only need to create a new object to accommodate the scenario without changing the schemata since they need to be trained.

The schema selection mechanism of SCOFF is as follows

$$\bar{\boldsymbol{h}}_{t,k,j} = \mathrm{GRU}_{\boldsymbol{\theta}_j}(\boldsymbol{a}_{t,k}^{(in)}, \boldsymbol{h}_{t-1,k}), \quad \forall k \in \{1, ..., K\}, j \in \{1..., n_s\} \tag{3.24}$$

$$\bar{\boldsymbol{q}}_{t,k} = \boldsymbol{h}_{t-1,k}\bar{\boldsymbol{W}}^q \tag{3.25}$$

$$\bar{\boldsymbol{\kappa}}_{t,k,j} = \bar{\boldsymbol{h}}_{t,k,j}\bar{\boldsymbol{W}}^e \tag{3.26}$$

$$j^*_{t,k} = \mathrm{argmax}_j(\bar{\boldsymbol{q}}_{t,k}^{\mathrm{T}}\bar{\boldsymbol{\kappa}}_{t,k,j} + \gamma), \quad \gamma \sim \mathrm{Gumbel}(0, 1) \tag{3.27}$$

$$\boldsymbol{h}_{t,k} = \bar{\boldsymbol{h}}_{t,k,j^*_k} \tag{3.28}$$

where n_s is the number of schemata and $\bar{\boldsymbol{W}}^q$ and $\bar{\boldsymbol{W}}^e$ are query projection matrix and key projection matrix for this schema attention mechanism. To put it simply, the selection of schemata is done through a matching and Gumbel argmax. The matching is done through two steps. The first is to update the current hidden state using all possible schemata, resulting in a set of update candidates. Then we match the current hidden state with the candidates through an attention mechanism and use the most well-matched candidate as our final update.

We can extend SCOFF in the same way as we extend RIMs. In the next chapter, we will show the results of object-centric SCOFF for comparison.

3.6 Training with Adam

In order to train our model, we need to formulate the problem as an optimization problem and attempt to solve it. We denote our network as a function

$$\hat{\boldsymbol{I}}_{i,t+1} = f(\hat{\boldsymbol{I}}_{i,t}; \boldsymbol{\theta}), \tag{3.29}$$

where the index i denotes the i-th sample (video clip), and the index t denotes the t-th time step. Assume there are N samples in our dataset D and there are T frames in each sample. The optimization problem can be formulated as follows

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\hat{\boldsymbol{I}}_{i,:} \sim \mathcal{D}} \left[\sum_{t=1}^{T-1} L\left(f(\hat{\boldsymbol{I}}_{i,t}; \boldsymbol{\theta}), \hat{\boldsymbol{I}}_{i,t+1} \right) \right], \tag{3.30}$$

where $\mathcal{D}$ is the distribution of our data and L is a loss function that calculates the sum of element-wise error, for example, the mean square error function; it can either normalize over the number of elements or not. Since our network f is non-convex, we cannot apply convex optimization techniques here to get the exact minimum but can only get local minimums with iterative approximations. Since our objective is differentiable, i.e., we can easily calculate the first-order partial derivatives (gradients) of our objective, we can efficiently optimize it using gradient descent. Gradient descent updates an estimate of the optimal parameter at each iteration using

$$\boldsymbol{\theta}^{(j+1)} = \boldsymbol{\theta}^{(j)} - c^{(j)}\boldsymbol{g}^{(j)}, \tag{3.31}$$

$$\boldsymbol{g}^{(j)} = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}}, \tag{3.32}$$

where $c^{(j)}$ is the step size at j-th iteration and $\boldsymbol{g}^{(j)}$ is the gradient of the objective function $\mathcal{L}(\boldsymbol{\theta})$ w.r.t. the parameter $\boldsymbol{\theta}$ when $\boldsymbol{\theta} = \boldsymbol{\theta}^{(j)}$.

Unfortunately, we do not know the true distribution of the data. Therefore, we cannot use the update equation (3.31) because we cannot calculate the gradient $\boldsymbol{g}^{(j)}$. However, when our dataset is large enough, we can approximate the expectation using the sample mean over all of our samples. The optimization problem thus becomes

$$\min_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T-1} L\left(f(\hat{\boldsymbol{I}}_{i,t}; \boldsymbol{\theta}), \hat{\boldsymbol{I}}_{i,t+1}\right). \tag{3.33}$$

We, therefore, update our parameters by

$$\boldsymbol{\theta}^{(j+1)} = \boldsymbol{\theta}^{(j)} - c^{(j)} \hat{\boldsymbol{g}}^{(j)}, \tag{3.34}$$

$$\hat{\boldsymbol{g}}^{(j)} = \nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}}, \tag{3.35}$$

where $\hat{\boldsymbol{g}}^{(j)}$ can be seen as an estimate of the true gradient $\boldsymbol{g}^{(j)}$. It is an unbiased estimate of the true gradient. Assume all samples in D are identical and independently distributed (i.i.d.) according to the distribution $\mathcal{D}$, we have

$$\begin{aligned}
\mathbb{E}\left[\hat{\boldsymbol{g}}^{(j)}\right] &= \mathbb{E}_D\left[\nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}}\right] \\
&= \nabla_{\boldsymbol{\theta}} \mathbb{E}_D\left[\frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T-1} L\left(f(\hat{\boldsymbol{I}}_{i,t}; \boldsymbol{\theta}), \hat{\boldsymbol{I}}_{i,t+1}\right)\right]\Bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}} \\
&= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\hat{\boldsymbol{I}}_{i,:} \sim \mathcal{D}}\left[\sum_{t=1}^{T-1} L\left(f(\hat{\boldsymbol{I}}_{i,t}; \boldsymbol{\theta}), \hat{\boldsymbol{I}}_{i,t+1}\right)\right]\Bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}} \\
&= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\hat{\boldsymbol{I}}_{i,:} \sim \mathcal{D}}\left[\mathcal{L}(\boldsymbol{\theta})\right]|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}} \\
&= \boldsymbol{g}^{(j)}
\end{aligned} \tag{3.36}$$

The issue with this update method is that at each iteration we need to estimate the gradient by calculating all the sample gradients of the whole dataset. This is clearly inefficient and expensive (in terms of the memory required for computation), especially during the first steps when we are still far away from the optimal parameter. We also see that The objective function is now stochastic. Moreover, our objective function consists of a sum of loss functions over different points of data. Therefore, the optimization can be made more efficient (especially in terms of the memory required for computation) by using stochastic gradient descent (SGD) as follows

$$\boldsymbol{\theta}^{(j+1)} = \boldsymbol{\theta}^{(j)} - c^{(j)} \hat{\boldsymbol{g}}^{(j)}_{\mathrm{SGD}}, \tag{3.37}$$

$$\hat{\boldsymbol{g}}^{(j)}_{\mathrm{SGD}} = \nabla_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_v(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}}, \tag{3.38}$$

and

$$\tilde{\mathcal{L}}_v(\boldsymbol{\theta}) = \frac{1}{|D^v|} \sum_{i \in D^v} \sum_{t=1}^{T-1} L\left(f(\hat{\boldsymbol{I}}_{i,t}; \boldsymbol{\theta}), \hat{\boldsymbol{I}}_{i,t+1}\right), \tag{3.39}$$

where D^v is a subset of the complete dataset D. More specifically, we divide D into V non-overlapping subsets. We call each one of these subsets a batch (or a mini-batch); the v-th batch is denoted by D^v. We often refer to the cardinality of a batch $|D^v|$ as batch size. Assume $|D^v|$ is the same for all v, then we can denote the batch size as M for simplicity. At each step j, we use a different batch D^v for inference and gradient calculation. Every time we finish iterating all batches, we say we have finished an epoch. A full training procedure, i.e., the training loss nearly converging to a constant,

usually consists of many epochs depending on the task. Since which batch D^v we are using completely depends on iteration step t, we thus replace the index v with j, and denote $\tilde{\mathcal{L}}_v(\boldsymbol{\theta})$ as $\tilde{\mathcal{L}}^{(j)}(\boldsymbol{\theta})$.

According to (3.38), $\hat{\boldsymbol{g}}_{\text{SGD}}^{(j)}$ is also an unbiased estimate of the true gradient. Again, assume all samples in D are identical and independently distributed (i.i.d.) according to the distribution $\mathcal{D}$, we have

$$\begin{aligned}
\mathbb{E}\left[\hat{\boldsymbol{g}}_{\text{SGD}}^{(j)}\right] &= \mathbb{E}_{D^v}\left[\nabla_{\boldsymbol{\theta}}\tilde{\mathcal{L}}^{(j)}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}}\right] \\
&= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{D^v}\left[\frac{1}{M}\sum_{i=1}^{M}\sum_{t=1}^{T-1} L\left(f(\hat{\boldsymbol{I}}_{i,t};\boldsymbol{\theta}),\hat{\boldsymbol{I}}_{i,t+1}\right)\right]\Bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}} \\
&= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{\hat{\boldsymbol{I}}_{i,:}\sim\mathcal{D}}\left[\sum_{t=1}^{T-1} L\left(f(\hat{\boldsymbol{I}}_{i,t};\boldsymbol{\theta}),\hat{\boldsymbol{I}}_{i,t+1}\right)\right]\Bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}} \\
&= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{\hat{\boldsymbol{I}}_{i,:}\sim\mathcal{D}}\left[\mathcal{L}(\boldsymbol{\theta})\right]|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}} \\
&= \boldsymbol{g}^{(j)}
\end{aligned} \tag{3.40}$$

Comparing (3.35) and (3.38), the only difference is their variances. It is easy to show that

$$\frac{\text{Var}\left[\hat{\boldsymbol{g}}^{(j)}\right]}{\text{Var}\left[\hat{\boldsymbol{g}}_{\text{SGD}}^{(j)}\right]} = \frac{M}{N}, \tag{3.41}$$

where M and N are the number of samples in D^v and D respectively. We skip the proof here for brevity. The batch size M here can normally be as small as 32 or even 16, while the dataset size N can be as large as 10^4 or 10^5. Therefore, when we use SGD, we normally would have a large variance with our gradient estimation, but we would save a great number of computation resources at each iteration. More importantly, calculations over the whole dataset might not even be possible due to the realistic limitation on memory.

Adam [68]. Another issue with the optimization is the high-dimensional nature of the parameter space in our problem. To deal with this issue, we adopt Adam [68] algorithm, an adapted version of SGD. The algorithm of Adam is shown in Algorithm 1. Intuitively, we can see that the difference between Adam and the vanilla SGD is that Adam uses the normalized mean of the gradient to update the parameter. It first estimates the mean and the uncentered variance of the gradient $\boldsymbol{m}^{(j)}$ and $\boldsymbol{v}^{(j)}$ using exponential moving average. Then it corrects the bias to get unbiased estimations $\hat{\boldsymbol{m}}^{(j)}$ and $\hat{\boldsymbol{v}}^{(j)}$. It finally uses a normalized mean of the gradient to update the parameter with a fixed step size α, resulting in the effective step being $\alpha\hat{\boldsymbol{m}}^{(j)}/(\sqrt{\hat{\boldsymbol{v}}^{(j)}}+\epsilon)$. The advantage of Adam includes being invariant to diagonal gradient scaling, computationally efficient, and so on. It has been empirically proven favorably comparable to or to outperform other stochastic optimization methods.

Algorithm 1: *Adam* [68]

Require: α: Stepsize

Require: $\beta_1, \beta_2 \in [0, 1)$: Decay rates for the exponential moving average estimate of the first and second order moments

Require: $\tilde{\mathcal{L}}^{(j)}(\boldsymbol{\theta})$: Stochastic loss function at each step j

Require: ϵ: A small number for numerical computation stability

Initialize: $\boldsymbol{\theta}_0$: Initial parameter

$\boldsymbol{m}_0 \leftarrow 0$: Initial first moment estimation

$\boldsymbol{v}_0 \leftarrow 0$: Initial second moment estimation

$j \leftarrow 0$: Initial step ;

while $\boldsymbol{\theta}^{(j)}$ *not converged* **do**

$t \leftarrow t + 1$;

$\boldsymbol{g}^{(j)} \leftarrow \nabla_{\boldsymbol{\theta}} \tilde{\mathcal{L}}^{(j)}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(j)}}$;

$\boldsymbol{m}^{(j)} \leftarrow \beta_1 \boldsymbol{m}^{(j-1)} + (1 - \beta_1)\boldsymbol{g}^{(j)}$;

$\boldsymbol{v}^{(j)} \leftarrow \beta_1 \boldsymbol{v}^{(j-1)} + (1 - \beta_1)(\boldsymbol{g}^{(j)} \circ \boldsymbol{g}^{(j)})$;

$\hat{\boldsymbol{m}}^{(j)} \leftarrow \boldsymbol{m}^{(j)}/(1 - \beta_1^j)$ (bias correction) ;

$\hat{\boldsymbol{v}}^{(j)} \leftarrow \boldsymbol{v}^{(j)}/(1 - \beta_2^j)$ (bias correction) ;

$\boldsymbol{\theta}^{(j)} \leftarrow \boldsymbol{\theta}^{(j-1)} - \alpha \hat{\boldsymbol{m}}^{(j)}/(\sqrt{\hat{\boldsymbol{v}}^{(j)}} + \epsilon)$

end

return $\boldsymbol{\theta}^{(j)}$: converged parameter

4 Experiments

In this section, we apply RIMs and `OC-RIMs` to video prediction tasks. By comparing with several related baselines, we show the significant performance improvement of `OC-RIMs`. We also provide qualitative and quantitative results to demonstrate `OC-RIMs`' ability to decompose the scene when trained in an unsupervised manner. After that, we evaluate the tracking consistency of RIM cells and objects with two new metrics we defined. We then make an inspection and discussion of the key components of `OC-RIMs`, i.e., input attention, RIM cell activation, slot attention, and communication attention.

4.1 Experiment Setup

4.1.1 Datasets

We test GRU, RIM baselines, and `OC-RIMs` on two video prediction datasets, **Moving MNIST** and **Moving Sprites**.

Moving MNIST. It is a dataset proposed in [23], where different digits appear in a random position and move towards a random direction with a constant speed. The digits will bounce back when they hit the boundary.

Moving Sprites. It is a dataset that we adapted from Sprites-MOT [28] for video prediction. This dataset contains 12800 video clips. Each video clip contains 20 frames of size $64 \times 64 \times 3$. These frames have a black background and at most three objects. Every object is generated randomly in the first two frames with a random scale, shape (circle/triangle/rectangle/diamond) and color (red/green/blue/yellow/magenta/cyan). All of them are given a constant velocity with a random direction. Objects might partially or completely move out of frame when they reach the boundary instead of bouncing back. The challenge of this dataset is that the objects can occlude each other and be partly out of frame, thus the model has to infer the depth and complete shape from the input. The goal of both tasks is to predict the future 10 frames conditioned on the past 10 frames.

4.1.2 Implementation Details

We use PyTorch [69] for all our experiments. The hyperparameters we used are listed in Table 4.1.

The structure of the convolutional encoder for RIMs baseline is shown in Table A.1. When we use slot attention, or when we feed feature maps directly to RIMs, we use the encoder below in Table A.2 to produce feature maps.

All the models in this section are trained end-to-end using teacher forcing [71]. For any video clip, we feed all the ground truth frames to get predictions and backpropagate

Hyperparameter	Value
Learning Rate	0.0007
Batch Size	32
Optimizer	Adam [70]
Epochs	400
Loss Function	MSE
Input Size	64
Input Key Size	64
Input Value Size	200
Input Dropout	0.1
Slot Size	64
Hidden State Size	100
Communication Key Size	32
Communication Dropout	0.1
# Active RIM Cells k_a	Equal to # RIM Cells

Table 4.1: Hyperparameters

the prediction loss. During the evaluation, we input the first 10 frames to the model and evaluate the prediction on the future 10 frames.

We use Adam [68] optimizer to train our models. Each model is trained for 400 epochs with a fixed learning rate for a fair comparison. Demonstrations of the training/test loss curve on both datasets are shown below. **Loss curve.** Below shows the training and loss curve with respect to epochs.

Figure 4.1: Demonstration of the training/test loss curve in one of our experiment runs on Moving MNIST. The blue curve is training loss and the red one test loss.

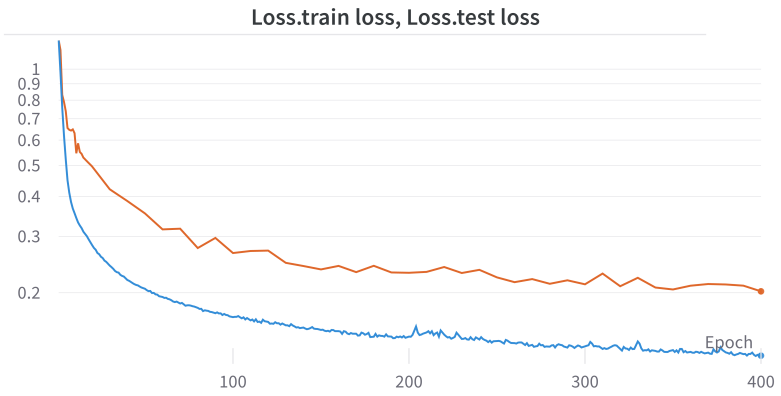

Figure 4.2: Demonstration of the training/test loss curve in one of our experiment runs on Moving Sprites. The blue curve is training loss and the red one test loss.

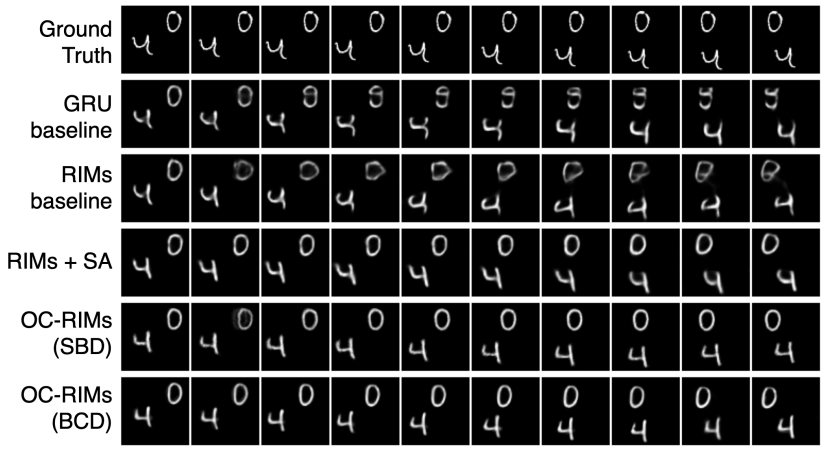

Figure 4.3: Predictions made by GRU and RIM baselines and `OC-RIMs` on Moving MNIST.

4.2 Prediction Performance

Figure 4.3 shows the predictions made by `OC-RIMs` and baselines. It is clear that with GRU and non-object-centric RIMs, the digits gradually deformed over time while RIM has less deformation than GRU. When combined with an object-centric feature extractor, i.e. slot attention, RIMs were able to reconstruct each digit much better and the deformation over time was largely reduced, even with a concatenation decoder. Moreover, when further extended with a component-wise decoder (BCD and SBD), `OC-RIMs` are able to reconstruct every single digit with higher sharpness.

We also use mean squared error (MSE) and structural similarity (SSIM) to quantitatively compare the prediction performance of each model. The results are shown in 4.2. Here RIMs baseline showed slightly worse performance compared with the GRU baseline. However, when we augmented it with slot attention, it immediately showed better performance than GRU. This suggests that structural inputs are crucially important for modular networks to function properly. Overall, `OC-RIMs` have achieved

the best results. The above results illustrated that object-centric modules like slot attention and component-wise decoder can greatly improve the performance of RIMs on video prediction tasks.

Model	Decoder	# Modules	MSE	SSIM
ConvLSTM [54]	-	-	103.3	0.7070
GRU [22]	CAT	-	89.49	0.8484
RIMs [21]	CAT	6	96.25	0.8312
OC-RIMs (**ours**)	BCD	6, 3	**73.88**	**0.8742**

Table 4.2: Prediction performance of different models on Moving MNIST. Style: **best** out of all model variants. # Modules refers to the number of modules (number of RIM cells and number of slots respectively). CAT refers to the concatenation decoder which is used in the original RIM.

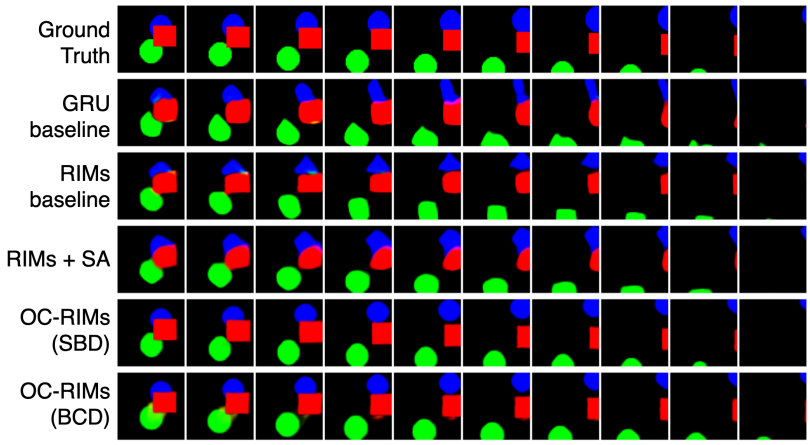

Figure 4.4: Predictions made by GRU and RIM baselines and OC-RIMs on Moving Sprites.

Model	Decoder	# Modules	MSE	SSIM
GRU [22]	CAT	-	85.58	0.8960
RIMs [21]	CAT	6	74.78	0.9057
OC-RIMs (**ours**)	SBD	6, 3	**39.60**	**0.9341**

Table 4.3: Prediction performance of different models on Moving Sprites. Style: best out of all model variants.

Similar models have been trained on Moving Sprites. Ground truth and predictions made by different models are shown in Figure 4.4. We observed a similar phenomenon as in Moving MNIST, where objects experienced severe deformation over time with GRU and RIMs. When augmented with slot attention, RIMs were still not able to correctly separate objects. Finally, OC-RIMs predict significantly better object reconstruction.

Numerical results are shown in Table 4.3. In this case, the GRU baseline showed much higher MSE than RIMs, which might suggest that the impact of modularity

inductive bias might depend heavily on the dataset. Meanwhile, with slot attention alone, RIMs failed to achieve better results but instead had a larger MSE. This is reflected in Figure 4.4 where RIMs+SA still predicts deformed objects. However, after we adopted both SA and component-wise decoders in `OC-RIMs`, the prediction performance improved significantly. The predicted objects were no longer deformed. This suggests that using an object-centric feature extractor like slot attention might not always guarantee the model will learn to separate objects, nor will the performance always improve. But when also combined with a component-wise decoder, the model could more easily learn object-centric representations and achieve notably better performance.

Our experiments with `OC-RIMs` on Moving MNIST and Moving Sprites both demonstrated that RIMs benefits greatly from object-centric modules and features, namely slot attention and component-wise decoders. Although the choice of BCD and SBD still depends on the specific dataset, `OC-RIMs` always yield the best performance compared with the other baselines. Meanwhile, each of the above modifications alone can almost always improve the performance of RIMs.

4.3 Decomposition Ability

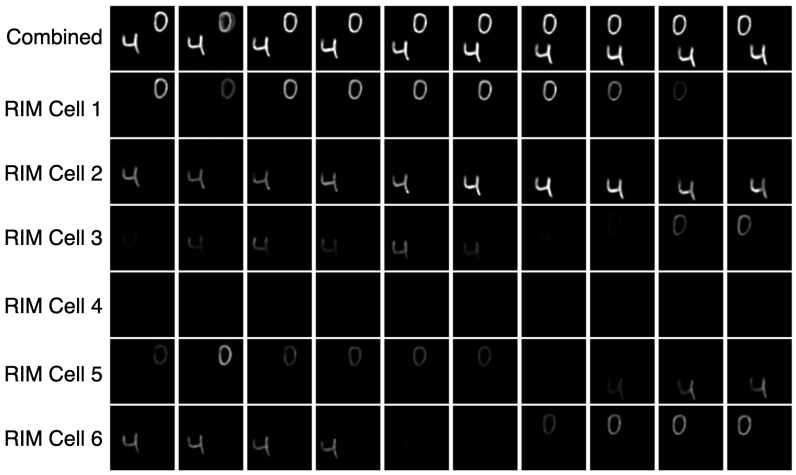

Figure 4.5: Individual Predictions made by each RIM cell within `OC-RIMs`.

Next, we show the individual predictions made by each RIM cell to have a clearer idea about what each RIM cell is processing. In Figure 4.5, we can see the specialized output from each RIM cell along with the combined prediction. It is seen that each RIM cell is mostly bound to one object (digit). We can argue that such factorization of the scene into different objects facilitates the prediction task since each object can be processed separately instead of altogether using a monolithic system.

In Figure 4.6 and Figure 4.7, we observe that each RIM cell attends to one object at any specific moment in time. More importantly, even when there is occlusion between objects, `OC-RIMs` are able to infer the complete un-occluded object. This suggests that `OC-RIMs` indeed learned object-centric representations instead of distributed features of

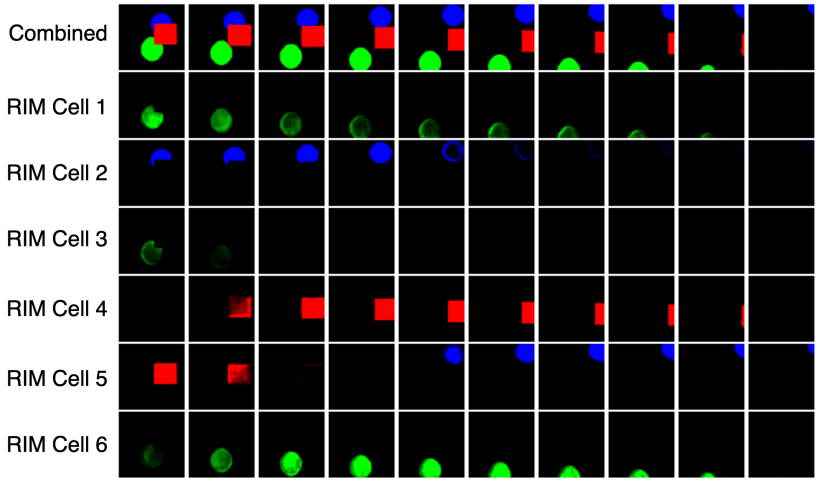

Figure 4.6: Predictions of each single RIM cell within `OC-RIMs` (with alpha mask)

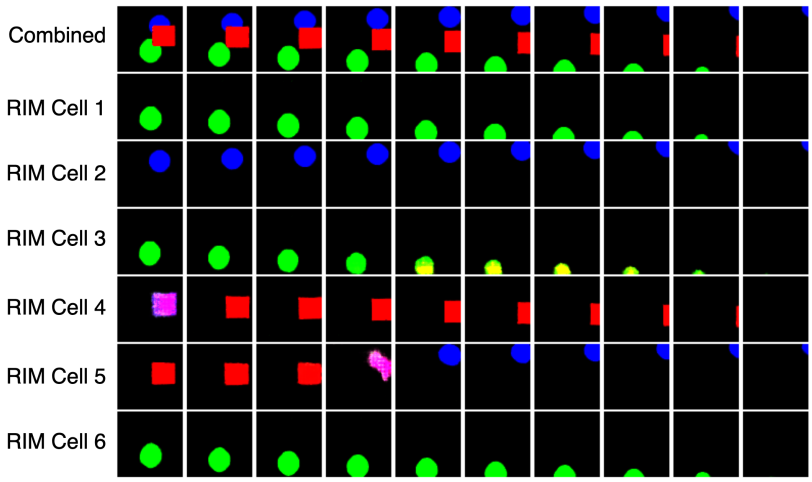

Figure 4.7: Predictions of each single RIM cell within `OC-RIMs` (without alpha mask).

the whole image, and is able to understand the underlying depth factor in the image.

From the three figures above, we can also observe that there exist repetitive objects. For example, in Figure 4.5, the first two RIM cells of out all six are mainly responsible for predicting digits '0' and '4'. But we can also see the last two RIM cells predicting '0' and '4' at the same time, but with lower brightness. It is most likely the last two RIM cells are making almost the same predictions but are just masked by the alpha mask predicted by the decoder. This phenomenon also happens in Moving Sprites dataset. Comparing Figure 4.6 and Figure 4.7, we see that RIM cells 4 and 5 sometimes make bad predictions (RIM cell 4 at the first frame and RIM cell 5 at the fourth frame), but they are masked by the alpha mask. This suggests that when we have more cells than objects, instead of letting one cell handle one object, `OC-RIMs` tend to let multiple cells split the prediction work and filter out the bad predictions in the last layer. It is difficult to tell whether this is more beneficial or detrimental. On the one hand, using multiple

networks/cells to handle the same task can be seen as a kind of ensemble, which could improve performance. On the other hand, the RIM cells have less independence and lose their composability since they always depend on other cells to work together.

Another possible source for repetitive prediction is input attention. In input attention, we apply two normalizations as in slot attention. First, we normalize over RIM cells, and then, we normalize over slots. The idea is that we first let each RIM cell compute to get different slots, and then give a normalized instead of a down-scaled slot to each RIM cell. This means two RIM cells can get the same slot as input even if one of them has a much higher matching score with that slot. In future work, one could try to disable the second normalization.

4.4 Tracking Consistency

From the individual predictions shown in Figure 4.5 and Figure 4.7, we can RIM cells do not always consistently track one object. For example, RIM cell 6 in Figure 4.5 switched from tacking the digit "4" to digit "0", and RIM cell 5 in Figure 4.7 switched from the red square to the blue circle. However, we naturally would like each RIM cell to bind to one fixed object instead of keeping swapping. In this section, we will first develop metrics to measure the consistency of RIMs object tracking and furthermore investigate why such switching will happen and how to possibly mitigate this effect.

4.4.1 Object Matching

Before measuring how consistently one RIM cell tracks an object, we need to first know which object it is tracking at any moment. This results in a matching problem between RIM cell predictions and the objects in the ground truth of the next frame.

For this goal, we propose to use image correlation to compute the similarity between the cell-wise predictions and ground truth objects. Assume we have a set of objects (represented as images) $\{y_i\}_{i=1}^{K_{\text{gt}}}$, and a set of RIM cell predictions $\{\hat{y}_j\}_{j=1}^{K_{\text{pred}}}$. First, we would like to define the correlation function. Here we represent an image with a tensor $\mathbb{U}^{H\times W\times C}$. The normalized correlation between two images x and y is defined as

$$\texttt{corr}(x, y) = \frac{\sum_{h,w} (x * y)_{h,w}}{\sqrt{\sum_{h,w} (x * x)_{h,w} \sum_{h,w} (y * y)_{h,w}}}. \tag{4.1}$$

where $*$ is the cross-correlation operator.

The above correlation value indicates the similarity between two images, which we use to measure the similarity between different object images and RIM cell predictions. We associate a RIM cell prediction with the object that has the largest correlation to it.

$$\hat{O}_i = \arg\max_j \texttt{corr}(\hat{y}_i, y_j) \tag{4.2}$$

where $\hat{O}_i$ is the ID of the object tracked by RIM cell i.

4.4.2 Average/Maximum Consistent Length

Using image correlation, we can find out which object each RIM cell is tracking. When testing on a series of time steps, we get a ID sequence $\{O_{i,1}, O_{i,2}, ..., O_{i,T}\}$ indicating what objects RIM cell i is tracking at different time steps. Furthermore, we would like to see if each RIM cell is consistently tracking the same object across time. For that, we propose two simple metrics, Average Consistent Length (ACL) and Maximum Consistent Length (MCL).

To define these two metrics, we first split the fore-mentioned ID sequence to several sub-sequences such that within each subsequence, the RIM is tracking the same object. For example, a sequence [1, 1, 1, 0, 0] would be split into [1, 1, 1] and [0, 0]. We call the length of each subsequence the consistent length, as, within each subsequence, a RIM cell is consistently tracking one object. The Maximum Consistent Length (MCL) in one sequence indicates how long a RIM cell stays consistent. The Average Consistent Length (ACL) shows the average consistency level of a RIM during the test time.

4.4.3 Numerical Results

Input to RIMs	Decoder	# Modules	ACL	MCL
Feature Map	SBD	6	<u>0.7214</u>	**0.8071**
Feature Map	BCD	6	0.6897	0.7835
Slots	SBD	3, 3	0.6440	0.7568
Slots	SBD	6, 3	0.6485	0.7521
Slots	SBD	6, 6	0.6341	0.7225
Slots	BCD	3, 3	**0.7267**	<u>0.7977</u>
Slots	BCD	6, 3	0.6757	0.7691
Slots	BCD	6, 6	0.6709	0.7733
	Average		0.6764	0.7703

Table 4.4: Maximum Consistent Length evaluation on different variants of OC-RIMs on Moving Sprites. # Modules refers to the number of RIM cells and the number of slots respectively. Style: **best** results and <u>second best</u>.

As shown in Table 4.4, the maximum consistent lengths of all variants of OC-RIMs are around 0.7703, suggesting that on average, during 77.03% of the time, a RIM cell is tracking the same object consistently. When we use feature maps as input, i.e., skipping slot attention, each RIM cell directly gets information from the feature maps and the models are able to maintain higher consistency. But with slot attention, the consistency slightly dropped. We argue that this might attribute to the input attention mechanism are not able to maintain temporal consistency. The input attention mechanism matches RIM cell hidden states and slots at each time step independently. Despite that the information of the previous time step is embedded in the hidden states, from the results, it is likely this information is still inadequate to keep the matching consistent across different time steps. Meanwhile, we also noticed that as the number of modules grows, the consistency metrics will also drop. Because the number of modules is more than the number of objects in the video, as shown in Figure 4.7, multiple RIM cells might

track the same objects and the repetitive predictions will be masked automatically. This allows for the repetitive RIM cells to switch objects arbitrarily because it would not affect the final prediction.

4.5 Beyond Performance

In order to analyze how different building blocks of our model affect both downstream performances and object representation learning, We performed an ablation study on RIMs and `OC-RIMs`. We explore the following aspects: (i) How different combinations of slot and RIM cell numbers influence both downstream performances and object representations, (ii) The performance difference between feeding feature maps and slots to RIMs, and (iii) What is the benefit of using a broadcasting operation in a component-wise decoder.

To this end, we ablated the model: (i) using combinations of 3/6 slots and 3/6 RIMs, (ii) feeding spatially flattened feature maps (FF) to RIMs (following the same approach used in [26]), we refer to it as `OC-RIMs` (FF), and (iii) choosing different component-wise decoders, either BCD or SBD. The numerical results are shown in Table 4.5 and 4.6 respectively.

Model	**Decoder**	**# Modules**	**MSE**	**SSIM**
ConvLSTM [54]	-	-	103.30	0.7070
GRU [22]	`CAT`	-	89.49	0.8484
RIM [21]	`CAT`	6	96.25	0.8312
`OC-RIMs` (FF)	`SBD`	6	81.22	0.8568
`OC-RIMs`	`CAT`	6, 3	88.99	0.8442
`OC-RIMs`	`SBD`	3, 3	87.25	0.8504
`OC-RIMs`	`SBD`	6, 3	<u>76.69</u>	<u>0.8688</u>
`OC-RIMs`	`SBD`	6, 6	76.98	<u>0.8688</u>
`OC-RIMs`	`BCD`	3, 3	89.74	0.8467
`OC-RIMs`	`BCD`	6, 6	81.21	0.8611
`OC-RIMs` (**ours**)	`BCD`	6, 3	**73.88**	**0.8742**

Table 4.5: Ablation study on varying the number of modules of Moving MNIST. Style: **best** performance and <u>second best</u>.

4.5.1 Input Attention

Input attention is responsible for transforming the encoded perceptual information and allocating it to different RIM cells. Below shows an example of the normalized input attention matching scores. The vertical axis represents RIM cells and the horizontal axis slots. Slot 3 corresponds to the null input (full-zero vector). In frame 1, for RIM cell 1, it puts almost equal attention on slot 1 and the null input, suggesting it is weakly matched with slot 1 and taking a down-scaled slot 1 as input. For RIM cell 0, it puts high attention on slot 2, meaning it is almost purely taking information from this one slot.

Model	Decoder	# Modules	MSE	SSIM
GRU [22]	CAT	-	85.58	0.8960
RIM [21]	CAT	6	74.78	0.9057
OC-RIMs (FF)	SBD	6	<u>39.66</u>	**0.9359**
OC-RIMs	CAT	6, 3	80.65	0.8999
OC-RIMs	SBD	3, 3	70.33	0.9022
OC-RIMs	SBD	6, 6	45.19	0.9306
OC-RIMs	BCD	3, 3	69.10	0.9077
OC-RIMs	BCD	6, 3	47.88	0.9284
OC-RIMs (**ours**)	SBD	6, 3	**39.60**	<u>0.9341</u>

Table 4.6: Ablation study on varying the number of modules of Moving Sprites. Style: **best** performance and <u>second best</u>.

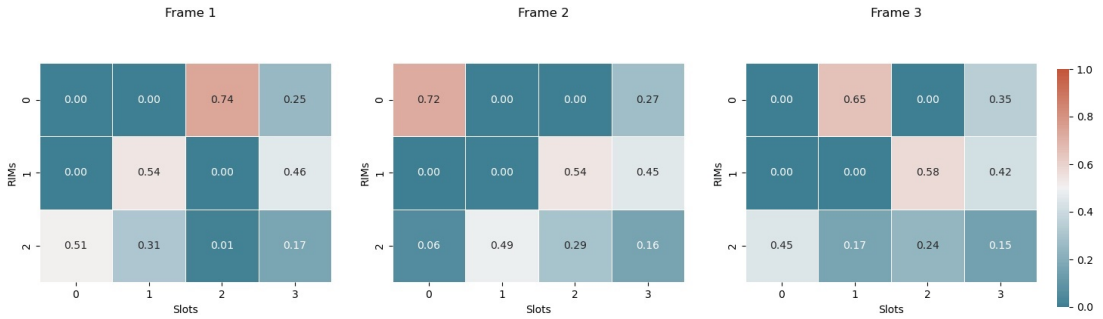

Figure 4.8: Examples of the input attention matching scores (normalized).

4.5.2 RIM Cell Activation/Utilization

RIMs have a selective activation mechanism based on input attention. The inputs to RIM cells consist of two parts, valid inputs and null inputs (full zero vector). We can rank the matching scores between RIM cell hidden states and the valid inputs. Only the RIM cells that put the most attention on the valid inputs should be activated. In [21], the authors argue that after training, RIMs are able to selectively activate the most relevant RIM cells depending on the input. Here we show the activation patterns of two samples with a baseline RIMs model on Moving MNIST. It is claimed that RIMs activate cells dynamically when there are distribution changes in the input [21]. However, this claim itself is weak because the distribution change is hard to strictly define. In Figure 4.9, we noticed that except for the two initial steps where the networks are warming up, for the rest of the time, the activation patterns of RIM cells always remain constant. It is hard to explain whether this is because RIMs fail to activate RIM cells dynamically or because there is no distribution change in the input.

It is also claimed in [21] that sparse activation (number of activation less than the number of RIM cells) is necessary for better performance. We have run experiments with different numbers of activations and different numbers of RIM cells. The results are shown in Table 4.7 and Table 4.8, which demonstrates performance impact of varying the numbers of activation k_a (number of RIM cells to activate). From our experiments,

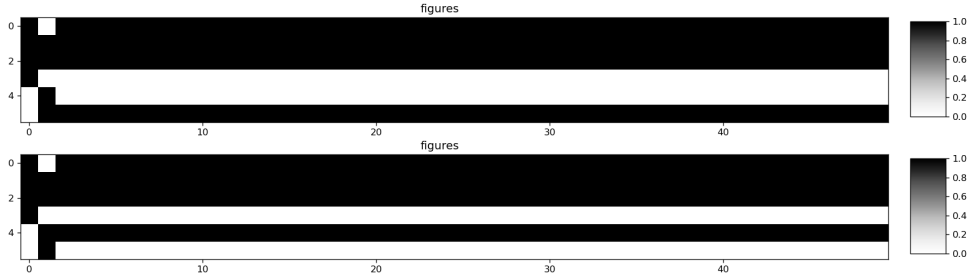

Figure 4.9: Activation patterns of two samples. The horizontal axis is the time step, and the vertical axis is the RIM cell. The black color means a RIM cell is activated at the corresponding time step. E.g., A black top-left cell suggests that the 0-th RIM cell is activated at the 0-th time step.

we can tell that sparse activation is not necessary for getting better performance. Instead, sparse activation always results in significantly worse performance compared with dense activation. Worse still, when we train a model with sparse activation and test it with full activation, the test MSE almost reaches 200. Because the unactivated RIM cells would have their gradients blocked during training, i.e., the weights of a RIM cell cannot be updated if it is not activated. It could be that when we have sparse activations, some RIM cells are not properly trained (because their gradients are blocked). Therefore, during evaluation with dense activation, those under-trained cells would disrupt other RIM cells and produce worse predictions in the end. In general, we can conclude that we can almost always use dense activation because it results in much better performance.

$\mathbf{k_a}$ / # RIM Cells	# Slots	MSE	SSIM
6 / 6	3	76.69	0.8688
3 / 6	3	105.99	0.8082
2 / 6	3	124.50	0.7668
3 / 3	3	87.25	0.8504

Table 4.7: Ablation study on varying the number of activations $\mathbf{k_a}$ on Moving MNIST. The used model is `OC-RIMs` with `SBD`.

We can also measure which RIM cells are most utilized to generate the output. This is measured by the L1 norm of the gradient of the sum of all pixels w.r.t. the cell-wise hidden states, as described below

$$u_i = \left|\left| \nabla_{h_i} \left(\sum_{m,n} O_{m,n} \right) \right|\right|_1, \tag{4.3}$$

where $O_{m,n}$ is the m-th row and n-th column of the output image O. The utilization patterns of two samples are shown below (normalized over RIM cells). Combined with Figure 4.9, we can tell that the predictions heavily depend on two activated RIM cells

$\mathbf{k_a}$ / # RIM Cells	MSE	SSIM
6 / 10	123.36	0.7674
4 / 10	155.29	0.7298
2 / 10	134.26	0.7500
4 / 6	131.66	0.7971
4 / 6 $\rightarrow$ 6 /6 *	191.26	0.7175

Table 4.8: Ablation study on varying the number of activations $\mathbf{k_a}$ on Moving MNIST. The used model is baseline RIMs with `CAT`. *: In this experiment run we trained the model with 4 / 6 and evaluated with 6 / 6.

and the others are barely used except in just a few time steps. This suggests that some information in the unactivated RIM cells is redundant.

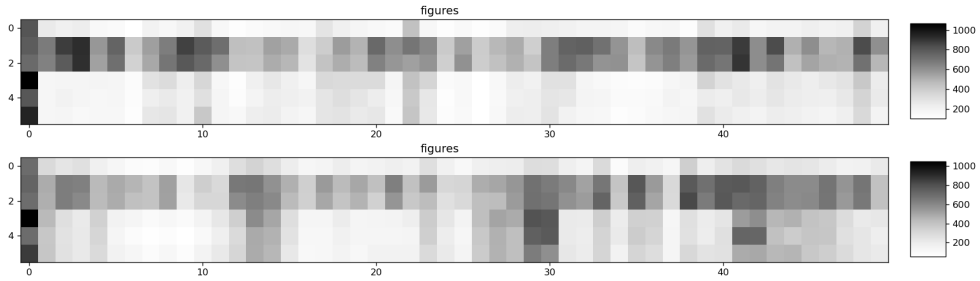

Figure 4.10: Utilization patterns of two samples. The horizontal axis is the time step, and the vertical axis is the RIM cell. E.g., The color of the top-left cell represents the utilization percentage of the 0-th RIM cell at the 0-th time step.

4.5.3 Slot Attention

We compare the performance of `OC-RIMs` with and without slot attention - `OC-RIMs` (FF). The results are shown in Figure 4.5 and Figure 4.6. According to both tables, the performance improvement brought by slot attention varies with the dataset. We argue that the advantage of `OC-RIMs` over `OC-RIMs` (FF) is that, with the per-frame GRU refinement in `OC-RIMs`, the model is able to decompose more complex shapes. While objects in Moving Sprites have simpler shapes, this advantage became less obvious.

In Figure 4.11, we see that at different moments, the three slots are focusing on different areas of the feature map.

4.5.4 Component-Wise Decoder

A closer look at Table 4.5 reveals that BCD performs better than SBD in the case of 6 RIMs and 3 slots. We also observed that the performance gap between these two component-wise decoders is close and is dependent on the number of RIM cells. However, SBD is better than BCD in most cases, which is most likely due to spatial

Figure 4.11: Example of the normalized matching score between three slots and different locations of the feature map at three different time steps.

broadcasting. Meanwhile, since spatial broadcasting is a non-parametric operation, SBD has fewer parameters than BCD in similar settings. It is likely for this performance and efficiency, SBD has been widely adopted within the recent object-centric learning literature. However, the optimal decoder choice still relies on the downstream task, data, and the model itself.

4.5.5 Number of Modules

We observed that the performance can vary largely with the number of modules we choose: number of RIM cells and number of slots. Since there are only three objects in each frame, intuitively 3 RIM cells should be adequate. However, models with 6 RIM cells show significantly lower MSE than those with 3. This suggests that one object might take multiple RIM cells to precisely predict. Meanwhile, the performance

dropped when we increase the number of slots from 3 to 6. This could suggest that using too many slots might mitigate the performance improvement brought by slot attention.

4.5.6 Independence of RIM Cells

Authors of [21] designed RIMs to have each RIM cell work near-independently. However, from Figure 4.5 to 4.6, we see that not every RIM cell can work independently to track one object. Sometimes it takes more than one cell to track the same object because one cell may make bad predictions and must be filtered out. From Table 4.6 and Table 4.6, we can also see that decreasing the number of cells indeed hurt the performance significantly. This raises great concerns because it is against the goals of [21]. In the following text, we test out the independence of RIM cells from two aspects: would the network still be able to make reasonable predictions, if we (i) disable the communication attention during test time, or (ii) randomly disable some of the cells.

Disable communication. In RIMs, each RIM cell can only get information through the communication mechanism. Specifically, for our datasets, all the objects move in an independent way - objects will not collide with each other. Therefore, all the cells should be able to make predictions even without communication. We ablate the communication module during the test and observe whether RIMs can still make fair predictions. Figure 4.12 shows the independent predictions made by each RIM cell where there is no communication mechanism.

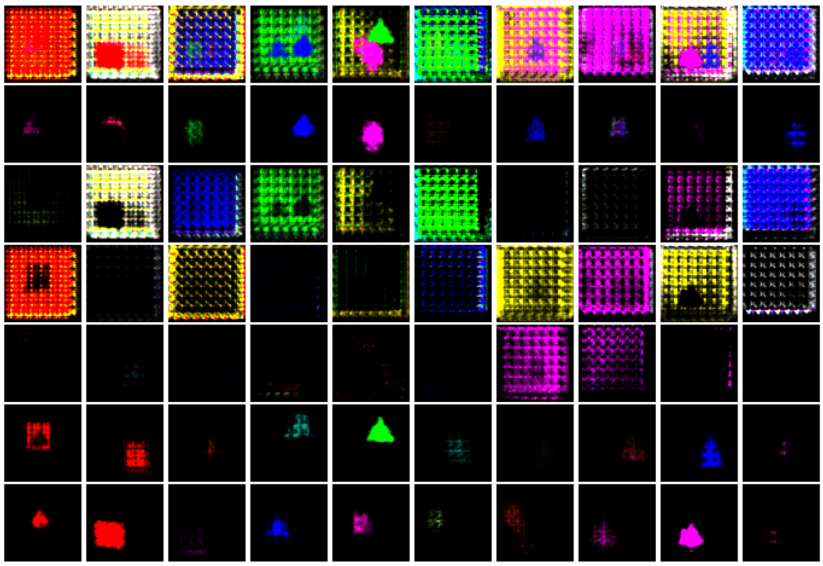

Figure 4.12: Independent predictions of each RIM cell without communication attention. The first row corresponds to the combined prediction and the other rows correspond to each RIM cell's prediction.

From Figure 4.12, we can clearly see that RIMs completely failed to make any reasonable predictions. Thus we can conclude that when during training, the RIM cells learn to co-adapt with each other through communication and therefore lose their

independence.

Disable cells. RIM cells have composability. Specifically, if we exchange their order or take out some of the cells, the network should still be able to work. In our previous experiment results, we also notice that some RIM cells are repetitive. It is worth testing whether if we remove these seemingly repetitive cells, we would still have the same predictions. In this ablation experiment, we disable two RIM cells during test time on an already trained model with six cells. Figure 4.13 shows the combined and individual predictions made by each RIM cell.

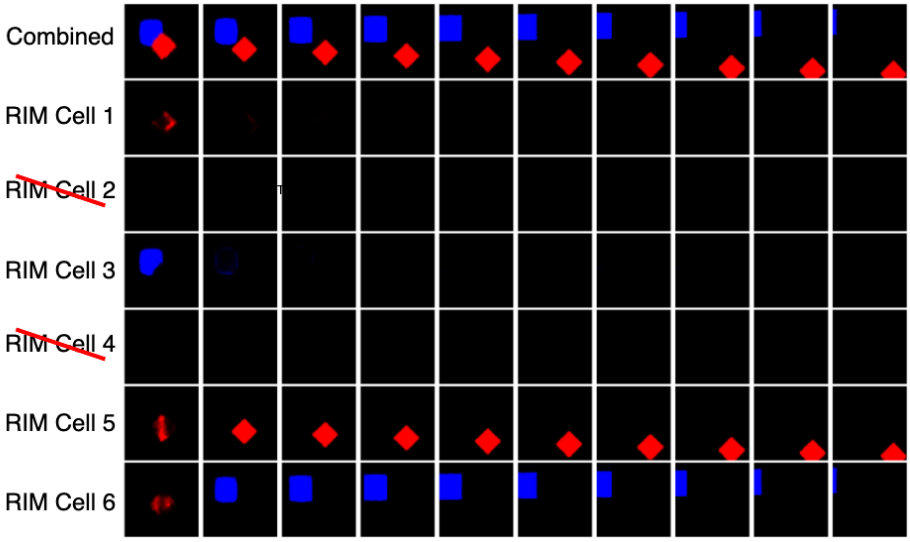

Figure 4.13: Independent predictions of each RIM cell when we disable RIM cells 2 and 4. The first row corresponds to the combined prediction and the other rows correspond to each RIM cell's prediction.

Comparing with the ground truth, we notice that one object is missing from the prediction and the red square becomes falsely blue. This indeed confirmed our previous conclusion that one object sometimes takes more than one cell to predict. When there are not enough cells, the predictions went wrong in terms of showing false color, fewer sharp edges, and/or losing some objects.

To sum up, when we disable communication, the model is completely not able to make reasonable predictions, which suggests each cell depends heavily on other cells. Despite the final prediction looks like each of them focuses on different objects, they have to use information from other cells to make such predictions. When we disable some of the cells while keeping the communication, the model is still able to make intelligible predictions, but they may lose track of some objects even when the working cells are still more than the existing objects. We can conclude that RIM cells can to some extent work flexibly because when we remove some cells, they can still function. But the use of communication leads to co-adaptation between the working cells. For future work, one should try either disabling the communication or limiting the information allowed to be transmitted during communication, for example, adding prior to the attention matching scores or using larger dropout.

4.6 Results of SCOFF

As we mentioned, we can seamlessly replace the transition model with SCOFF. Below we show the results of using SCOFF as our transition model. Figure 4.14 to 4.16 show the training loss, test MSE, and test F1 score of 10 experiment runs with SCOFF.

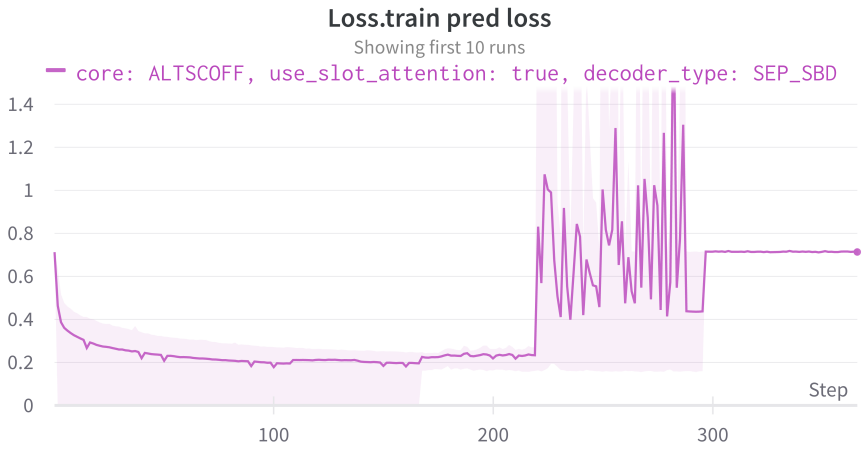

Figure 4.14: Training loss of object-centric SCOFF model.

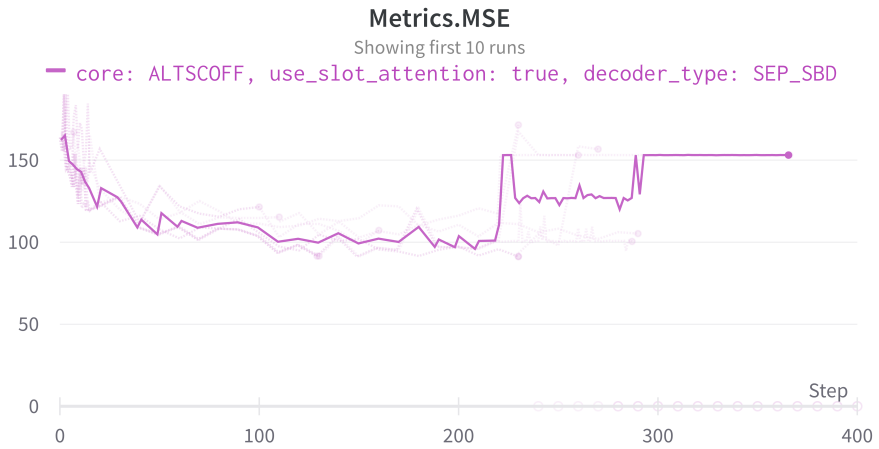

Figure 4.15: Test MSE of object-centric SCOFF model.

From the figures above, we can clearly see that SCOFF experienced training failures with all 10 runs. Even during the stable epochs, the MSE was much higher than its RIMs counterpart. We believe the instability is likely due to the Gumbel softmax since it involves sampling. It is not like dropout [72], where the effect of dropping out a neuron is only setting that neuron to zero. Here different sampling result means we infer over different models, which may cause instability.

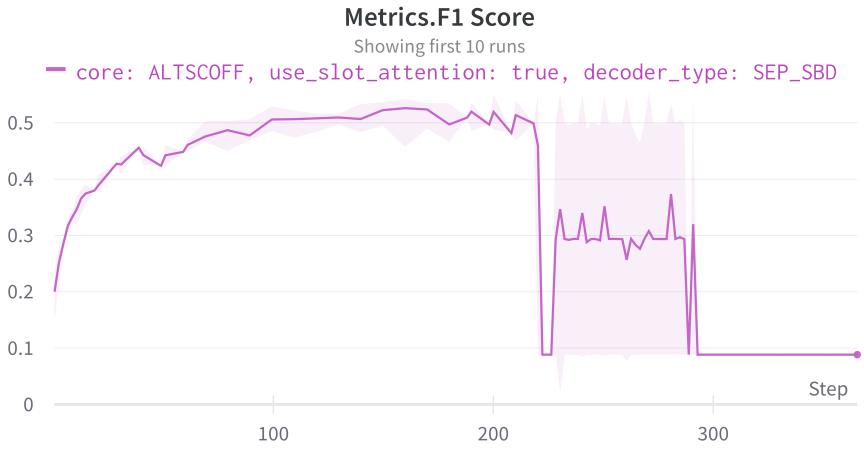

Figure 4.16: Test F1 score of object-centric SCOFF model.

4.7 Concluding Remarks

We look at recurrent independent mechanisms with an object-centric perspective to extend it to a model we call `OC-RIMs` which factorize video frames into representations of single objects. This model combines the original RIMs with two additional modules, a slot attention unit to formulate object-centric features, and a component-wise decoder to generate per-object predictions. Our qualitative and quantitative experimental results indicate the importance of object-centric representations by demonstrating a significant performance boost and the ability to decompose the scene into individual objects in a downstream video prediction task.

We notice that each RIM cell could not efficiently and independently track the same object and one RIM cell would occasionally switch to track another object. From the qualitative results in Figure 4.5 to 4.6, we notice that

For future work, we would explore a more efficient and consistent way to bind each RIM cell to one distinct object. Besides, the performance of RIM still depends heavily on the number of cells. It is important to enable RIMs to selectively activate cells in a more effective way, and therefore make it a more flexible model.

5 Conclusion

RIMs are a recently proposed modular neural network that has not yet been fully studied. This model learns the dynamics in a time sequence by modular blocks and decomposes the task into sub-tasks. In this research, we first applied RIMs to video prediction in order to understand whether it has superiority over monolithic networks. It is shown that RIMs can indeed surpass related baselines such as GRU on Moving MNIST dataset and can achieve comparable performance on Moving Sprites. Next, we extend the modularity in RIMs by introducing modularity in representations, i.e., object-centric representations. We implement this by using a Slot Attention module on the encoder side and using a component-wise decoder to replace the original concatenation decoder, resulting in `OC-RIMs`. In terms of MSE and SSIM, `OC-RIMs` showed significantly better performance over RIMs and other baselines. We also provided qualitative results demonstrating that `OC-RIMs` generate less deformed future frame predictions and that `OC-RIMs` learn to decompose a scene into objects automatically. When there are occlusions in the video, `OC-RIMs` are also able to infer the complete shapes of occluded objects.

To further understand how RIMs and `OC-RIMs` work, we perform extensive ablation studies. Noticing that `OC-RIMs` are able to bind different RIM cells to single objects, we explored whether each RIM cell is consistently tracking one object. Since our predictions do not contain an explicit label for which object it is tracking, we need to first match the individual prediction with objects. We use image correlation for this matching task. Next, we propose to use two simple new metrics, ACL and MCL, to measure the tracking consistency. The numerical results showed that a RIM cell, on average, tracks the same object consistently for 77.03% of the time.

Next, we dive deeper into every component of the `OC-RIMs` architecture. Starting from the input attention of RIMs, by inspecting the normalized attention scores, we saw that this module can indeed aggregate and allocate multiple inputs to each RIM cell. The selective activation mechanism is based on input attention, where the authors of RIMs claimed that sparse activation is necessary [21]. But in our experiments, by both the activation patterns and numerical results, we showed sparse activation is not needed and dense activation always results in much better performance.

SA and component-wise decoder are two new modules we introduced in `OC-RIMs`. We showed the attention scores of SA, where we see that each slot puts attention on different areas of the feature map, in which way SA is able to generate object-centric representations.

Overall, the newly proposed modular neural network RIMs showed better or comparable performance in video prediction across two datasets. By pushing the modularity further, we extended `OC-RIMs` by object-centric representation learning and achieved a significant performance improvement. As a by-product, `OC-RIMs` were able to let each RIM cell attend to one object at any specific moment. It showed that modular-

ity in both dynamics and image representations can improve performance in the video prediction task.

5.1 Limitations and Future Work

Due to the time limit of this thesis, we could not fully explore the possibilities of RIMs for achieving the best performance in video prediction. We have compared RIMs with comparable monolithic baselines, but we have not compared them with current state-of-the-art (SOTA) models since either they have a distinct architecture or they have multiple dedicated designs for video prediction, while RIMs are a generic modular RNN. To further prove the superiority of modular networks over monolithic ones, the modular counterparts of current SOTA models should be able to achieve a performance improvement.

There are also concerns and limitations of RIMs architecture. The first issue is the design of activation, which harms the performance significantly. The second issue is the communication mechanism. This mechanism should be an optional module in RIMs. But when we disabled this mechanism during tests, the model failed to make any reasonable predictions. This suggests that RIM cells learn to rely heavily on communications and are thus not working in a near-independent way, which is against the design goals of RIMs.

The last direction of future work is extending Slot Attention. As a simple yet effective feature extractor, slot attention is designed for static images. In `OC-RIMs` and SAVi[73], slot attention is used per-frame and is therefore still time-independent. In other words, slot attention does not learn any temporal dependencies. Can we thus extend slot attention to not only focus on different areas in one frame but also attend to different time dependencies across multiple frames?

Future work can focus on the following aspects

1. developing modular counterparts of current SOTA models for video prediction. Specifically, we can apply similar attention-based selective allocation and update mechanisms to these models.
2. new ways of selective updating to replace the original way of activation.
3. selectively using RIM cells to make predictions so that the redundancy in RIM cells can be reduced and the consistency improved.
4. constraining the information each RIM cell can receive and transmit through communication so that the independence of RIM cells can improve, which might help improve the robustness of the model.
5. extending slot attention to learn both spatial and temporal dependencies.
6. better training strategies for `OC-RIMs` or RIMs.

Here we briefly elaborate on some of the possible future studies.

Fully convolutional modular network. Inspired by [56], where the authors used a fully convolutional network but achieved results comparable to SOTA models. The conclusion of this paper makes us re-rethink whether we really need a

`Encoder+RNN+Decoder` structure. The core of RIMs is modularizing hidden state vectors and selective updating using attention. The concepts can be adapted to apply to convolutional networks. We can modularize feature maps in the channel dimension, and use the attention mechanism to apply different convolutional kernels to them. To make the operations fully convolutional, we can also use convolutions to get the queries, keys, and values instead of the linear transformation in the attention modules.

Modular Transformer. In recent years, Transformers [20] have gradually replaced RNNs in state-of-the-art sequential models for their extraordinary performance. For example, in SAVi [73], the transition model acting as a predictor is a Transformer. It is worth exploring how we can transfer the modular designs in RIMs to Transformers in order to achieve better performance.

Improved selection mechanism. Each RIM cell now consists of a hidden state and a recurrent unit. The recurrent unit of each RIM cell cannot be used by other cells. When we bind each cell to an object, it becomes a problem when we have variable numbers of objects. Because when we have more objects during tests than during training, we cannot easily increase the number of cells. Therefore, we can unbind cell hidden states and cell recurrent units. In this way, we would have two selection mechanisms in our network. The first is selecting inputs for each cell's hidden state, the other is selecting which recurrent unit to apply to each hidden state. However, whether or not a simple attention mechanism is capable of such selections remains to be answered. We could work on designing a new selection mechanism to implement this type of improved modular network.

Better training strategies. Our experiments with RIMs show that it is difficult to make each RIM cell specialize, which is the core of RIMs, while those with SCOFF also show that they are difficult to train as training often becomes unstable. Therefore, we believe we need to reflect on the training strategies for such modular neural networks. [74] has worked on this idea and proposed that we could use meta learning on RIMs to achieve better performance. More can be explored on this subject.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015.

[4] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 27, Curran Associates, Inc., 2014.

[7] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, Feb. 2019.

[9] P. Hu, Y. Lu, and Y. Y. Gong, "Dual humanness and trust in conversational AI: A person-centered approach," *Computers in Human Behavior*, vol. 119, p. 106727, June 2021.

[10] T. Vidal and M. Schiffer, "Born-Again Tree Ensembles," in *Proceedings of the 37th International Conference on Machine Learning*, pp. 9743–9753, PMLR, Nov. 2020.

[11] C. Enrique, J. Gutiérrez, and A. Hadi, "Learning bayesian networks," *Expert Systems and Probabilistic Network Models, Monographs in Computer Science, Springer, Berlin*, pp. 481–528, 1997.

[12] P. Smolensky, R. T. McCoy, R. Fernandez, M. Goldrick, and J. Gao, "Neurocompositional computing: From the central paradox of cognition to a new generation of ai systems," *arXiv preprint arXiv:2205.01128*, 2022.

[13] A. Goyal, A. Didolkar, N. R. Ke, C. Blundell, P. Beaudoin, N. Heess, M. Mozer, and Y. Bengio, "Neural Production Systems: Learning Rule-Governed Visual Dynamics," *arXiv:2103.01937 [cs, stat]*, Mar. 2022.

[14] A. Goyal, A. Lamb, P. Gampa, P. Beaudoin, S. Levine, C. Blundell, Y. Bengio, and M. Mozer, "Factorizing Declarative and Procedural Knowledge in Structured Dynamical Environments," p. 24, 2021.

[15] A. Dittadi, S. Papa, M. De Vita, B. Schölkopf, O. Winther, and F. Locatello, "Generalization and robustness implications in object-centric learning," 2021.

[16] C. P. Burgess, L. Matthey, N. Watters, R. Kabra, I. Higgins, M. Botvinick, and A. Lerchner, "MONet: Unsupervised Scene Decomposition and Representation," *arXiv:1901.11390 [cs, stat]*, Jan. 2019.

[17] X. Chen, W. Wang, J. Wang, and W. Li, "Learning object-centric transformation for video prediction," in *Proceedings of the 25th ACM international conference on Multimedia*, pp. 1503–1512, 2017.

[18] M. Engelcke, A. R. Kosiorek, O. P. Jones, and I. Posner, "Genesis: Generative scene inference and sampling with object-centric latent representations," 2019.

[19] N. R. Ke, A. Didolkar, S. Mittal, A. Goyal, G. Lajoie, S. Bauer, D. Rezende, Y. Bengio, M. Mozer, and C. Pal, "Systematic evaluation of causal discovery in visual model based reinforcement learning," 2021.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[21] A. Goyal, A. Lamb, J. Hoffmann, S. Sodhani, S. Levine, Y. Bengio, and B. Schölkopf, "Recurrent Independent Mechanisms," *arXiv:1909.10893 [cs, stat]*, Nov. 2020.

[22] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014.

[23] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised Learning of Video Representations using LSTMs," *arXiv:1502.04681 [cs]*, Jan. 2016.

[24] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf, "Object-centric learning with slot attention," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11525–11538, 2020.

[25] Z. Lin, Y.-F. Wu, S. V. Peri, W. Sun, G. Singh, F. Deng, J. Jiang, and S. Ahn, "Space: Unsupervised object-oriented scene representation via spatial attention and decomposition," 2020.

[26] R. Assouel, L. Castrejon, A. Courville, N. Ballas, and Y. Bengio, "Vim: Variational independent modules for video prediction," in *Conference on Causal Learning and Reasoning*, pp. 70–89, PMLR, 2022.

[27] N. Watters, L. Matthey, C. P. Burgess, and A. Lerchner, "Spatial Broadcast Decoder: A Simple Architecture for Learning Disentangled Representations in VAEs," *arXiv:1901.07017 [cs, stat]*, Aug. 2019.

[28] Z. He, J. Li, D. Liu, H. He, and D. Barber, "Tracking by animation: Unsupervised learning of multi-object attentive trackers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1318–1327, 2019.

[29] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, pp. 541–551, Dec. 1989.

[30] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Apr. 2015.

[31] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1492–1500, 2017.

[32] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997.

[33] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Light Gated Recurrent Units for Speech Recognition," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, pp. 92–102, Apr. 2018.

[34] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[35] W. Yu, Y. Lu, S. Easterbrook, and S. Fidler, "Efficient and information-preserving future frame prediction and beyond," 2020.

[36] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating Videos with Scene Dynamics," in *Advances in Neural Information Processing Systems*, vol. 29, Curran Associates, Inc., 2016.

[37] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *arXiv:1409.0473 [cs, stat]*, May 2016.

[38] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

[39] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, "Video Swin Transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3202–3211, 2022.

[40] A. Goyal, A. Didolkar, A. Lamb, K. Badola, N. R. Ke, N. Rahaman, J. Binas, C. Blundell, M. Mozer, and Y. Bengio, "Coordination Among Neural Modules Through a Shared Global Workspace," *arXiv:2103.01197 [cs, stat]*, Mar. 2022.

[41] Y. Bengio, "The consciousness prior," *arXiv preprint arXiv:1709.08568*, 2017.

[42] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[43] K. Greff, R. L. Kaufman, R. Kabra, N. Watters, C. Burgess, D. Zoran, L. Matthey, M. Botvinick, and A. Lerchner, "Multi-object representation learning with iterative variational inference," in *International Conference on Machine Learning*, pp. 2424–2433, PMLR, 2019.

[44] A. Clark, "Whatever next? predictive brains, situated agents, and the future of cognitive science," *Behavioral and brain sciences*, vol. 36, no. 3, pp. 181–204, 2013.

[45] S. L. Mullally and E. A. Maguire, "Memory, imagination, and predicting the future: A common brain mechanism?," *The Neuroscientist*, vol. 20, no. 3, pp. 220–234, 2014.

[46] J.-T. Hsieh, B. Liu, D.-A. Huang, L. F. Fei-Fei, and J. C. Niebles, "Learning to Decompose and Disentangle Representations for Video Prediction," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018.

[47] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," *Advances in neural information processing systems*, vol. 29, 2016.

[48] C. Choi, J. H. Choi, J. Li, and S. Malla, "Shared cross-modal trajectory prediction for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 244–253, 2021.

[49] T. Lan, T.-C. Chen, and S. Savarese, "A Hierarchical Representation for Future Action Prediction," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), vol. 8691, pp. 689–704, Cham: Springer International Publishing, 2014.

[50] B. Soran, A. Farhadi, and L. Shapiro, "Generating Notifications for Missing Actions: Don't Forget to Turn the Lights Off!," in *2015 IEEE International Conference on Computer Vision (ICCV)*, (Santiago, Chile), pp. 4669–4677, IEEE, Dec. 2015.

[51] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human Trajectory Prediction in Crowded Spaces," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Las Vegas, NV, USA), pp. 961–971, IEEE, June 2016.

[52] "Activity Forecasting." https://link.springer.com/content/pdf/10.1007/978-3-642-33765-9_15.pdf.

[53] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra, "Video (language) modeling: a baseline for generative models of natural videos," *arXiv preprint arXiv:1412.6604*, 2014.

[54] X. SHI, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. WOO, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[55] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-Conditional Video Prediction using Deep Networks in Atari Games," in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[56] Z. Gao, C. Tan, L. Wu, and S. Z. Li, "SimVP: Simpler Yet Better Video Prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3170–3180, 2022.

[57] X. SHI, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. WOO, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[58] E. L. Denton, S. Chintala, a. szlam, and R. Fergus, "Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks," in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[59] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," Feb. 2016.

[60] T. Xue, J. Wu, K. Bouman, and B. Freeman, "Visual Dynamics: Probabilistic Future Frame Synthesis via Cross Convolutional Networks," in *Advances in Neural Information Processing Systems*, vol. 29, Curran Associates, Inc., 2016.

[61] A. X. Lee, R. Zhang, F. Ebert, P. Abbeel, C. Finn, and S. Levine, "Stochastic adversarial video prediction," *arXiv preprint arXiv:1804.01523*, 2018.

[62] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle, "Neural autoregressive distribution estimation," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 7184–7220, 2016.

[63] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[64] L. Castrejon, N. Ballas, and A. Courville, "Improved conditional vrnns for video prediction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7608–7617, 2019.

[65] R. Rakhimov, D. Volkhonskiy, A. Artemov, D. Zorin, and E. Burnaev, "Latent video transformer," *arXiv preprint arXiv:2006.10704*, 2020.

[66] S. Aigner and M. Körner, "The importance of loss functions for increasing the generalization abilities of a deep learning-based next frame prediction model for traffic scenes," *Machine Learning and Knowledge Extraction*, vol. 2, no. 2, pp. 78–98, 2020.

[67] T. Kipf, G. F. Elsayed, A. Mahendran, A. Stone, S. Sabour, G. Heigold, R. Jonschkowski, A. Dosovitskiy, and K. Greff, "Conditional object-centric learning from video," *arXiv preprint arXiv:2111.12594*, 2021.

[68] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Dec. 2014.

[69] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[70] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[71] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[72] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," p. 30.

[73] T. Kipf, G. F. Elsayed, A. Mahendran, A. Stone, S. Sabour, G. Heigold, R. Jonschkowski, A. Dosovitskiy, and K. Greff, "Conditional Object-Centric Learning from Video," Mar. 2022.

[74] K. Madan, N. R. Ke, A. Goyal, B. Schölkopf, and Y. Bengio, "Fast and Slow Learning of Recurrent Independent Mechanisms," *arXiv:2105.08710 [cs]*, May 2021.

Implementation Details

A

A.1 Encoder Architecture

The structure of the convolutional encoder for RIMs baseline is as follows.

Layer	Activation	Parameter
Conv2d	ELU	(16, 4, 2)
LayerNorm [4]		
Conv2d	ELU	(32, 4, 2)
LayerNorm		
Conv2d	ELU	(64, 4, 2)
LayerNorm		
Flatten		
Linear	ELU	64
LayerNorm		

Table A.1: Encoder Structure for RIMs baseline. Parameters for Conv2d layers represent output channel, kernel size, and stride respectively. Parameter for Linear layer is the output size.

When we use slot attention, or when we feed feature maps directly to RIMs, we use the encoder below in Table A.2 to produce feature maps.

Layer	Activation	Parameter
Conv2d	ELU	(16, 4, 2)
LayerNorm [4]		
Conv2d	ELU	(32, 4, 2)
LayerNorm		
Conv2d	ELU	(64, 4, 2)
SoftPosEmbed		
SpatialFlatten		
Flatten		
Linear	ELU	64
LayerNorm		

Table A.2: Encoder Structure for producing flattened feature map (FF). SoftPosEmbed stands for soft positional embedding. Spatial flattens the height and width dimensions.

Hyperparameter Sweep

B

Here we show a sweep in the hyperparameter space. Specifically, we show the effect on the performance of varying the number of slots, the number of RIM cells, slot size, and input value size (in the input attention). The results below are produced on Moving MNIST.

B.1 Loss and Metrics over Epoch

Figure B.1 to B.4 show how the training loss and metrics evolve around training epochs.

Figure B.1: Train loss curve versus epoch (step).

B.2 Correlation between Metrics

Figure B.5 to B.6 demonstrate the correlation between the three metrics we are using to evaluate models.

B.3 Performance Impact of Hyperparameters

Figure B.7 to B.9 demonstrate the hyperparameters' impact on performance (test MSE).

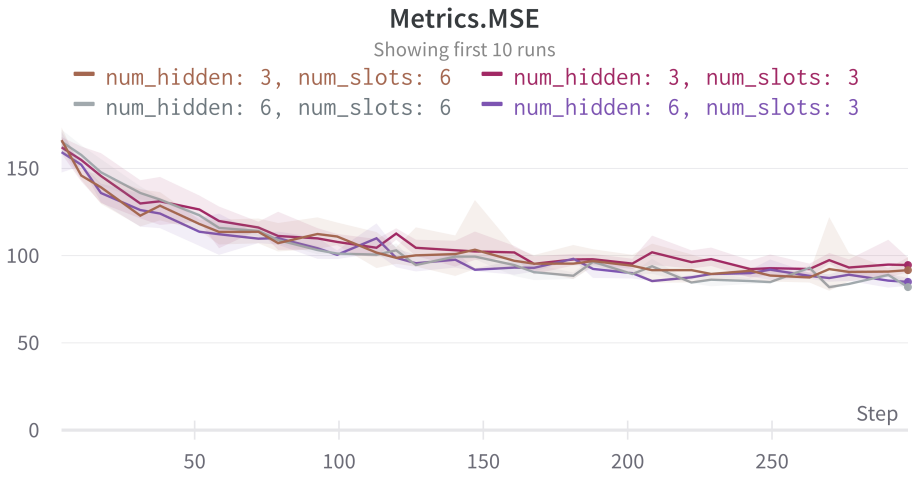

Figure B.2: Test MSE curve versus epoch (step).

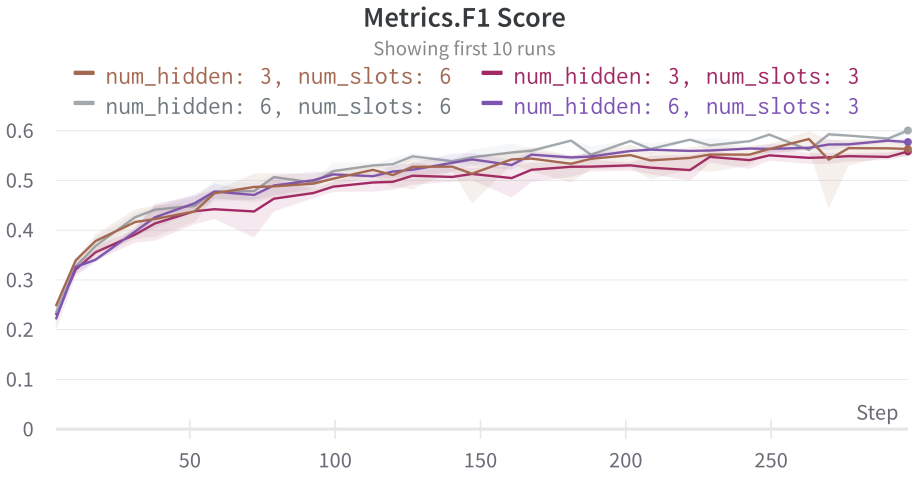

Figure B.3: Test F1 score curve versus epoch (step).

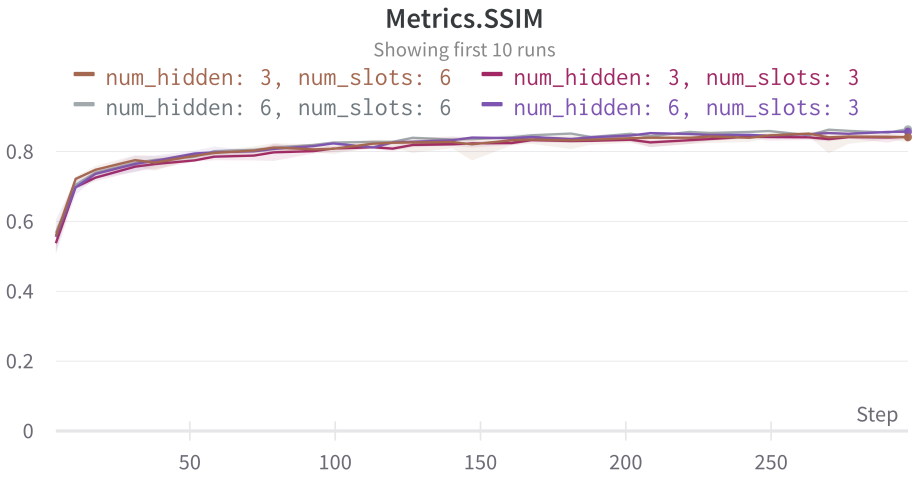

Figure B.4: Test SSIM curve versus epoch (step).

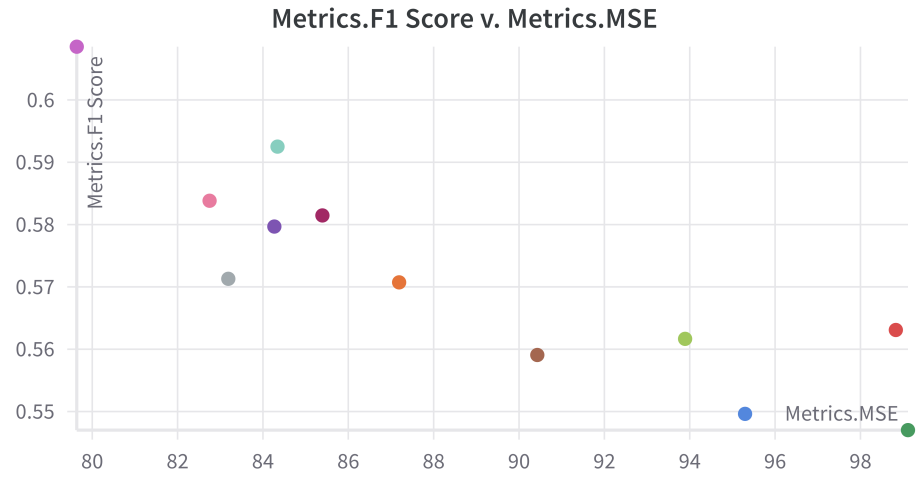

Figure B.5: Correlation between two metrics: F1 score and MSE.

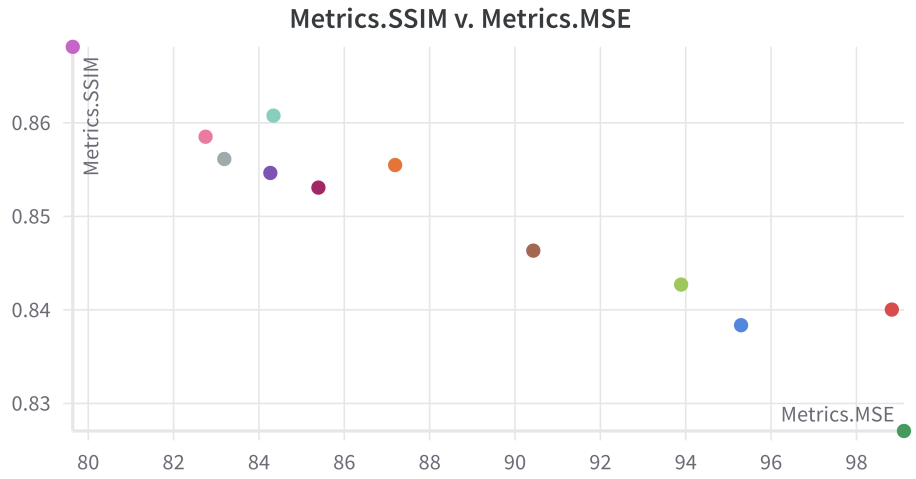

Figure B.6: Correlation between two metrics: SSIM and MSE.

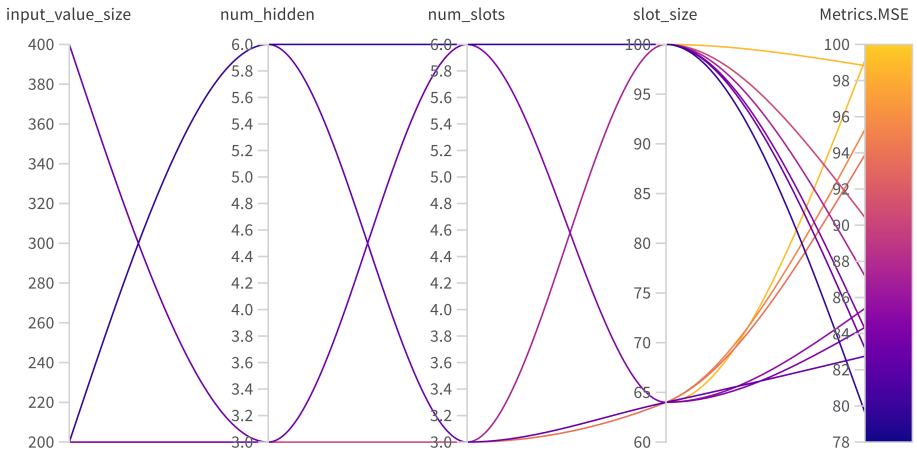

Figure B.7: Relations between test MSE and hyperparameters, namely input value size (input_value_size), number of RIM cells (num_hidden), number of slots (num_slots), and slot size (slot_size).

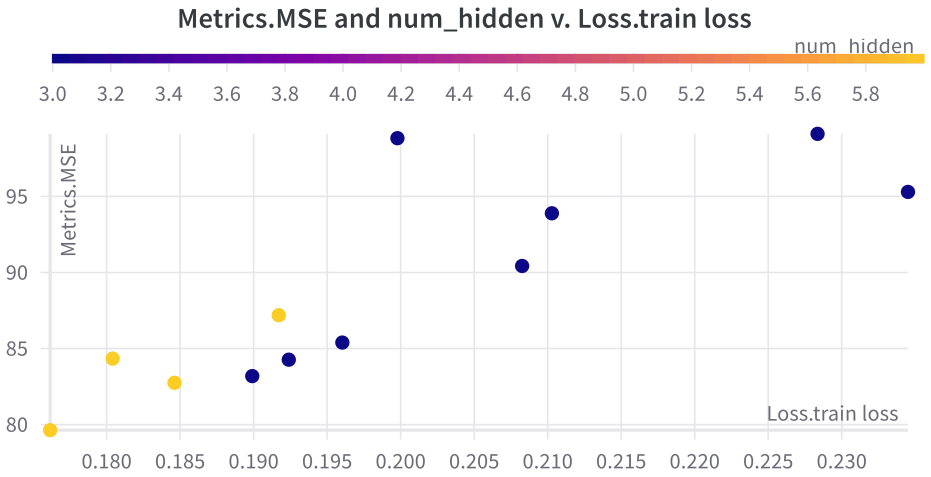

Figure B.8: Correlation between test MSE and training loss.

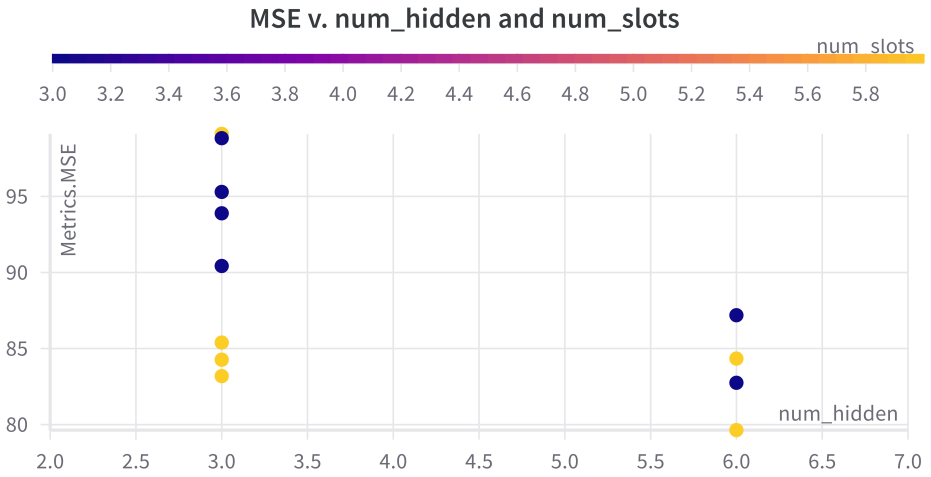

Figure B.9: Correlation between test MSE and the number of RIM cells and the number of slots.