
UNIVERSITAT DE BARCELONA


**Universitat de Barcelona**

**Department of Mathematics and Computer Science**

**Master on Artificial Intelligence**

**Final Master Thesis**

---

# A Comprehensive Survey on Deep Future Frame Video Prediction

---

***Author: Javier Selva Castelló***

***Supervisors: Sergio Escalera Guerrero and Marc Oliu Simón***

***January 2018***

# Abstract

The recent advances in Deep Learning techniques and the need for huge amounts of labeled information have encouraged new research in unsupervised learning procedures. These would allow for virtually unlimited data that could be used to improve many Machine Learning solutions. In particular, Computer Vision tasks could greatly benefit from successful models that learned valid abstract features of images and videos without supervision. Different generative models (and in particular Generative Adversarial Networks) and Convolutional Auto Encoders have shown some features that allow them to learn visual data distributions without the need of labels. Traditional generative models have focused on simply drawing samples from the estimated data distribution and autoencoders to reconstruct or denoise the inputs as a way to build systems that learn useful representations of images or videos. However, recent years have seen an increase in proposals that attempt to predict future frames in a video sequence as a procedure to learn valid visual abstract features.

The focus of this project is to analyse the task of future frame video prediction in the form of a Comprehensive survey. Frame prediction is a young field that started around 4 years ago and a broad analysis of the problem has not been carried out yet. In the present document, the state of the art is thoroughly examined to provide a deeper understanding of the task. The different proposals to solve future frame generation are analysed and compared from both theoretical and experimental viewpoints. In order to perform an effective comparison, available online models are tested using a shared objective and common video prediction benchmarking datasets. Strengths and weaknesses of the different proposals are examined and the produced results analysed. Based on those results, the different problems and key issues of video prediction are discussed in order to shed some light on the reasons that contribute to prediction quality that different approaches are able to produce. Finally, it is important to highlight the usage of the produced theoretical analysis and results on two different scientific publications: a CVPR 2018 submission and a survey for a Springer book.

**Keywords** frame prediction, generative models, adversarial models, video modeling, unsupervised learning, representation learning

# Contents

# Chapter 1

# Introduction

## 1.1 Computer Vision

The field of Computer Vision (CV) focuses on making computers able to gain high-level understanding of visual data. CV works for identifying and organizing elements in a visual scene in order to gain abstract perception and perform different tasks such as objects and face classification, image restoration, semantic segmentation, etc. Traditionally, CV has focused its efforts in extracting useful information from images in the form of abstract features that represented the necessary elements of the image for the task at hand. Then, these image features were used with a Machine Learning (ML) algorithm to learn valid feature patterns to successfully solve specific problems. Recent advances in learning algorithms, and in particular in Deep Learning (DL) techniques, have blurred the line between feature extraction and prediction with a new set of architectures that are able to extract useful features and perform learning at the same time. In general, DL has several advantages over traditional methods: composed of multiple hidden layers, they allow for the model to create more abstract and stratified representations of the data. This models have existed through ML history but was not until the recent developments for training such models and the advances in the hardware industry (particularly those performed in the Graphic Processing Units (GPUs) which allow fast use of graphical operations at very low level) that deep architectures have seen a renewed interest and huge application. A particularly interesting family of DL models is Convolutional Neural Networks (CNNs). They are based on the idea of parameter sharing across the input, meaning that the same weights will be applied to the whole of the data. For this reason they can deal with highly dimensional inputs (for example and in general, raw images) from which they extract useful features and perform pattern learning simultaneously. CNNs do not only execute both tasks together, but they are also exceptionally good in doing so. In summary, they allow for deeper architectures but also for inputs with great dimensionality, thus avoiding the need of having to manually design algorithms for feature extraction. This new models are consequently able to better exploit all structural and visual information contained in an image and find by itself the features that better solve a specific task. Furthermore, the layered abstract representation that is obtained grants a system that is better able to build richer features: the first layers of the architecture will extract low level features used by consequent layers to portray more complex information.

## 1.2 Unsupervised Learning

Even though DL models present those advantages, they come with a special drawback: huge amounts of data are needed in order to train successfully such systems. This is particularly problematic in the case of supervised learning. Even though the 21st century is an era of information where data is more than abundant, supervised learning needs for humans to manually annotate the data in order for a ML algorithm to learn the necessary patterns to perform the task. The need in DL for large amounts of information is then an important bottleneck in the process of learning complex image models, as huge amounts of labeled images are needed. This has encouraged for recent developments to study new ways of learning visual representations in the form of unsupervised tasks. While supervised learning tries to learn a function from some input data to an output label, unsupervised learning tries to learn hidden structure of the data by itself. If these approaches were to succeed, DL and an internet full of images could bring unimaginable advances to the field of CV.

## 1.3 Generative Models

An idea that has recently brought increasing attention for solving unsupervised learning in CV, is using generative models where "labels" are the images themselves, resulting in some sort of semi-supervised (or weakly supervised) environment, as manual annotations are not needed. In a nutshell, these proposals try to approximate the data distribution by producing it. Different generative models perform this task in varying fashions. Some try to produce new completely unseen images given a huge set of training examples as sampling from the learned distribution. Others try to reconstruct or denoise the input after it has undergone a transformation or projection into feature space. It may seem a little farfetched to generate images in order to solve the different issues that CV tries to solve. However, there is an important reason for working with generative models: a model that successfully generates realistic images is assumed to have learnt the underlying patterns in the data, which can then be extrapolated to several CV tasks. Some recent works have shown that models trained for a specific task could leverage the discovered patterns for uses in other tasks (e.g. [86]). Take for example a generative model that draws random vectors from a probabilistic normal distribution and learns to build car images from that. In order to do so the model would have to be able to transform a random vector into something resembling a car. This architecture will learn consequently that specific parts of the vector should transform into wheels and windows, headlights and specific overall shape in order to approximate the images that are being used to correct its behaviour. Afterwards one could fine tune such model with a few labeled examples to detect cars in an image: the task of understanding car features has already been solved in an unsupervised way. So, in the end the model would have learned specific features that make a car by trying to picture one. In this sense, it is possible to understand why a generative model would be an appropriate approach for learning useful representations of visual data. In a similar fashion, humans learn about the world with almost no supervision. Since an early age babies explore the physics of the world just by observing and interacting without supervision. They are able to form complex predictive models of the physics that govern the world around them. These models, over time, get sophisticated enough to allow for reasonable predictions to unseen situations [115]. By throwing things and watching them fall to the ground several times childs may build a model

that resembles something close to gravity: things without support go down.

## 1.4 Using temporal information

CV is a field that usually tries to pick up on human visual perception. It applies those ideas to make automatic artificial systems perform the same underlying information processing that humans do without problem to solve specific tasks. For this reason many researchers have suggested [3, 32, 74, 87, 138] that temporal information could help the task of learning about the structure of objects. In this sense, this could help solve several CV problems such as occlusion in object recognition, identifying an object when shown from a different perspective or size, reusing information for objects that are similar but not the same, etc. For this reason, and thanks to the recent advances in DL that allow for inputs with high dimensionality, recent approaches [37, 66, 88, 136] have tried to learn useful image representations using videos. Videos of natural scenes contain very diverse phenomena: objects deform and move quickly, occlude and dis-occlude each other, scene lighting changes, interaction among different objects and camera movement is some of the complexity that the time dimension adds to visual data. These new systems not only learn visual models from single images but also how they change over time, the physics of movement, the different perspectives, better understanding of depth and occlusion, etc. giving the model a broader insight of visual data. The features that may arise with such architectures allow for relative representations. In other words, being able to understand not only the absolute visual abstraction of a static object but also how it may evolve over time, and thus its different points of view and appearance relative to the scene it is embedded in. Time allows these models to create rich relationships among instances of the same visual elements and in doing so it enriches the possibility of visual abstractions of data.

Given the promising advantages of both, generative models and video modeling, it seems like a good idea to work with generative models of videos that leverage all possible information necessary to perform several CV tasks in an unsupervised fashion with successful results. A particular kind of generative models with these characteristics are the ones that focus on future video prediction. In other words, the models that given a sequence of frames are capable of generating the next following frames or even frames that are further in the future. The interest in such architectures is double: on one hand, the ability to generate images and videos needs a proper visual representation that could be used for many other CV tasks; on the other hand, such model would learn to draw specific visual features from a random vector, but also how to encode the input video in a way that is representative enough as to generate future frames from it. These characteristics make these models ideal for learning to extract useful information from images and videos. They could also be later used to perform other CV tasks by fine-tuning the models with small supervised sets. As the model has already learned to work with visual information, the only remaining work is to train the system towards a specific task. In the whole of this document, the terms “video prediction”, “frame prediction”, “future frame generation” and other possible variations will be used indifferently to refer to the idea of predicting the next frames in a video given a sequence of past frames.

## 1.5 Predictive Coding

The idea that this could be a useful way to learn comes not only from experiments on unsupervised learning but also from the way humans perceive. The human visual system has the ability to learn visual concepts from mainly unsupervised examples. It is robust to new views, changes in lightning, occlusion and within-class variations. In theoretical neuroscience it has been proposed that in order to do so the brain has to build internal models of the world and its complexity. With them, it is constantly predicting future mental and perceptual states based on memory and context (Visual Short-Term Memory (VSTM), and long-term memory [44]). The sensory input is understood when the predictions match the reality. This is achieved by feedback sent from the higher visual areas to the early (low level) visual areas [75]. This framework is known as Predictive Coding [94]. The main idea of this model states that the brain continually generates models of the world in order to predict sensory input. In other words it predicts an image of the future and uses the actual future (when it arrives) to correct those predictions in order to learn. In this sense, the brain would be using only the mismatch between its models and reality to produce better models [21]. So it would seem that video prediction is no nonsense. The models could compare its predictions against the actual future frames in order to send a signal up the network hierarchy to improve itself, as it is generally done in backpropagation but using other frames as supervision.

## 1.6 Applications and Motivation

Future frame prediction could serve as an invaluable technique to learn image and video representations in an unsupervised manner. As mentioned above, some have shown how knowledge learned for a specific task can be transferred as pre-training for other tasks. However, there is a difference between predicting as an objective and predicting as a medium for learning useful representations for other tasks. Mainly in the sense that some have proposed methods for video prediction oriented to very specific problems and is difficult to see how those models could extrapolate to a more general video prediction task (e.g. [71, 79]). However, visual representations acquired in an unsupervised setting by predicting future frames could benefit many CV tasks: object detection, tracking and motion prediction, gesture recognition, body pose recovery and video segmentation are some examples.

Furthermore, future frame generation not only could serve as a useful scenario for unsupervised learning, but also as a mean to develop several applications such as robotics [54], video compression [5], slow motion effect [62] and inpainting [31]. For instance, a predictive system for robots would contribute to create visual representations of the future in order to help decision making [29, 49, 13] and improve the agent-environment interactions. It also would enable agents to learn about the world on their own (or with little supervision), helping to scale up interactive learning. A synthetic example of this is Oh et al. [79] who used a visual predictive model to improve the capabilities of a Deep Q-Learning algorithm for playing Atari games. Another particular case of autonomous robots that has attracted huge interest throughout history, and particularly in recent years, is self driving cars. This emergent type of vehicle could highly profit from the abilities that a proper visual predictive system would provide [65, 20, 61]: such cars could, for instance, brake or slow down when predicting a pedestrian walking across the street. Other applications include anticipated detection of dangerous situations in surveillance

cameras, such as fights, robberies, falls in elderly people, etc. Weather prediction is another field that could benefit from video prediction, where radar cloud images could be predicted to forecast weather [103, 53].

In summary, video prediction encloses a huge potential for building complex models of the world useful for many different tasks. It would enable artificial systems to create a better understanding of their environments and tasks to solve. It could help build richer features to work with and has the potential to get us closer to the way human brains work.

## 1.7 Objectives

The purpose of this project is to study current State of the Art (SOTA) techniques for video prediction, presenting them within a comprehensive survey. The research of this problem will involve understanding the complexity of it, the different solutions that have been proposed, studying the strengths and weaknesses of the present SOTA, what are the preconceptions and misconceptions common to this field, comparing the different possible approaches and drawing a complete and insightful description of the problem. The project is organized from the standpoint of several objectives:

- First, a complete analysis of the state of the art to better understand the problem: Video prediction drinks mainly from image processing (information extraction, generative models, motion detection, optical flows, etc.) and sequence modeling. Regarding the former, common techniques for future frame prediction are Convolutional Neural Networks which can serve the purpose of both creating a rich abstract representation of visual content and generating new images. Regarding sequence modeling, commonly recurrent models such as Recurrent Neural Networks (RNNs) or in particular Long Short-Term Memory networks (LSTM) are used. Finally, the video prediction problem has resulted in the development of new ideas and several combined techniques for solving it: Auto Encoders (AE) or Generative Adversarial Networks (GANs) are generally used in many different ways as will be detailed in following sections.

- Second, an extensive and detailed description of the problem taking special care of pointing the common approaches with their respective advantages and disadvantages. The principal issues that arise when trying to solve future prediction will be studied together with proposed solutions: blurriness produced by common losses used for training and a continuum of possible futures, future prediction as a complex multimodal problem and recurrent models that accumulate prediction errors when trying to predict further in the future are some examples of problems that arise and that will be broadly discussed.

- Third, a comparison of the models from two different points of view will be produced: theoretical model description and quantitative and qualitative results. On one hand, a report of the different models from an architectural standpoint and their ability to learn and produce reasonable futures. On the other, a quantitative comparison will also be carried out by establishing a baseline or common ground. Default settings for the experiments will be defined, as every approach is different in essence and every author uses different data, predicts a different number of frames from different past information or measures results with various metrics. In order to perform these tests, the available code from SOTA models

will be modified (as far as possible without changing the essence of the proposal) to fit the proposed experimental setting. Finally, an extensive analysis of the results also relative to the model used per se. This last part is probably the core of the contributions of this project that seeks to establish the strengths that a model for frame prediction should have. The insights produced from the experiments in the balanced setting expects to provide just that.

- Finally, to highlight additional results from this thesis: part of the data collected and analysed for this project has been used in a CVPR submission together with a new proposal for future frame prediction (Arxiv version can be seen in Appendix A); also, the survey part of the thesis is in the process of appearing as a book chapter in a Springer book on spatio-temporal prediction.

# Chapter 2
# Basics

The focus of this section is on highlighting some basic concepts necessary for understanding the task of future frame prediction and the proposed solutions. The main idea is to guide the reader to better understand the models, procedures and evaluations for video prediction that will be detailed in the following sections. In particular, a brief description of the most commonly used models of ML (specially the ones used for the task at hand), the difference between shallow and deep architectures, common losses or metrics and other relevant information will be described.

## 2.1 Feedforward vs. Recurrent

In Neural Networks (NN) there are two basic architectures depending on the direction of data flow: feedforward when the information runs only from the input layer to the output layer and recurrent when the information can, in some stages, run backwards. The former is the traditional approach, generally used to deal with a single input to predict one output. The latter, has been used to process data that has some kind of sequential nature. For example, recurrent networks are commonly used in natural language processing, where a word is used as input in every time step to end up forming a whole input sentence. In the case of speech recognition, audio is fed sequentially so the network can deal with every instance at a time but also use the whole context to better perform the task. In summary, the key idea for using a recurrent neural network is that information from the past might be crucial to solve the present task.

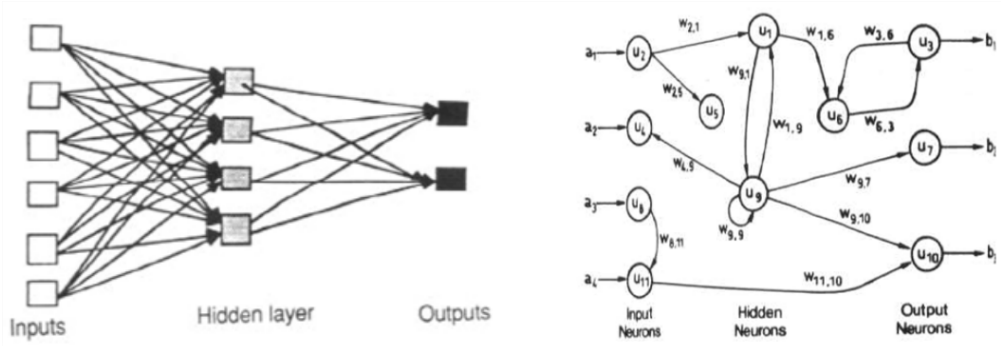


**Figure 2.1:** *Left*: Feedforward model. *Right*: Recurrent model, which includes data flow within neurons in the same layer, to previous layers and from neurons to itself. Source: [133]

RNNs allow for connections among neurons present in the same layer or even backwards

between layers, enabling information to persist through time steps. That is the key element that separates feedforward NNs from recurrent models. In particular, the difference is only found in the hidden layers, as input and output layers don't generally use recurrent connections. However, the most common case is neurons that have connections to themselves. These special neurons have an inner state that is updated every time step by combining the new input and the state from the previous iteration. Thus, the neurons build a representation of the data formed by potentially very long sequences, allowing for sequence learning. A feedforward model that needed to model such temporal data would need to use a big and complex architecture as it would have to input all elements at once. On the contrary, RNNs are capable of dealing with sequential data by using simpler models, as a single neuron is capable of remembering and modeling the past. In fact, a RNN may be explained as a feedforward network that shares parameters among layers, every one of which produces an output. When a simple RNN is posed in such fashion, it is said to be *unrolled*. Unrolling a network simply consists in unfolding it in order to remove the cycles in the model graph. It is a metaphorical term used to describe the fact that a recurrent model is equivalent to a feedforward one replicated over time.

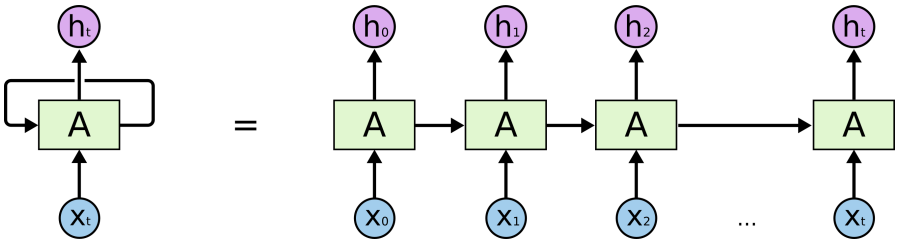


**Figure 2.2:** Simple display of the equivalence of a RNN and a FNN after unrolling. Source: [82]

## 2.2 Long Short-Term Memory (LSTM)

LSTMs [43] are a particular type of RNN. The main difference with a typical recurrent network is the complexity of the cells that are used to build it. This allows this type of networks to model not only short term dependencies but also longer ones (which common recurrent networks have problems doing [9]). Similarly to other RNNs, LSTMs contain an inner cell state that is updated by combining new input and previous outputs (also known as hidden states[1]), in order to make information persistent through time. However, in LSTMs different gates will influence how the state information will be modified, allowing for a finer management of memory. A basic LSTM cell is composed of three gates: input, output and forget (see Figure 2.3). First, the forget gate ($f_t$) decides which information from the cell state ($c_t$) is kept by looking at the new input[2] ($x_t$). Next, the input gate ($i_t$) selects which information from the input is going to alter which data contained in the cell state. Finally, the output gate ($o_t$) is in charge of determining which elements from the inner state are used as output ($h_t$).

Other variants of LSTM also use the cell state as relevant data to alter the way different gates behave, known also as cells with peephole connections. For example, the Gated Recurrent Units

[1]The hidden state used as input for a recurrent cell can come from a former layer or the LSTM cell itself in a previous time step.

[2]Here input is used to refer to new elements of the actual input sequences and previous hidden states.

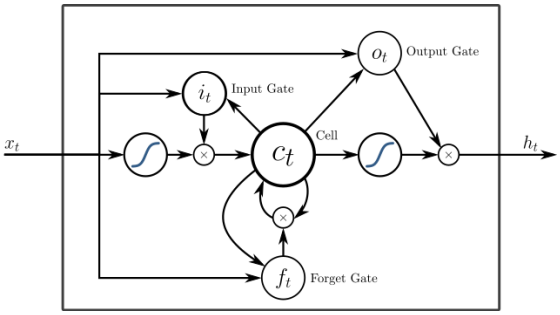


**Figure 2.3:** Diagram of an LSTM cell. Source [41]

(GRU) [14] are an alternative to LSTM cells that merges cell state and hidden state into a single element, hence not needing an output gate and reducing the number of parameters. In other words, the inner state is also the output of the cell. Also, instead of using input and forget gates, it uses update and reset gates which are combined in different ways. LSTM networks have been used on a wide variety of learning tasks, such as speech recognition, machine translation, caption generation for images or for recognizing actions and generating natural language descriptions of videos to name a few. Finally, it is important to highlight a special kind of LSTM as some of the models for video prediction use it: the Convolutional LSTM. It is an architecture proposal that naturally arises when working with video sequences. A Conv. LSTM is a type of cell that can be used as the normal LSTM to build RNN models but their gates are convolutional filters (see section 2.6. Networks built with these cells are able to simultaneously capture time and spatial variations with a single element.

## 2.3 Deep Learning

Deep Learning, also called hierarchical or representation learning, are a set of ML techniques. These architectures are different from shallow models principally in their topology: deep models use several hidden layers to build more complex representations of the data. Traditionally, the most common procedure was to work with shallow networks that included one or two layers. This was mainly due to the absence of effective procedures to successfully train deeper models. However, advances made during the present century brought new algorithms to train these models: unsupervised pre-training [10] or new regularization functions [109] allowed to end with the vanishing gradients problem that kept deep models from learning proper data representations. Thanks to that, nowadays it is very common to use deep architectures to solve virtually any ML problem. The main reason to do so is that deep models learn better representations as each layer specializes in creating an abstraction of the data based on the previous layers. Thus, low level layers will capture low level information of the data with which subsequent layers will build more abstract and useful representations. In summary, deep learning helps create complex and structured abstract representations of the data that embody the necessary features for solving the task that the model is being trained to solve. These representations are in general difficult to explain or understand, but they serve their purpose.

## 2.4 Variational Auto Encoders (VAE)

Autoencoders (AE) (first seen in [52]) are a class of models trained in an unsupervised manner to learn useful representations of data. They are composed of two main parts: an encoder, which maps the input data to a latent vector representation, and a decoder in charge of reproducing the input data based on that. This is considered an unsupervised task as the desired output is the same as the input, which avoids the need for supervised labels. Autoencoders have shown to be very effective for learning proper data representations needed for reconstructing the data. This is due to the fact that in order to solve that problem, the encoder has been forced to represent certain patterns in the data in such a way that allow it to be later reconstructed. Those learned representations may be later used for other tasks. For instance, trained encoders are often employed to perform supervised tasks with few training examples after they have learnt necessary latent representations. This also shows how knowledge acquired during different tasks may be exploited to solve other problems (as was mentioned in section 1.6).

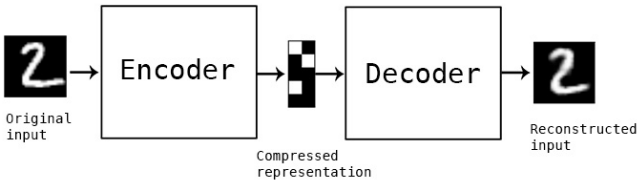


**Figure 2.4:** Diagram of a simple Auto Encoder. Source: [16]

The idea of transferring representations learnt by an autoencoder to other tasks works also in the other direction. Instead of using the encoder to extract useful features which can be used for a classification task, Variational AEs (VAEs) focus on using the decoder to produce new unseen data from a random latent representation. The main idea behind VAEs is to explicitly model the distribution of the input data (its mean and variance) in order to be able to sample from it. There are other types of autoencoders, such as Denoising AE [128], but the basics for them are the same with slight variations.

## 2.5 Generative Adversarial Networks (GAN)

GANs were first proposed by Goodfellow et al. [35]. They are a generative model for data based on competition. The method involves training two models at once: a generative model $G$ and a discriminative one $D$. These two models perform a confronted minimax game where $G$ tries to produce data as real as possible and $D$ tries to tell apart real from generated data. Even though $G$ could be any generative model and $D$ any type of classificator, when both $G$ and $D$ are neural networks, the model is called Generative Adversarial *Networks*. This particular case allows training using successful learning methods (backpropagation and dropout) for deep architectures. Thus, removing the need for Markov chains (for sampling) and overcoming some of the problems that arise when training deep generative models. While other generative models, such as VAE, try to do is to perform maximum likelihood estimation on the data distribution. This may sometimes result in huge probabilistic intractable computations. Instead, what GANs do is sample from simple distributions (such as random noise) and learn a transformation from

that to the more complex data transformation. They present some difficulties for training and converging and an extensive literature has been written about this. Recently, a variation of GANs called the Wasserstein GANs [4] showed promising results on solving some of the problems that arose when training these models, in particular regarding the training algorithm based in rounds (first train $G$ with fixed $D$ for some time steps, then the other way around) and the proper use of the discriminator loss in order to train the generator.

## 2.6 Convolutional Neural Networks (CNN)

CNNs (first seen as such in [60]) are a particular type of neural networks where every cell represents a filter (also known as kernel) that is going to be applied to the input data. Although CNNs have been successfully used for different types of data, such as text or audio, they are most commonly used for images. In this section only CNNs for images are considered, as they are the focus of the CNNs used for video prediction. Despite this, CNNs applied to other types of data are very similar and are not far from what is described below. A convolution is, fundamentally, applying a kernel to a whole image by sliding through it. In other words, given an image and a filter (which will generally be little and smaller than the image) a convolution produces a new image where every pixel results from the filter application at a particular position in the original one. In order to use a filter, every value in it is multiplied by a pixel in the image, adding the result and normalizing. Thus every pixel in the convoluted image will be the result of a linear combination of neighbouring pixels in the original one.

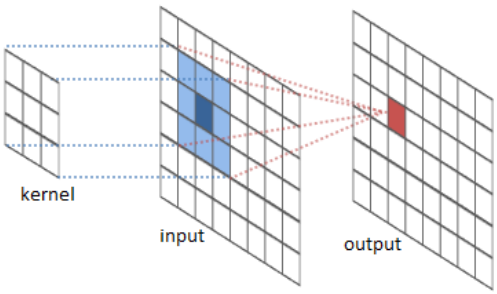


**Figure 2.5:** Diagram depicting a convolution. The kernel will be slided over the input image and produce the resulting convolved output. Source: [81]

A convolutional network is, then, a NN that applies several filters to an image at every layer. These filters have usually a square shape and are applied by moving the filter over the image a specific distance every time (stride). A particularly interesting feature of this type of networks is feature discovering and hierarchical representations of the data when used in a deep setting. In other words, they do not depend on manually extracted features but build their own instead, which enables the model to use whatever features present in the data work best for solving the task at hand. This allows for CNNs to directly handle raw and big dimensional data. Because the filters are small and shared across the whole image, high resolution pictures can be used without the number of parameters of the model exploding. Each layer will use the convolved images produced by the previous layer, being the lower layer the only that has direct access to the raw data. Furthermore, subsequent layers will all build new more abstract representations

based on what former layers have built. In practice, the first layer of a CNN will form filters that look for specific types of edges (horizontal, vertical, corners, etc.); a second layer will build up on that detecting simple geometric figures (curves or circles, rectangles, triangular shapes, etc.); a third layer could detect more specific forms (eyes, windows, mugs, etc.); final layers find more complex figures as the combination of many of the more simple filters (cars, faces, ducks, etc.). An important CNN feature to highlight is the fact that these filters are trained from scratch, without using any prior knowledge to condition the learning process. In conclusion, it would seem that convolutional networks are drawn to produce useful filters necessary to extract meaningful information from visual content.

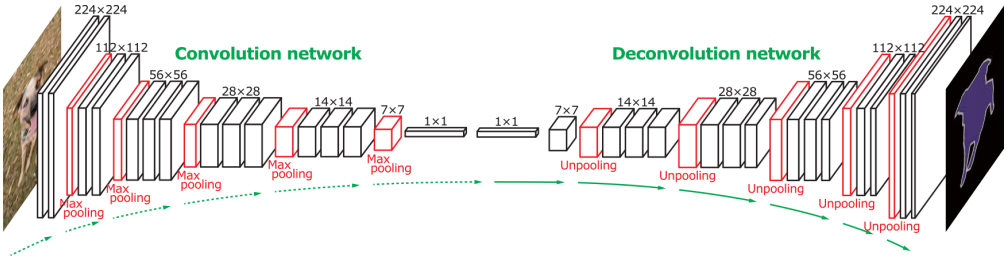


**Figure 2.6:** Architecture of a Convolutional AE. The *left* side of the model depicts a typical CNN with convolutions and pooling layers to reduce dimensionality. Note how the number of channels increases through time as several filters are applied at every layer. The *right* side shows a Deconvolutional network that works in the opposite direction: it tries to build an output with data dimensionality based on feature vectors. Source: [78]

Convolutional Networks are not only formed by convolutional layers. There are also other types of layers that help the model better learn and reduce the number of unnecessary trainable parameters. There are two main types of layers that help: normalization and pooling layers. The former are used to avoid losing control over the values present in the convoluted images after several filters have been applied. The latter are used to reduce the amount of parameters needed, as they reduce the size of the data by taking only the most representative pixels. For example, a max-pooling will take the pixel with maximum intensity among a set of neighbouring pixels and keep only that one to form a reduced image. There are different pooling types, however, average and maximum are the most commonly used. Other way of reducing dimensionality of the data is using a stride (displacement of a filter when going over an image) bigger than one. This makes sense as a stride of one would be a sliding window where the filter is calculated for every pixel and results in an image the same size. Using a stride of two (or bigger) leaves out the filter application to every second pixel, thus reducing the size of the resulting convolved image.

The convolution process in a CNN is the procedure of generating more abstract and less dimensional representations of the input. CNNs have been usually employed to perform classification tasks by adding some simple layers at the end. However, the opposite is also possible, and CNNs have been used to generate data, such as images, from abstract representations. Transposed convolution (also known as fractionally strided or simply upconvolution or deconvolution) and unpooling layers can be used in order to transit from a small latent representation to a big dimensional specific representation as an image. A deconvolution layer carries out a regular convolution but reverts its spatial transformation. In other words, what these layers do is to perform normal convolution on a padded input (see Figure 2.7). The input is spread out to match

the desired output resolution and filled with zeroes (generally) or it is simply padded externally. A normal convolution is then applied in order to obtain the desired "deconvolution". This results very useful when building generative models of images that map latent representations of data to real RGB images or to build Convolutional AEs.

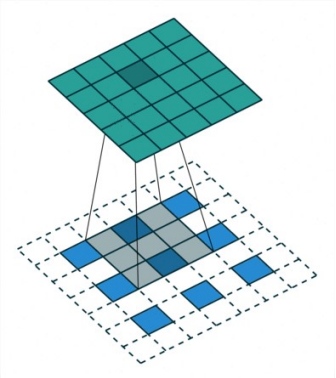

**Figure 2.7:** Example of a deconvolution. The input (blue) is first scattered and then the convolutional process is applied to produce and output which dimensionality is bigger (green). Source: [26]

## 2.7 Optical Flow

Flow estimation is a method that has been traditionally used to estimate motion from images. In particular given two images, an optical flow estimator tries to find out the position of one pixel in the other image. The result of optical flow estimation is a vector for every pixel in an image representing motion. There are multiple methods for obtaining optical flow, but it is not the purpose of this section to go into detail, as not many methods in the SOTA for future frame prediction use it.

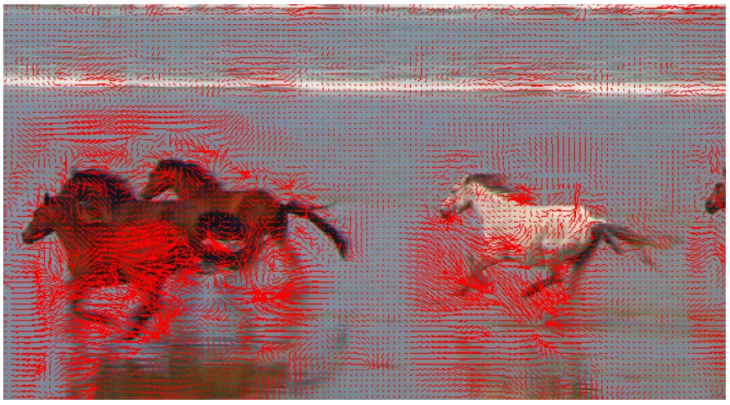

**Figure 2.8:** Image depicting Optical Flow estimated over an image. Each red line displays movement for a specific patch on the image. Source: [100]

Having an image and a flow has allowed for warping the original image and predict a future frame. Basically, the predicted motion between two frames is extrapolated to the future in order

to displace the pixels from the last image to produce a new one.

## 2.8 Loss Function

In ML, a loss function is a function used to drive the learning procedure in the correct direction so the parameters of the model adjust to better solve the task it is being trained to solve. In particular, a loss consists in a function that will compare the output produced by the model against the actual expected output. Then, the parameters are adjusted respective to that difference as to make the model closer to the correct response. This is done normally by using an algorithm known as backpropagation through time (BPTT) [96], which is based on the idea of gradient descent. In other words, it drives the weights as to diminish the error gradient produced by those weights. Next a brief description of the most common losses used for video frame prediction. In them, the generated next frame ($t_i$) is compared against the next ground truth frame ($y_i$) to produce an error generally (except for the adversarial loss) at pixel level.

### L1 and L2

The most basic difference. These losses make a pixel wise subtraction of both images. These loss function are very similar to one another and the only difference lies in squaring the difference or not.

$$L_1(y_i,t_i) = \sum_{i=0}^{n} |y_i - t_i| \qquad L_2(y_i,t_i) = \sum_{i=0}^{n} (y_i - t_i)^2 \tag{2.1}$$

### Mean Squared Error

Similar to $L_2$, this loss is based on the squared pixel wise difference of both images. The main difference is that MSE averages the result. However, many regard $L_2$ and MSE as the same metric and reference them using both names (see below).

$$MSE(y_i,t_i) = \frac{1}{n} \sum_{i=0}^{n} (y_i - t_i)^2 \tag{2.2}$$

### Euclidean Distance

Again, very similar to $L_2$, but using the squared root on the error summation.

$$ED(y_i,t_i) = \sqrt{\sum_{i=0}^{n} (y_i - t_i)^2} \tag{2.3}$$

The main difference between $L_2$, MSE and this loss, are the magnitude at which they represent the error, thus encouraging different results. It is also important to highlight that sometimes the authors of different models for video prediction refer to these three metrics with disregard to the actual formula. For this reason, in order to get specific detail on the loss they used it is best to revise the original publication.

### Cross Entropy

Cross Entropy (CE) has been traditionally used as a measure for optimal number of bits to represent an approximate probability distribution (different from the actual one). CE as an error function is generally used for logistic regression to measure difference among probability distributions, in particular the real ground truth and the approximated predictions in classification tasks. Comparing images with this loss is done by assuming that images represent a probability distribution. The generator would be trying to approximate the ground truth. Given that, CE can only be used when working with grayscale images (one channel), which pixel values are equivalent to a probability (0 to 1).

$$CE(y_i, t_i) = \frac{1}{n}\sum_{i=0}^{n} t_i \log y_i + (1 - t_i)\log(1 - y_i) \tag{2.4}$$

Many models for video prediction use normalized versions of the CE loss. The most common are binary (limited to 0 and 1), or Softmax and Sigmoid that basically consist in applying those activation functions to the loss.

### Adversarial

GANs are often used as a loss. When considered as such, the discriminator network is thought of as a metric that tells apart good from bad generations, thus guiding the training to lower regions of the error gradient. This loss can take many forms depending on how it is used and will be different from model to model. In general the idea is to use the gradient discriminator to guide the generator towards weights that produce more realistic results, hence fooling the discriminator.

# Chapter 3
# Overview

## 3.1 Historical outline

The history of Artificial Intelligence is about human understanding and digital replication of intelligence. Through the evolutionary process of ML many models have been proposed that resemble or are explicitly based on the behaviour of the human brain. Video prediction and the neuroscience theory of predictive coding is not an exception to this. In fact, one of the first attempts to predict visual stimuli dates back to 1996 when Softky [105] proposed a model for unsupervised pixel prediction inspired in the predictive coding theory. For this purpose, they used a Spiking Neural Network for unsupervised Hebbian learning to predict the location (in a grid) of a moving pattern in the future. The architecture is based in three neural bidirectional layers. The generated predictions are used for biasing the neurons through special synapses and also for comparing against the actual future in order to learn patterns of failed predictions.

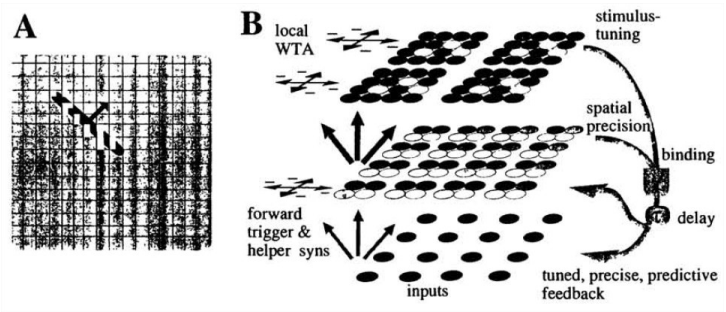


**Figure 3.1:** Spiking Network proposed by Softky to predict activations of pixels in a grid. Source: [105]

Even though this architecture is too organic (regarding its inner behaviour) as to scale up to model the complexities of real videos, it demonstrated that predictive coding could serve as an effective tool to train a model in an unsupervised manner. After this no other similar attempts have come up in the SOTA as videos were too dimensional for the available models at the time (both at theoretical and hardware level). Sequence modeling and prediction was left mainly for tasks in Natural Language Processing (NLP). The advances in that field will later highly influence the first attempts to create similar architectures for video prediction (e.g. [93, 110]). It was not until 2007 when Sutskever et al. proposed the "Temporal Restricted Boltzmann Machine" (TRBM) [112], its follow up Recurrent TRBM (RTRBM) [114] and the Structured RTRBM (SRTRBM) by Mittelman et al. [72] that video modeling architectures came

back to the CV scene. They used an unsupervised generative setting to denoise sequences of small bouncing balls (among other experiments). With this proposal they set the ground for models able to build valid representations of videos and to generate frames based on some other input frames. However, this was not yet *future* frame prediction.

Around those years, N. Verma published several models (e.g. [123, 122]) with different colleagues. Most of these models attempt using simple networks like ANN or RBFNN, Autoregressive models and different Fuzzy rule-based systems to predict future images. Each model generally predicts specific pixel values in the next frame. A composition of such models creates a complete frame predictor. They use raw intensity values or velocities for every pixel extracted with optical flow estimation from the last two frames as input. The predicted frame is then arranged combining the predictions for every pixel. The models are usually tested using single sequences of a fighter plane landing. In 2012 Kar [48] presented a simple ANN model inspired by Verma. He applied dimensionality reduction to the input images (PCA and Isomap) and trained one separate model to predict every channel in the future frame. The input was the concatenation of all the dimensionally reduced frames (they performed experiments with a varying amount of input frames, from 1 to 10). The desired output was the next image (channel) in the sequence. Even though these models lack of the complexity needed to model complicated motions of objects in videos they serve as proof that predicting just the next frame can be challenging but solved nonetheless even using small pixel neighbourhoods and getting a good enough approximation. However, as it will be detailed in the following, a proper architecture for future frame prediction should be able to account for elements in the whole scene and have the ability to generalise to other environments.

Briefly after that, Sakaino [97] proposed a model to "predict near-future spatio-temporal image changes using fluid like image sequences". He was going to predict future weather radar images based on Optical Flow and a physical model for fluids. Given two frames he would estimate pixel displacement and then predicted the next flow with the advection equation (flow extrapolation) and Navier-Stokes equation (to describe the motion of viscous fluid substances). Even though this was one of the first actual approaches to predict future frames given a video sequence it lacks any generalization capabilities as it depends on the fluid equation. However, many current SOTA use the advection equation on Optical Flow estimation as a baseline against which to compare their models.

## 3.2 Current framework

In 2014 the first deep models for video prediction appear and the systems that can account for high dimensionality allow for video prediction models to spread. Since then up to the present day more than 30 different architectures have been proposed. However, "frame prediction" poses only the basic concept of the task: given a sequence of frames a future frame architecture has to accurately generate one or more frames that follow it. Nevertheless this is not as straightforward as it may seem. Predicting the future is a multi-modal problem, in the sense that for a given past many possible futures may be possible. Even if a model correctly learns to represent real physics and understands motion details it may predict a future which is not on the actual sequence.

The objective of this section is to highlight some of the principal design choices that may arise when building a model of this characteristics and point out trends in the final decisions. First of all it is important to consider that some proposals use single frame video prediction. As

it has been stated above the problem considered here involves frame prediction given a *sequence* of previous frames. In this sense no model that generates the future given a single frame will be considered. Furthermore, given the complexity and multi modal nature of the problem of using past structure and motion from frames to generate the future, using a single frame to condition the prediction should not be enough to accurately produce the next frames and hence it is considered as a different problem. However, some proposals show some interesting capabilities and structural elements that could be adapted to frame prediction as posed in this work. For this reason some of these models will be detailed at the end of this section in section 3.10. It is also important to mention here the case of Liu et al. [62], Zhao et al. [141], Goroshin et al. [39] and Zhou et al. [142] who used only two frames in order to generate the next one. It may seem like a limited system, however it at least accounts for the transition from one frame to the next in order to try next frame extrapolation; also, some of them proposed other architectures that did use more frames as context before predicting. Other models such as Ranzato et al. [93] consider also two frames to condition their recurrent predictions; or the example of Lotter et al. (2015) [63] who started to generate one prediction at a time after the second frame entered the system. These models do have been considered for comparison. In the present section many details of these chosen models are layed out, but the overall architecture for every proposal can be seen in chapter 4.

## 3.3 Models architecture

Given the sequential nature of future frame prediction, using a recurrent model to account for the time dimension may seem as a straightforward solution. Nevertheless, this is far from what it may seem. Even though most proposals do indeed use some sort of recurrence (generally through LSTM layers) it is also many the attempts that have tried to model future frame prediction without the use of such architectures. Another issue to be addressed is the amount of input and output frames and how are they going to be handled. When combining these two problems other challenges arise, as it will be detailed briefly, especially with autoencoder architectures. These complicate (and enrich) the scene, by allowing for all the aforementioned issues to be dealt with differently in encoder and decoder.

**Completely non-recurrent models** have a fixed amount of input and output frames established when the model is designed and which cannot change after training. There are two main ways of building such a model: a fixed architecture dependent on the number of frames which are all fed at the same time but independently or using a volume of concatenated frames as input (also fixed) which are fed as one single element (see Figure 3.2). The former is generally used with models that do not use many frames as input, generally two. This is the case of Zhao et al. [141], Goroshin et al. [39] and various of the models proposed by Zhou et al. [142]. In general these models have limited capabilities and are trained for specific purposes that do not need nor intend to generalize to a more complex frame prediction task. The other non-recurrent trend is to use a frame volume as input. This is the case of the feedforward model of Oh et al. [79], Klein et al. [53] and Amersfoort et al. [119] where a volume of input frames is used to predict a single future frame. Also Walker et al. [135][1] and Vondrick et al. [132] who used a volume to predict

[1] This model uses two parts, one that exhibits recurrence explained below and the one considered here that uses a volume of pose vectors to predict a volume of pixel level frames. This last feature was deduced from the diagrams

a volume in order to avoid the feedback problem described in subsection 3.4.1. Finally Liu et al. [62] and Mathieu et al. [67] performed experiments on both volume to frame and volume to volume.

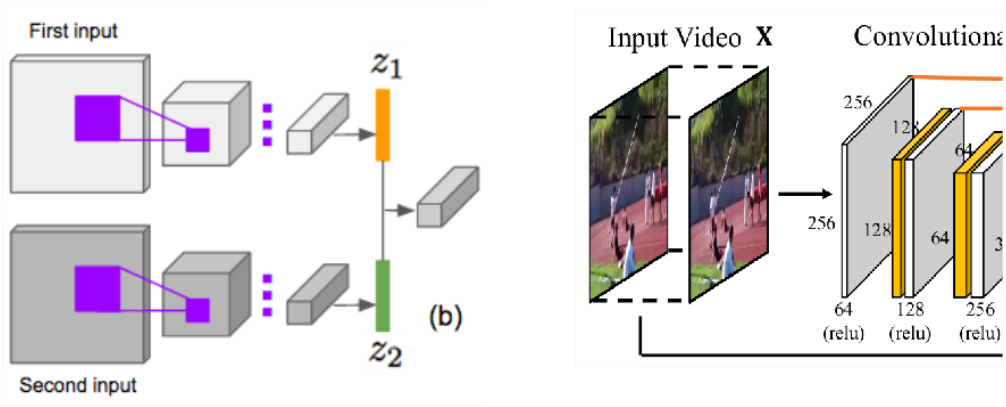


**Figure 3.2:** Examples of non-recurrent models. Left: Two frames are used as input but they are fed into the model separately. Source: [142]. Right: Various frames are used as input and fed simultaneously to the model as a concatenated volume. Source: [62]

**Partly recurrent models** are generally autoencoder architectures that incorporate recurrence only on one part of the model: either in the encoder or the decoder. Regarding the former, frames are fed sequentially to the model to build a latent representation of the whole input and then produce a single prediction. The latter generally employs a set of frames to obtain the input features and then the decoder is recursively applied to generate future frames at every time step. This kind of models allow for predicting further without having to use the encoder again (which seems to be a desirable quality as will be discussed later). Encoder recurrent models have commonly some convolutional layers followed by LSTM layers, some examples of this include the recurrent model of Kalchbrenner et al. [49] and possibly Liang et al. [61][2]. Decoder recurrent models include the sequential model of Zhou et al. [142] which uses a single encoded input frame and then unrolls the decoder to produce the predictions. Other models that behave in this way (by unrolling the decoder after all input has been fed to the system) are considered below on the fully recurrent models. Also, the architecture by Michalski et al. [70] could be considered to fit (despite its non explicit recurrence) in this category as their model allows for continued predictions once all the input has been used with no further need for new input. They use a pyramid of transformation predictors which may be kept applied to the feature level transformations to further deform input frames into the future.

**Recurrent models** are those that employ some sort of recurrent connections to model the temporal dimension of video sequences. They represent the bulk of the literature for future frame generation. Most architectures use LSTM or Convolutional LSTM layers for modeling their ability to handle relationships among frames in a sequence. The main contrast with partly

but not explicitly stated in the paper.

[2]Even though it is not explicitly established in the paper, the fact that the motion encoder has some Convolutional LSTM layers may suggest that a frame is fed at every time step. They state that the model is conditioned on 10 frames before producing 1 single future frame. On test time, they report to have predicted up to 5 frames, which sounds like using predictions as feedback to generate further. However, this was also not said in the paper.

recurrent models is the use of such temporal components in both parts of an autoencoder architecture. There are different issues to consider in recurrent models. Usually they use an input frame and generate the next one in the sequence at every time-step, hence have to run the whole architecture to form a prediction. This approach includes most of the literature proposals including Ranzato et al. [93], Lotter et al. (2015) [63], Pătrăucean et al. [91], Liang et al. [61], Denton et al. [24] and probably Brabandere & Jia et al. [20][3]. Kalchbrenner et al. [49] and Lotter et al. [65] use this same approach but start generating a prediction at a time after a certain amount of input frames have been encoded. Also, even though Oh et al. [79] included recurrence only in the encoder, they used first 11 sequential frames to condition the encoder and from there they input and output one frame at a time.

In a similar fashion, Cricri et al. [18] use a window input approach where 10 frames are used to predict one. To predict further they shift the window and take the generated frame as new input (hence using 9 ground truth and 1 prediction, and so on). This is similar to the approach by Amersfoort et al. [119], despite not using a recurrent model (plain CNN). They use a volume of among-frames transformations as input in order to predict the next transformation (which applied to the last frame in the sequence will generate the future). In order to create a recurrent-like approach they take the predicted transformation and shift one place the input previously used in order to predict the next one.

Finally, other interesting recurrent proposal is first applying the encoder on the whole input sequence and then unroll the decoder for multiple predictions. In this sense the model has less computational requirements as it avoids using half the model in every generation. This is the approach used by Srivastava et al. [110], Shi et al. [103], Walker et al. [135][4], Villegas et al. (b) [127] and Oliu et al. [84]. As with Michalski et al. [70] mentioned above, this allows the models to predict potentially any desired length in time during testing without using new input.

## 3.4 Training details

This section highlights some key issues to pay attention to when training a model for video prediction. In particular, some matters that may arise and tricks to better train such models. The three more used techniques for improving the training procedure are the use of feedback predictions during training if necessary, curriculum learning or scheduled sampling to progressively use longer sequences for training and pre-training the model for reconstruction.

### 3.4.1 Feedback predictions

Disregarding the recurrent nature of the model, many proposals in the literature base their ability to predict further in time in using predicted frames as new input for the models, an approach common in natural language processing tasks. However, relying in already predicted frames is a problem as artifacts appeared during the first prediction will generally get worse over the next ones. In other words, errors tend to accumulate over time. This causes generated frames to degrade over time as they get further away from the last seen ground truth input.

[3] No explicit mention to the amount of input frames they use at every time step. Even though it is possible that they use two frames as input, every time step produces a single future frame.

[4] Again, the fact that they perform the recurrent predictions of pose vectors in this fashion is deduced from the figures but not explicitly stated in the paper.

It is important to note that most recurrent models generate only one frame at a time, and are generally trained to predict up to a fixed length using new ground truth frames. However, this makes sense only partially because the models are not used to seeing their own predictions, which is only done during testing. This broadens the gap between training and testinting results. Using more ground truth during training possibly enables the model to build abstract representations of the input based on more real frames, thus allowing for a latent encoding that is able to better summarize a sequence. However, recurrently predicting one frame but not training the model to work with its own predictions at any point causes the model to easily accumulate prediction errors in the long term. This results in very good generations near the ground truth but a quick degradation of the results as the predictions get further away. There should be no need of using more ground truth frames during training than the model is going to see during testing. All ground truth necessary is shown to the network as initial conditioning and as supervision at the other end. For example, Srivastava et al. [110] proposed a conditional variant of their model where the decoder is conditioned on the frames it has already generated but they only use it in such way during testing. In train time they condition the decoder on ground truth frames. In contrast, Mathieu et al. [67] reported that even though predicting a volume was more challenging for the model and that more parameters were needed, the quality of the predictions produced in this fashion outperform those generated using volume to frame with feedback.

For this reason some [79, 29, 63, 18, 119, 126] have proposed to use a fixed amount of ground truth frames and then use the model predictions as feedback also during training. Doing so has shown to help build a system that is more robust to errors and to their propagation. During test time it will have learned to detect specific features of the data that it must not propagate. All the models who used feedback predictions to further generate during training reported improved results with respect to use feedback only at test time. However, Villegas et al. (b) [127] went a step further and stated that employing feedback predictions at all is a problem per se, and that using them during training is a patch rather than a solution. They propose a two fold alternative by avoiding to use them at all (as mentioned above) and by working at feature level (which will be explained in section 3.5 with more detail). Actually, in general, all the models able to do so [127, 70, 110, 103, 84] stay clear of the problem of feedback by keeping the predictions going after a given input. They further evolve the latent representations in order for the model to generate next futures. Thus, they eliminate the need of having to encode again an input with potential errors.

### 3.4.2 Curriculum learning

Curriculum learning or scheduled sampling is a training architecture that consists in progressively increasing the complexity of the task being solved. It was first proposed by Bengio et al. [8] and has been successfully applied to the task of sequence modeling [7]. Some models for video prediction have too suggested to use this strategy to better train a model towards predicting longer and sharper sequences. By first training a model to predict the following frame given an input sequence and gradually predict more frames into the future, the model generates more robust time representations. Once it has converged for a given number of frames the representations may be extended for a longer sequence. This was the approach taken by Michalski et al. [70] who reported "that starting with few prediction steps and iteratively increasing the number of prediction steps as training progresses considerably stabilizes the learning.". Finn et al. [29]

also employed this technique resulting in improved results. Likewise, Lotter et al. (2015) [63] fine tuned the model to predict further using its own predictions after being trained for a specific amount of ground truth frames.

In a similar fashion, some models have proposed to use a loss over multiple predictions in order to produce sharper results (see section 3.7). Oh et al. [79] proposed a combination of feedback predictions during training, curriculum learning and a loss over an increasing number of generated frames. They train the model to predict short term future frames and fine-tuned for longer sequences as the process converges. They use that setting with 1, 3 and 5 time steps (progressively increasing by employing feedback predictions) using a loss over the whole generated sequence. They too report this technique to have helped stabilize the learning procedure.

### 3.4.3 Pre-training

Finally, and in the line of gradual training proposed by the curriculum learning approach, some models have seen an increase in performance when pre-training for sequence reconstruction before performing frame prediction. Michalski et al. [70] and Zhou et al. [142] reported this procedure to kick-start the predictive model by providing useful initialization representations by just pre-training to reconstruct random input images. Walker et al. [135] can also be considered to perform this kind of pre-training as they use a composite model formed by different parts of separate autoencoders, one of which is trained for input reconstruction.

## 3.5 Prediction atom

Even though the most common approach is to directly try to model the future frame generation task at pixel level, many have proposed to deal with predicting transformations instead of pixels. Furthermore, recent approaches have suggested to avoid both at all and perform predictions at feature space. The key difference can be mainly found on where the loss is applied for learning. However, there are some other details worth commenting of the different choices that have been made regarding this aspect of frame prediction.

### 3.5.1 Pixel

As it has been said, the most used approach is to predict at **pixel** level. Most models compare the generated frames against ground truth performing some sort of image subtraction on the loss of choice in order to train the models. These models include [141, 110, 39, 103, 79, 67, 63, 65, 142, 18, 126, 61, 84]. Even though Ranzato et al. [93] predict the indexes of a patch dictionary and built the prediction at pixel level by averaging the predictions for various patches, they can also be considered in this class of models. A special mention goes to Kalchbrenner et al. [49] who actually predict individual pixels sequentially to model not only dependencies among previous frames but also to ensure that all the pixels from the current prediction are consistent with one another.

### 3.5.2 Transformations

Some authors have pointed out that the various artifacts and blurry predictions that generally result in generated frames by some proposals are caused by the fact that the whole frame must be

hallucinated. They argue that in most cases a big part of a frame in a sequence is unchanged or is the result of small patch displacements. In other words, they propose that a model should be able to account only for the differences of the future frame with respect to the past sequence. Then those differences can be applied to the last input in order to predict. With this, an architecture should be better able to predict as it has to account for less variability. Predicting motion may help the model to focus on understanding movement dynamics as it does not have to pay attention to appearance. Structure, appearance and in general low level details are already contained in the last frames to which the model has access as if it was short term memory. For this reason, a future frame generating architecture should only need to regard modeling the essential parameters as to transform it. There have been different approaches to work with **transformations** some of which use difference images or directly encode transformations (affine or optical flow). The models that use the latter are [70, 91, 29, 119, 62, 132, 61]. Even though many of these models still use a loss at pixel space to guide the training, the key issue is their ability to represent transformations as the basic component that the model has to deal with instead of whole images. Klein et al. [53], Brabandere & Jia et al. [20] and in some sense Pătrăucean et al. [91] use a model that predicts transformations in the form of filters which are then applied to the last input frame in a convolutional fashion (similar to the Spatial Transformer Networks [47]). Whereas Pătrăucean explicitly tries to model optical flow prediction, Klein and Brabandere are building a model that dynamically produces the filters for a convolutional layer to apply to the input in order to create a prediction. Finn et al. [29] actually predict multiple transformations for separate patches and then a dynamic mask is in charge of selecting which motion is applied to which pixels. Amersfoort et al. [119] use an Affine Transformation Extractor and worked completely at that level. They used affine transformations as input and compared the predicted motion against ground truth next transformations. In this sense they are using a loss at transformation level instead of pixels, which further reduces error propagation (less levels of the procedure may be responsible for error introduction).

### 3.5.3 Features

A dual approach between pixel and transformation level is the one proposed by Liang et al. [61]. They use a two stream decoder that produces a motion prediction and a pixel prediction respectively given a single feature representation. Afterwards both predictions are used to generate their equivalent in the other modality (a pixel prediction from the estimated motion and vice versa). The two elements in each branch are then used in a dual adversarial setting to ensure that the predictive model is able to account for both coherent motion and good pixel level predictions. Finally, recent proposals have suggested to remove the physical barrier and try to predict directly at **feature** level. All the models mentioned above with the ability to predict longer sequences once all the input has been encoded may be regarded as doing this in some sense. However, there are a couple of approaches worth of mention in this respect. Walker et al. [135] and Villegas et al. (b) [127] both manually designed that feature level and proposed a model for future frame prediction oriented to human motion. They used pre-trained pose extractors and propose an architecture for future pose prediction. A second system was later trained to generate frames from the predicted poses and the last ground truth frame. It is important to mention that the loss was applied at both pose level for the pose prediction and at pixel level for the image generation. Even though these models demonstrate high quality modeling of human motion,

they have trouble when predicting the movement of anything else. Denton et al. [24] try to surpass this limitation by letting the model build its own latent representation onto which to map. They use adversarial training to encourage the model to separately encode content features and moving features in the scene. Then predictions were carried out only for the moving elements in a similar fashion as it did for the pose prediction models described before. The predicted new representations are then applied to the content vectors to generate pixel predictions.

## 3.6 Explicit separation of appearance and motion

Following the line of argument of the architectures that proposed to work with transformations instead of pixels some have proposed to explicitly model motion and appearance separately. In a sense, actually, the proposed architectures that predict transformations which are then applied to the last frame in the input sequence are doing exactly that. In general, every method that predicts transformations instead of hallucinating at pixel level, or the case of Klein et al. [53], Pătrăucean et al. [91] or Brabandere & Jia et al. [20] who dynamically predict filters to be applied to the last frame, could be regarded as performing separation on appearance and movement. In some sense, the fact that the input frame is forwarded to a point of the network where the *predictive* effort has already been made "disregarding" the actual frame (more or less) accounts for some separation between visual content and transformations, weather it is a filter or a flow (predicting transformations).

As Walker et al. [135] stated, traditional "approaches try to model all the structure and scene dynamics at once, in unconstrained settings they often generate uninterpretable results". These models propose to handle appearance (color, intensities, objects structure, etc.) and movement separately to help the model better account for details of both visual elements. Mainly because this will make a more efficient treating of the model parameters as the learning is slightly more guided. Leaving these models aside, some architectures that perform this explicit separation are next detailed. Finn et al. [29] perform this separation by using a background/foreground mask that allow their proposal to copy pixels directly from the previous frame. Both Walker et al. [135] and Villegas et al. (b) [127] perform this separation by the explicit limitation of movement to the human pose vectors. Finally, on one hand, both Denton et al. [24] and Villegas et al. (a) [126] proposed to separately encode motion and appearance: the former uses an adversarial setting for encouraging motion vectors to include no content at all whereas the latter used difference images for a recurrent motion encoder and a single frame for the content encoder. Their main difference is that while Villegas uses LSTM for motion encoding, Denton does so by extracting motion vectors from single frames. On the other hand, Liang et al. [61] proposed the opposite approach by having separate motion and content decoders as has been explained before.

Finally, residual connections between encoder and decoder have shown some capabilities to help autoencoder architectures disentangle different abstraction levels of the data. This is possible due to higher levels in the feature hierarchy not having to account for low level details. For this purpose many models use skip or residual connections that copy some version of the data present in the encoder to the equivalent decoder level. Generally after a convolution or residual block that *decides* which parts of the information get copied. This is the case of [29, 62, 135, 126]. Lotter et al. (2016) [65] proposal contains multiple recurrent blocks at frame level, each minimising the discrepancies from the previous block prediction. Cricri et al. [18] used recurrent bridge connections between every layer in the encoder and decoder. Finally, Oliu et

al. [84] proposed a recurrent autoencoder model that allows for state sharing between encoder and decoder, thus highly reducing the amount of parameters needed to account for the same behaviour that bridge connections provide.

## 3.7 Blurry predictions

Many have pointed out that typical losses used for modeling video sequences such as $L_2$ or MSE produce blurry results. Due to the high uncertainty in the task of future frame prediction, such losses respond to the many possible futures averaging them and hence producing blurry frames. This may actually compromise the learned representation, as a deterministic network will learn to average between all the plausible predictions instead of producing a sharp specific future. These metrics produce blurry results because they are a pixel-wise distance between possible frames. Thus the average frame is the one that minimizes the distance to any possible future and consequently the error value. Amersfoort et al. [119] and Liu et al. [62] reported that MSE produced blurry predictions even when using it as a loss in transformation space, causing the model to produce an averaged motion.

Different proposals have been made to deal with this. Ranzato et al. [93] proposed to predict image patches from a dictionary instead of going for pixel level, but as every pixel was the average result over multiple predictions they ended up with more uncertainty. Goroshin et al. [39] propose using a loss at feature space and a new pooling and regularization operators to help the decoder choose among multiple modes. Kalchbrenner et al. [49] addressed the problem by making the prediction of every pixel dependent on the already generated pixels for the current frame and previous ones. With this framework they ensure that the modality of every pixel is consistent with one another and the already predicted frames. However, having to predict a pixel at a time results in a model which takes long time to predict a single frame.

Some have proposed to use the training loss over multiple frames instead of single frames to encourage predictions to be consistent time wise [103, 65, 20, 18]. This accounts for the error caused by feedback predictions being a source of error magnification. The further from ground truth predictions are the more difficult it is to account for uncertainty. So many have suggested to use a loss over more than one prediction in an attempt to reduce long term blurriness. Also, some of the aforementioned techniques for better training frame prediction models (reconstruction pre-training and curriculum learning) and unrolling the decoder after all input has been encoded all have shown to help reduce blurriness both at single frames and over time as they are especially good at providing error propagation avoidance.

Finally, one of the most used techniques to generate sharp predictions are GANs. Many proposals have used an adversarial loss as a way to ensure realistic frames. By using a discriminator network that is trained to tell apart real frames from generated ones, the predictive model is encouraged to produce frames that are less blurry (closer to a specific possible future) and therefore more realistic. The adversarial setting has been applied independently to single frames (which ensures sharp looking images) [142, 61, 127] and to whole sequences [67, 63, 132, 126, 135][5] to ensure consistency and coherence between ground truth and generation and among generated frames. In fact, this last proposal generally uses a concatenation of both ground truth and generated frames to be fed into the discriminator. It is also important to note that adversarial losses

[5] However, [67, 135] did not explicitly stated so in the paper, it was inferred from their architecture and figures.

are not generally used alone and it is widely common to employ them in composition with other pixel wise losses that help guide the generator learning process.

## 3.8 Extra features

As stated in the introduction, one of the many applications of video prediction could be to use it as a support system in agents that have to interact in real environments. A good visual system able to predict future states of the world could enhance planning capabilities of the agent. In this sense, an interesting area of research would be that of future frame models that are conditioned also on actions or states of such an agent. For this purpose, Finn et al. [29] proposed the robotic Pushing Dataset. It contains sequences of robotic arms pushing objects in a restricted space. Each frame is coupled with the robot state and the desired action to be performed in the next frame.

**Figure 3.3:** Frame examples of the robotic Pushing Dataset. Source: [28]

Both Finn et al. [29] and Kalchbrenner et al. [49] have proposed action and state conditioned models that learn to produce predictions based not only on the input frames. Both demonstrated that the model learns relationships between action and desired future. For the same given input frames, different action conditioning produced different next frames in accordance with the used desired action.

Other action conditioned models include the proposal by Oh et al. [79] that predicted frames of Atari games based on the next action chosen by the player. The model is able to produce the correct future for many time steps after ground truth frames are no longer provided based only on the action. Nevertheless, videogames are very constrained and it would be difficult to the model to account for more complex real environments.

Finally, the models by Walker et al. [135] and Villegas et al. (b) [127] that use pose vectors extracted with pre-trained models as the atom for their predictive system could be regarded as using some extra features to help form the predictions.

## 3.9 Measuring results

Generative models for images present one particular issue that other ML problems do not need to account for: measuring results is a task that can result as complicated as the problem itself. Future frame prediction is one of such problems. In general, most models use the training loss itself as a metric to report how good the model is. In this sense, many different error functions (loss or not) have been used in by the analysed models to report their results: L2: [110], Euclidean: [53, 135], Standard Error: [53], MMD: [135], RMSE: [93, 141], DSSIM:

[84], Perplexity: [93], Entropy: [93], Cross Entropy: [103, 91, 49], Binary Cross Entropy: [20, 18] and Negative Log-likelihood:[49].

This causes the comparison between models to be a huge mess, as no common ground seems to be found. However, the most used metrics used (with a big margin) are MSE [134, 79, 63, 65, 142, 61, 84], PSNR [67, 29, 142, 62, 126, 61, 127, 84] and SSIM [67, 29, 142, 62, 126, 61]. Also, some have proposed to measure only moving regions arguing to "only analyze the part around the person and remove the influence of accumulated variations of the background" [134, 126]. However, if a model is distorting the background when it should not, it is probably not a good future frame prediction model.

Nevertheless, all this metrics account for how different is the generated prediction compared against the actual ground truth future. As it has been mentioned before, video prediction is a multi modal problem and a model that has correctly learned the physics of movement and predicts a plausible future may be incorrect regarding the actual future. For example Vondrick et al. [132] pointed out that their proposal was focused on generating a plausible future more than actually matching ground truth frames. For this purpose the Inception Score for generative models was proposed in [98]. It consists on training a traditional model for some classification task. For example train a model that given a sequence of frames classifies it as pertaining to a specific human action. Then, test the model on ground truth sequences as an upper bound. Finally compare the results using artificial sequences produced by the generative model in question to see how the classifier performance is affected. Some of the reviewed models employed this metric too [71, 119, 135, 24, 127].

Others have gone for human quality assessment. In general this is done through workers of the Amazon Mechanical Turk[6] who are offered a choice between different model or baseline predictions to assess which one looks more realistic. In this sense this is a relative metric where the results are expressed by the percentage of times that generated videos by a specific proposal were chosen over the ones generated by other architecture. Some of the analysed models used this [142, 62, 132, 127] to compare their models against several baselines or other SOTA models.

Finally, many have proposed to use the model trained for video prediction for other tasks. In general the model is fine-tuned for it after it has learned representative features by generating future frames and measure how well does it perform compared to other traditional methods. This also serves as an assessment on how well the abstract representations that the model has learned are capable of generalizing to other tasks. For example, similar to the Inception Score, many [110, 63, 65, 132, 61] fine tune the model to perform a classification task, generally by replacing the generator layers by some simple classification layer and report classification accuracy obtained by their model. Oh et al. [79] trained a Deep Q Network (DQN) [73] to play Atari Games on a real game emulator and replaced it by their predictive model. Similar to the Inception Score for video games, they compare the game score of the DQN in both environments using the emulator results as an upper bound. Also, they use their predictive model as an informative tool to improve the DQN model through informed exploration of possible states, thus helping the bot better choose among possible actions depending on the predicted visual results. Shi et al. [103] and Klein et al. [53] used the predictions for the task of weather prediction. To establish the quality of the results they used different weather forecasting metrics. Finally, both Liang et al. [61] and Liu et al. [62] which implicitly model Optical Flow prediction compared the ones

[6] https://www.mturk.com/

produced by their models against traditional flow methods. Liu also tested the model for frame interpolation and generation of slow motion effect.

## 3.10 Single input frame models

This section details some models that have been proposed for future frame generation that base their prediction on single input frames. Even though they are generating the future, the framework for these models is very different, as they are estimating the future from a single static image instead of understanding a evolution or history of past events in order to extrapolate the forthcoming ones. There is a big literature on models that try to predict Optical Flow or, more generally, motion from single frames. However only a few of those have been considered and in particular, just proposals that actually generated the next frame and not only predicted a transformation in whichever form. One of the first such models was proposed by Pintea et al. [92] similar to optical flow, they estimated motion as flow derivatives. Similar to the models considered for this project, they use the next frame as a way of supervising the training procedure. However, the desired model should be able to infer motion given a single frame. They use a Structured Random Forest and use patch based HOG descriptors of frames and the associated ground truth motion (extracted using Optical Flow) to train the model. Each split in the tree is used to direct the choice towards the specific motion using euclidean distance to measure the difference. They test the model for different applications, one of which is next frame prediction. Even though the results exhibit several artifacts and that the model does not account for the whole structure of the image, the proposal seems very capable of properly detecting the moving regions and keep the static ones unchanged, which is a very desirable capability for frame prediction models.

Xue & Wu et al.[139] propose the Cross Convolutional Network: a model to sample and synthesize not one but many possible future frames from a single input image. As different images will probably have different motions, they suggest using different motion kernels for every input. In order to do so they develop the cross convolutional layer, which takes both feature maps and kernel weights as input. This layer applies the kernels to the feature maps and a motion decoder will produce the output difference image between the current input current frame and the predicted next frame. In this sense they are predicting the motion needed to transform the input frame into a possible future. Their architecture is then, composed of four parts: a variational motion autoencoder that produces the kernels, a pyramid convolutional encoder that produces feature representations of the input frame at multiple sizes, the cross convolutional layer and the motion decoder. During training, the image encoder takes a single frame as input, and the motion encoder takes that same frame and also the difference image between next and current frame. During testing however, instead of using the motion encoder they sample random motion vectors from a prior distribution which allows the system to produce different possible futures for the same input image. The model could easily be adapted to frame prediction by adding recurrence to the encoder, resulting in a model similar to [53] or [20] where filters are dynamically produced and then applied to the input in order to generate a prediction. They only performed experiments on toy datasets of moving shapes or limited human motion on blank backgrounds.

Vondrick et al.[130] propose a model for video generation that gathered much attention even out of the scientific community. It used Convolutional GANs to produce video without con-

ditioning on the past (or doing so in a single frame). They propose to generate background and foreground separately in videos where the camera is static. In order to do so they use an adversarial setting on the whole generated sequences. The generative model has two separate streams: one that models the static part of the video (background) and other that models the moving objects (foreground). To provide temporal invariance they use 3D spatio-temporal convolutions. Both streams are then combined in single frames to produce a video. They used camera-stabilized Flickr videos on separate classes (babies, beach, train station and golf) to test the system. Even though the videos exhibit several artifacts, it is clear that the model was able to produce videos with representative features of the generated class. Furthermore, the model was tried on an action classification task and obtained fairly good results given the amount of labeled examples used. However, these results were far from other SOTA for action recognition that used fully supervised videos.

Mirza et al. [71] proposed a simple model for tower fall prediction. Their objective is to enhance a clasificator for stability of a block tower with an architecture capable of predicting the future given a tower configuration. They used a future frame generator in order to improve classification accuracy. They chose a Convolutional AE to predict the last frame of the sequence given the first (5 time steps apart). Furthermore, they included an LSTM layer between encoder and decoder as an alternative model. However, this last model was trained using ground truth only but tested using its own predictions as feedback. In this sense the model didn't learn to predict more than one time-step further from the input. To test the model they evaluated classification accuracy using the first ground truth frame of the sequence and the last generated frame on the task of stability prediction. The classificator obtained better results when helped by the last frame in the sequence, as it generally exhibited the blocks fallen or still set up.

Finally, Vukotić et al.[134] proposed using a Conditional CNN AE to predict future frames given a single input image and a temporal distance in the future. The model is formed by a convolutional encoder, a conditional term that expresses how far in the future the prediction should be and a decoder that produces the prediction. The model was trained to predict up to 200 ms in the future. This allows the architecture to predict further in time without having to iterate the prediction process. Thus, this model does not have to deal with accumulating prediction errors and artifacts. This models offers some generalization capabilities that teach the model not only to predict the next immediate changes but also to account for longer term movement. Such feature may allow the model to better understand time lineality and extent. However, this model still have some drawbacks, as predicting from a single image increases the difficulty of predicting the correct future and the results usually represent that uncertainty with blurriness and other artifacts.

# Chapter 4
# Frame Prediction Models

This section contains a brief description of every model that has been considered for comparison in the future frame prediction framework. The contents that follow are thought to be an extensive and (almost) complete layout of the architectural designs that have been made to solve this problem. As mentioned before, any proposal that uses only one frame to produce the predictions has not been considered. One frame is not (except for very specific environments) informative enough to produce valid future predictions in videos containing natural fenomena and motion. Being said that, this does not take into account recurrent models where a prediction is made at every time step and the first generated frame is based only on one frame. Such models do have been considered as they actually used multiple frames as input in the long term.

This section is divided in two parts: first, a qualitative description of every proposal is prsented coupled with a figure showing the architecture. Second, a more quantitative comparison of the models may be seen in section 4.2 where specific details of every proposal have been structured and summarized. Some specific remarks of the models have also been included in the first part after the architecture display. These are not necessary for understanding the overall proposal but shed some light on details that are important to consider for the task at hand or clarifying specific data of the table in the second part. The models are roughly sorted by publishing time, however the actual publishing order may be slightly different to the one shown here due to date variability on the various consulted scientific sites.

## 4.1 Models

**Modeling deep temporal dependencies with recurrent "grammar cells"**

Michalski et al. [70] propose modeling a time series by dealing with transformations. For this purpose they use bilinear models, which are able to infer transformations from pairs of images. In particular, they use a pyramid of Gated Autoencoders for predicting the next frame in a sequence. The lower layer of the model extracts transformations among frames. A second layer would model changes among transformations, allowing the system to represent varying transformations in the sequence. The model can keep growing in depth to represent more abstract representations. Thus, the model is actually predicting transformations which are then applied to the last frame in order to generate a pixel level prediction. To predict longer sequences the model keeps generating based on the already predicted transformations applied to the last produced frame. They pretrained the model for reconstruction before training for prediction.

Also, layers were added during training so the model would first learn low level transformations not dependent on higher abstractions.

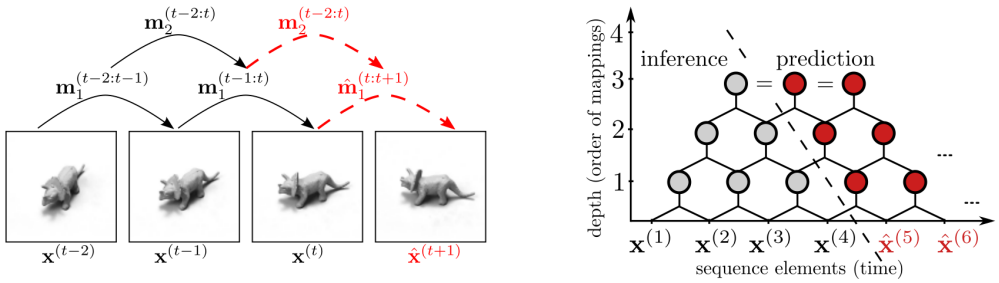


**Figure 4.1:** Michalski et al. proposed architecture for the grammar cells. Each layer builds a higher transformation representation. Source: [70]

Regarding the NORB videos dataset [69] which they used in some experiments, it is formed by sequences built using static images of the NORB dataset [59]. They use images of a single object from its different points of view in order to build videos of rotating objects. Thus, they do not actually contain natural motion. Also, the resolution, input and output frames included in Table 4.1 were inferred from information on the paper but were not explicitly mentioned.

### Video (language) modeling: a baseline for generative models of natural videos

In this paper by Ranzato et al. [93] a predictive model based in Natural Language Modeling is proposed. In their approach, instead of working with some loss function at pixel level, they decided to discretize the input and predictions using *k-means* over (10,000) patches of images. Thus, they reduced the prediction task to predicting an appropriate centroid in the generated clusters for every patch in the generated image. They used a Recurrent CNN to better handle spatial correlations, and a simple RNN as baseline for comparison. They tested the model for frame prediction as well as for filling missing frames.

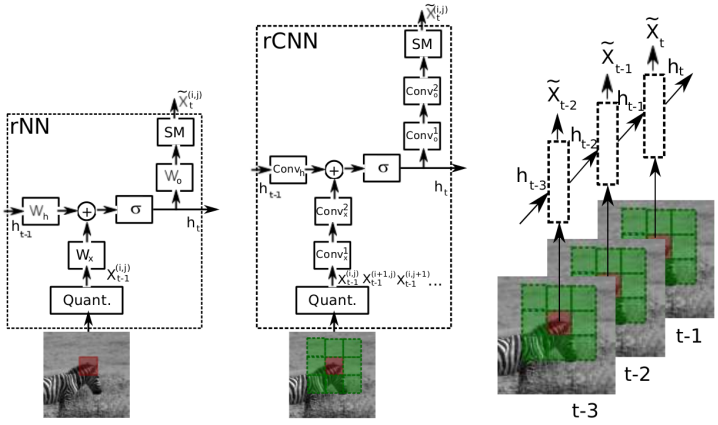


**Figure 4.2:** Ranzato et al. proposed architecture for the prediction model. *Left* shows the rNN model; *Center* shows the Convolutional rNN model; *Right* shows the predictive process based on each patch and its spacial neighbours through time. Source: [93]

Regarding the van Hateren dataset [42] used in some experiments, no explicit amount of input and output frames appeared on the paper so for the information in Table 4.1 the same as for the UCF-101 dataset was assumed.

### Predictive Encoding of Contextual Relationships for Perceptual Inference, Interpolation and Prediction

Zhao et al. [141] propose a model based on the neuroscience theory of predictive coding. Depending on training, this model is able to reconstruct missing frames as well as predicting future frames. In other words, they build a model that tries to maximize the inter predictability of the frames in a sequence. There are four different parts to this model: first, the input layer receives a video sequence; second, a hidden layer in which every unit takes a subset of (neighboring) input frames and produces a latent representation of part of the input sequence; next, a prediction layer that reconstructs every frame in the sequence. This is possible because the internal latent representations of the model encode context relationships among frames from the whole sequence. Finally, the reconstructed inputs are compared against ground truth in order to update the contextual representation and the different weights of the model. Missing frames in the input sequence will also be reconstructed based on context allowing the system to perform both frame interpolation and prediction. To produce training sequences for their experiments, they apply different affine transformations to static frames of natural and synthetic image datasets.

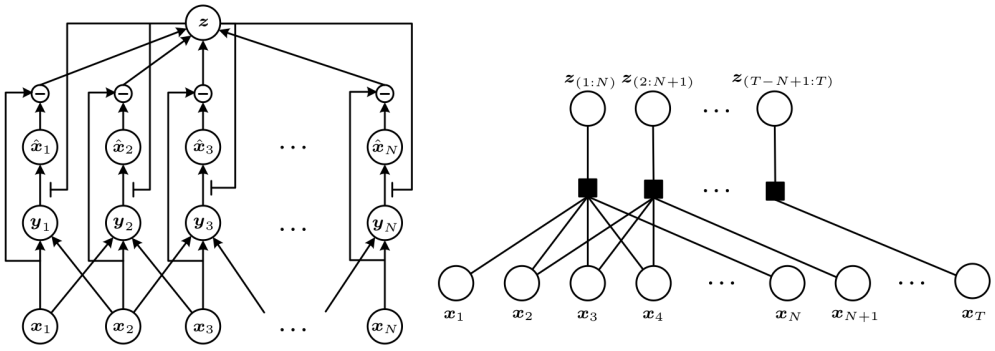


**Figure 4.3:** Zhao et al. proposed architecture. *Left* displays the computational circuit of the *Predictive Encoder* model; *Right* shows the model unrolled for a sequence of length $T$. Source: [141]

They used four different datasets, all of them formed by static images. They generated sequences by applying affine transforms (translation, rotation and scaling) to those images. The first experiment was carried out by randomly sampling from one out of three datasets: MNIST digits [58], Whitened Natural images [85] and Face images [40]. In the second experiment they used the NORB dataset [59], which already contains images of objects from different points of view, which were used to build sequences of rotating objects. Also, the amount of input and output frames, as well as the resolution shwon in Table 4.1 is general for all experiments and datasets. It is also important to note that they did not train nor predicted on full images but on cropped patches.

### Unsupervised Learning of Video Representations using LSTMs

This paper by Srivastava et al. [110] proposes using a LSTM AE to learn representation of video sequences. They use two kinds of input sequences: on one side, patches of images and, on the other, high-level representations of video frames extracted using a CNN. The AE architecture is formed by a LSTM encoder that learns useful features and a LSTM decoder that generates future frames from the latent video representation. They obtain the best results when using two separate decoders: one that attempts to reconstruct the input and, at the same time, another that tries to predict future frames. This composite model is further complemented with an additional input term that conditions the decoders on ground truth frames during training and previous predictions during testing. When compared with other action recognition techniques they obtain close to state-of-the-art results in the three cases: RGB data, explicitly computed flow features or both.

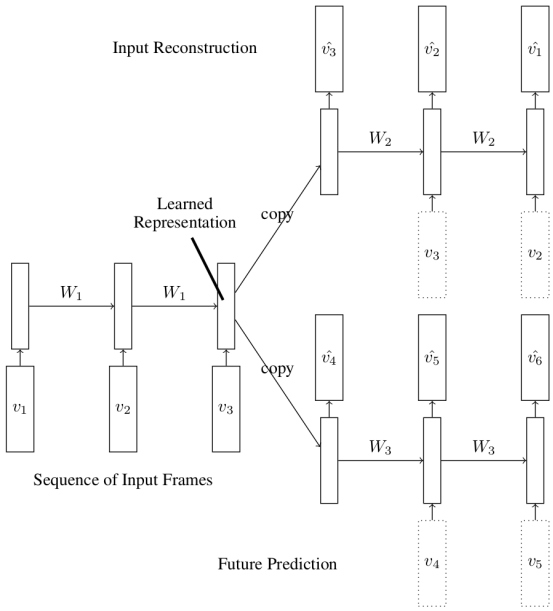


**Figure 4.4:** Srivastava et al. proposed composite LSTM architecture. The dashed line boxes are the conditional terms for the decoders. Source: [110]

In their experiments a pretrained model was used to extract RGB features from the Sports-1M dataset [50]. In particular, features from 16 frames were extracted using the CNN by [104] trained on ImageNet [22] and fed into their model instead of input directly at pixel level. Even though they do not report prediction results on Sports 1M, a model was trained to predict frames with this dataset for later fine-tunning with UCF101 [107] and HMDB-51 [56] for action classification. Sports-1M was not used raw in any case but with the extracted features commented above.

### Learning to Linearize Under Uncertainty

Goroshin & Mathieu et al. [39] propose a Convolutional AE architecture stressing the importance of the pooling for learning to linearize transformations and make better predictions. They

define soft versions of the *max* and *argmax* pooling and use them together to define what they call *phase pooling*, which outputs both values. Such setting allows them to implicitly defining priors on some features behavior, such as stability of intensity or linear movement. In order to properly handling uncertainty they use cosine distance between encoded input and target besides L2 loss at pixel level for the complete prediction.

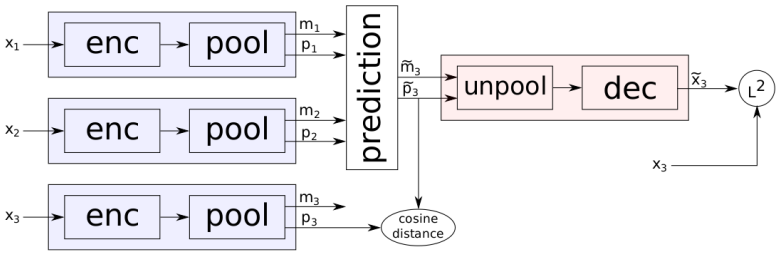


**Figure 4.5:** Goroshin et al. proposed AE for frame prediction. Source: [39]

They make a use of data similar to Zhao et al. [141]. Even though the NORB dataset [59] is formed by static images, they contain objects seen from different points of view that can be used to build video sequences of rotation. Regarding the YouTube Videos [38], they did both: used them as normal sequences and performed affine transformations on single frames to generate the sequences.

### Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting

Shi et al. [103] used Convolutional LSTM layers to build an autoencoder that predicted future frames in the context of precipitation prediction. They stress the importance of handling correctly the spatial correlations in the data, as the motion of clouds is highly consistent in a local region. For doing so, a certain cell in the grid will determine its future state by considering the inputs and past states of its local neighbors. They trained trying to minimize error loss for the whole sequence instead of individual frames. In this case, they were able to measure the accuracy of their predictions by using different weather prediction metrics, obtaining promising results.

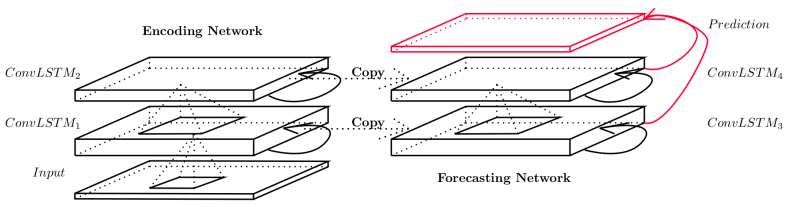


**Figure 4.6:** Shi et al. proposed AE model based on their Convolutional LSTM layer. Source: [103]

### Action-Conditional Video Prediction using Deep Networks in Atari Games

Oh et al. [79] proposed two autoencoder convolutional models: one feedforward and one recurrent. Both are trained to predict next frames of Atari Games oriented to Reinforcement Learning and use player-action conditioned predictions. In the recurrent case the encoder has first a set of simple convolutional layers followed by a LSTM layer that receives high level features as input. It receives and generates one frame at each time step. The feedforward uses two CNN that work with multiple frames at a time as input and produces a single frame. In both cases the action conditioning occurs on top of the features extracted by the encoder. They trained on different Atari games using curriculum learning. In other words, instead of using the error from every single prediction to learn, the model is trained to minimize the average squared error over multiple step predictions. The amount of steps used increments through training. Finally, another important feature was one of their testing proposals. They used a DQN [73] trained to play those games and replace the Atari Emulator by their models. The resulting game score allowed them to asses a comparison among the different tested models.

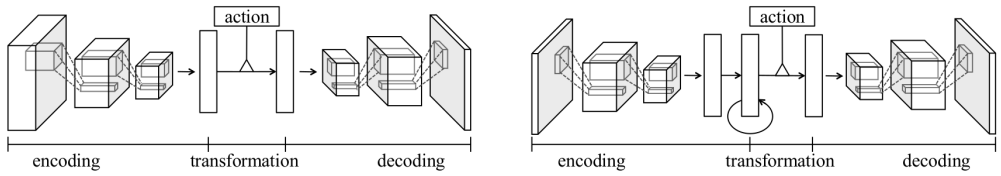


**Figure 4.7:** Oh et al. proposed methods. *Left* shows the feedforward AE; *Right* displays the recurrent proposal that incorporates one LSTM layer. Source: [79]

They used the Arcade Learning Environment (ALE) platform [6] and their Deep-Q-Network (DQN) replication of [73] to generate game-play video datasets of Atari games. In this case, the difference in the amount of input and output frames shown in Table 4.1 is not regarding various datasets but the different models tested: on one hand the feedforward CNN takes 4 frames at each time-step; on the other hand the recurrent LSTM takes one frame for each time-step, but it is unrolled through 11 frames to initialize the LSTM hidden units before making a prediction. Also, the generated frames are conditioned on the game action that leads to them.

### A Dynamic Convolutional Layer for Short Range Weather Prediction

Klein et al. [53] propose a new Dynamic Convolutional Layer that uses input-based filters instead of keeping constant ones learned only during training. What is trained instead is a convolutional sub-network to output the appropriate filters for a given input sequence. A second sub-network applies those filters to the last frame in the input sequence to produce a prediction. Using this architecture they experiment with two alternatives: on one hand they try using full resolution frames as input to predict the central 80% of the next frame; on the other hand they test a similar model but using patches over a sliding window and then combining the predictions to form the whole next frame. They tested on weather radar imaging for which the second approach produced better results.

They perform experiments using both, full resolution images and patches over a sliding window to generate a prediction for the next frame in the input sequence. Also, their predictions are only for the central region of the input. In other words, the prediction has smaller resolution

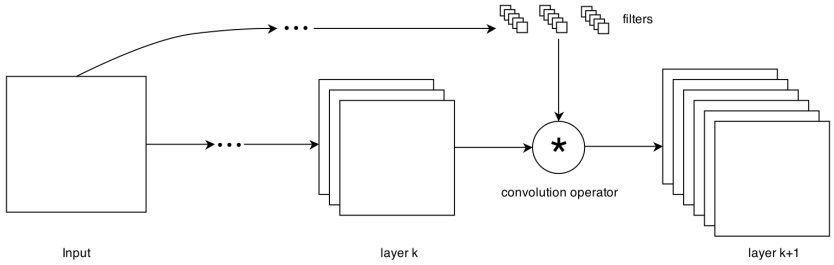


**Figure 4.8:** Klein et al. proposed architecture. The input is used to build the filters that will later transform the last frame in the future frame. Source: [53]

than the input: they predict $200 \times 200 \times 1$ and $10 \times 10 \times 1$ for the full resolution and patch sizes (shown in Table 4.1) respectively. This is done because margins are the zone wigh highest uncertainty for prediction as new elements (clouds in this case) are common to appear.

### Deep multi-scale video prediction beyond Mean Squared Error

In this paper by Mathieu et al. [67] they stress the need for other losses as some functions such as MSE produce blurry predictions. Their approach consists in using an adversarial loss in a GAN setting where both the generative and discriminative parts are multi-scale Convolutional Networks. Furthermore, they propose different loss functions composed by L2, a newly proposed Gradient Difference loss that penalizes the differences of image gradient predictions in the generative loss function and the aforementioned adversarial loss. They found that a GAN can play a very important role to secure sharp predictions that look qualitatively better than other generated frames using simple pixel-wise losses.

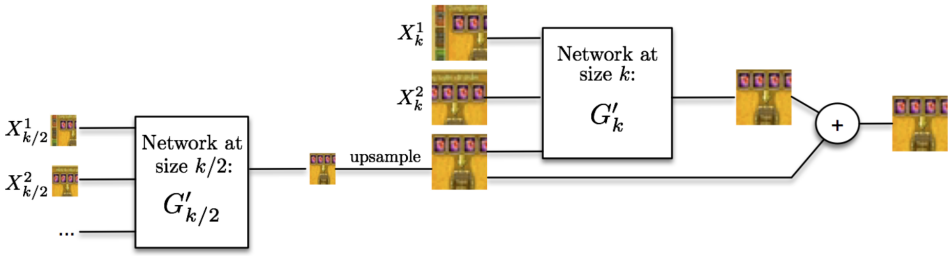


**Figure 4.9:** Mathieu et al. proposed multi-scale architecture. The predictions at different scales are used to produce a final prediction. Source: [67]

Apart from the data pointed out in Table 4.1, they also used the UCF-101 dataset [107] at $64 \times 64 \times 3$ to fine-tune a model trained with Sports-1M [50] at the resolution that appears below.

### Unsupervised Learning of Visual Structure Using Predictive Generative Networks

Lotter et al. (2015) [63] propose the Predictive Generative Networks. PGN consist in a Convolutional Autoencoder architecture with a recurrent (LSTM) layer between encoder and decoder. They used a weighted loss combining MSE and an adversarial setting. For the latter, a GAN structure was implemented where the discriminator tries to tell apart real sequences from

sequences which last frame is generated. In other words, the ground truth input sequence is encoded and either predicted or ground truth frame are appended to that latent representation. A final MLP tries to classify between real or generated. In order to avoid a cold start in the adversarial setting they pretrained generator and discriminator separately. The former was pretrained using MSE only and the latter using higher weight for the adversarial loss. They found that the adversarial loss encourages less blurry (more realistic) results than a pixel-wise reconstruction error.

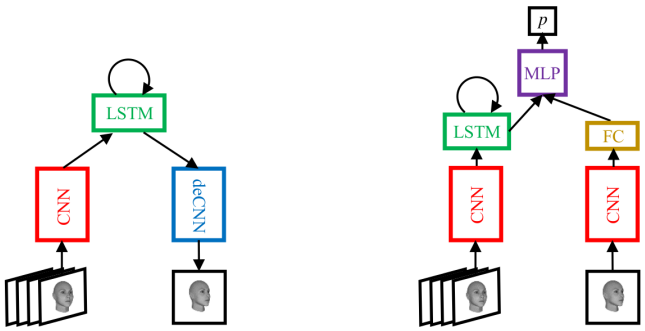


**Figure 4.10:** Lotter et al. architecture. *Left* shows the recurrent AE generator network that produces a single frame given the input. *Right* depicts the discriminator network that receives a ground truth sequence and the real/predicted next frame. Source: [63]

It is important to note that they trained the network using a variable number of input frames for the Bouncing Balls dataset [113], ranging from 5 to 15. Leaving that out, they always predicted the single next time step.

### Spatio-Temporal Video Autoencoder with Differentiable Memory

Pătrăucean et al. [91] propose using a Convolutional autoencoder to handle spatial correlations with a nested LSTM (with Conv. LSTM cells) and Optical Flow estimator for temporal dependencies among frames. They use a common reconstruction loss at pixel space between prediction and actual frame. Furthermore, la Huber loss as penalty to ensure local smoothness in the Optical Flow estimator is also used. They tested the model on both synthetic and natural sequences and further validate the model using it for a semantic segmentation task. Finally, they use an extension of their model to perform semantic segmentation on videos. Compared to other (baseline) models, the architecture trained jointly to perform frame prediction and semantic segmentation performed best.

They used a recurrent architecture that receives and outputs one frame at each time step, but it is unrolled to predict longer sequences (using previous generated frames as feedback). They perform experiments on synthetic and natural sequences. For the former they use the moving MNIST dataset [110]. For the latter they use HMDB-51 [56] for training and PROST [99] and ViSOR [124] for testing.

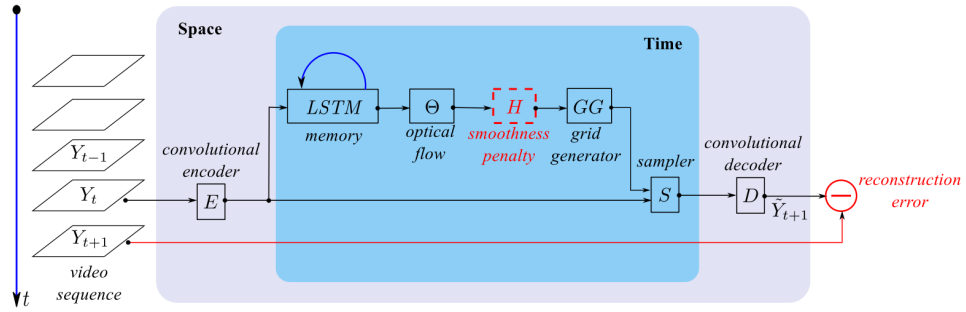


**Figure 4.11:** Pătrăucean et al. nested AE architecture. The inner LSTM network predicts flow to be applied to the input frame which has been encoded in the outter network. Source: [91]

### Unsupervised Learning for Physical Interaction through Video Prediction

Finn et al. [29] propose three different Convolutional LSTM models. Instead of predictions at pixel level, they calculate a set of motion predictions for individual pixels or pixel groups. A set of masks is used in order to allow the system to build new frames applying each motion to the appropriate pixels in the previous frame. These masks are also generated by the system depending on the input. In this sense they are generating multiple possible frames and then combining them to form a prediction. This setting allows them to disentangle background from moving objects and objects among themselves, producing more realistic results. They state that this model could be specially useful to guide autonomous agents motion through the physical world. The model would allow them to predict the future given visual input. For this reason they propose the Robotic Pushing dataset consisting in sequences of robotic arms pushing objects. The data contains not only video but also the state of the arms and current desired action for each frame. Conditioning on this extra information allows the system to better predict future frames.

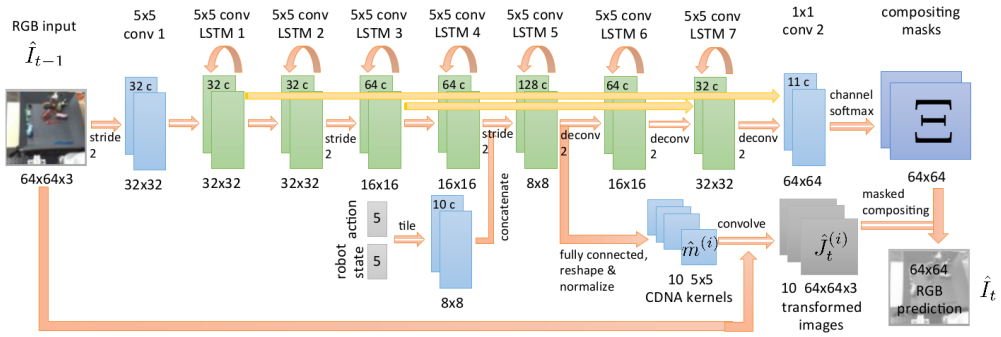


**Figure 4.12:** Finn et al. . Source: [29]

When performing experiments on the Robotic Pushing dataset they used 2 frames coupled with the state and actions as ground truth. Then, the system produced 1 frame at every time step (conditioned on the desired ground truth action) using its own predictions to keep generating up to 8 frames during training and up to 18 during testing. For Human3.6M they used the same approach but without conditioning on actions and predicted up to 10 and 20 frames during

training and testing respectively.

### Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning

This paper by Lotter et al. (2016)[65] poses a model inspired in neuroscience literature. In particular they base their architecture in the concept of Predictive Coding, where the system is continuously making predictions and comparing them to the actual input. The error is then propagated so the predictive model can be updated. This system is formed by several layers, each composed by four components: a convolutional input, the representation module that produces the predictions (Convolutional LSTM), a prediction (convolution of the representation module output) and an error representation module. Each layer of the system makes a prediction at its specific level of abstraction which is compared to the actual input. The error is then fed as input to the next layer so more abstract representations may be built. The representation module is conditioned on the previous error and the prediction from higher layers. Even though the loss function used at the lower layer is equivalent to an L1, the loss at higher layers is "implicitly embedded in the network as the firing rates of the error neurons". They tested the system on synthetic and natural video sequences for prediction. Also, the system was used for classification tasks (using linear SVM for decoding the class) with promising results.

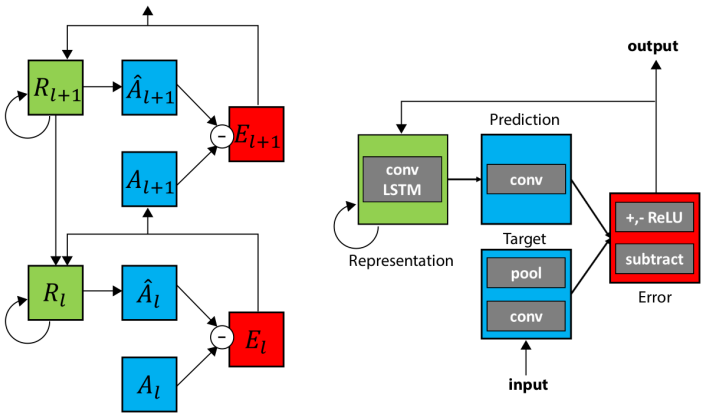


**Figure 4.13:** Lotter et al. (2016) proposed PredNet architecture. *Left* shows two stacked layers of the model. See how the error propagation flows bottom-up while the predictions flow top-down. *Right* depicts a single layer into detail. Source: [65]

Here, the FaceGen software was used to generate sequences of rotating 3D faces as an initial model testing. Regarding the real motion datasets, they used KITTI [34] for training and Caltech Pedestrian [25] for testing. The model was trained to predict one step ahead of time. The input sequences where 10 frames long so predictions where made for 1, 2 ... 10 input frames, always based on ground truth. In later experiments, they fine-tuned the model to predict further based on its own predictions up to 15 frames. Thus, they predicted up to 5 frames more without ground truth support, but always 1 frame at every time step.

### Dynamic Filter Networks

Brabandere & Jia et al. [20] propose Dynamic Filter Networks for video and stereo prediction. This architecture is composed of two different parts: first, the filter generating network that produces filters conditioned on input; second, the dynamic filtering layer that applies the produced filters to another input. In particular, they use a Conv. AE (with a recurrent connection) for the filter generating network. They propose two alternatives: on one hand a dynamic convolutional layer, which generates a single filter to be convolved through the whole image; on the other hand, a dynamic filtering layer, that generates different filters to be applied at different locations. The generated filters are then applied to the last input frame by means of a simple convolutional layer that applies the generated filters to the input in order to produce future frames. They showed better results when using the dynamic filtering layer that allowed the model for greater variability in its predictions. Also, they used an smoothing penalty to encourage neighboring filters to produce similar transformations.

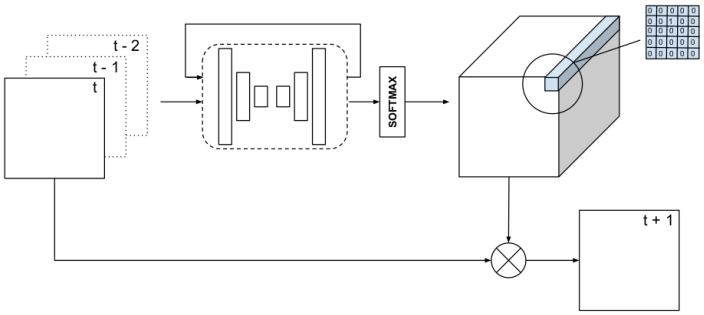


**Figure 4.14:** Brabandere et al. Dynamic Filter Network. The upper recurrent CNN AE produces a set of filters for different patches of the image. These are then applied to the last frame of the input sequence to form a prediction. Source: [20]

They used a recurrent approach that predicted one frame at every time-step. Even though it was only fed ground truth during training, the model was shown its own predictions as input at test time after some initial ground truth (see Table 4.1) was fed to the system.

### Learning Temporal Transformations From Time-Lapse Videos

Zhou et al. [142] propose using time-lapse videos to learn long term dependencies in natural sequences enabling the system to predict longer time scales. They build their own dataset with YouTube time-lapse videos of Blooming, Melting, Baking and Rotting. They propose three different models for three different frame prediction tasks. First, a pairwise generator where an input image and a time difference (similar to the single output frame model by Vukotić et al. [134]) are used to predict one frame in the future at a specific temporal distance. For this they use a simple Convolutional Autoencoder and condition on the number of steps in the future. Second, given two frames, predict the next one in the series extrapolating the time elapsed between both input frames. Based on the first architecture, they use two stacks in the encoding part and remove the conditional input. Finally, given a single frame, recursively generate future states. In this case they first use a Convolutional Encoder followed by a LSTM layer and finally

a Convolutional Decoder. It is also important to mention the use of an Adversarial loss in most of their experiments, which involves using a GAN setting for training their models. Also, they found much improvement when first pretraining their models for the task of reconstruction on ImageNet [22] and then fine-tune for the specific prediction task.

Even though two of the models use only a single frame to predict, both architectures have some interesting capabilities. Using a time differential to condition the distance of the prediction may help a model to model the time dimension more clearly. Also, unrolling the decoder once the input is encoded is a predictive approach that avoids using feedback and generally helps alleviate the long term blurriness.

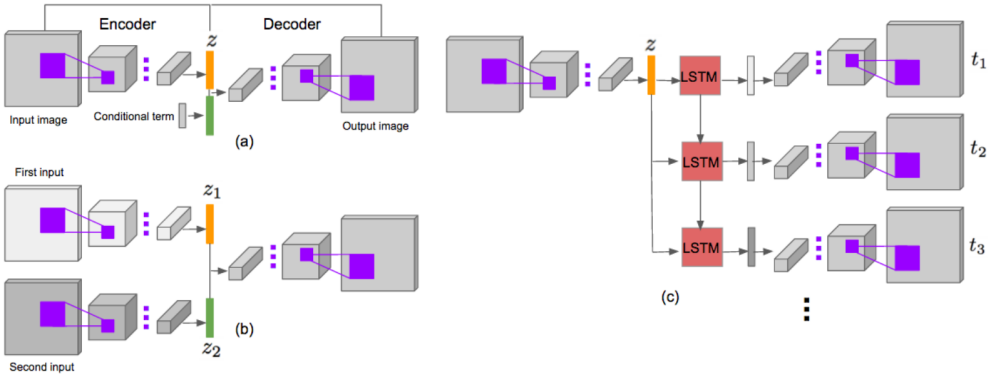


**Figure 4.15:** Zhou et al. three proposed models. *Left up*: Model that given a single frame and time differential predicts the next frame. *Left down*: Two separate endocers for two consecutive frames. *Right*: Recurrent model. Source: [142]

In this case, the difference in amount of input and predicted frames shown in Table 4.1 is for the three different proposed models. Also, the *Extra features* column regards there to the conditional time diferential of the first model.

### Video Pixel Networks

This architecture for video prediction proposed by Kalchbrenner et al. [49] tries to estimate the joint distribution of the pixel values. In order to do so they condition not only in previous frames but also in the already predicted pixels of the frame being generated. They use an autoencoder architecture. On one hand, the encoder is a Resolution Preserving CNN which outputs are given to a Convolutional LSTM. The decoder on the other hand is a PixelCNN (van den Oord et al. [120]). The encoder models time while the decoder models spatial and color dimensions. With this architecture they achieved to avoid blurriness in the results which highlights the importance of directly handling multimodality on future frame prediction. By generating every pixel at a time and conditioning on the previously predicted ones for the current frame the model ensures coherence when multiple futures are possible.

They follow a very similar approach to train on the Robotic Pushing dataset as the one used by Finn et al. [29]. They first condition on some initial frames and the rest are feedback predictions with ground truth actions. See Table 4.1 for more details.

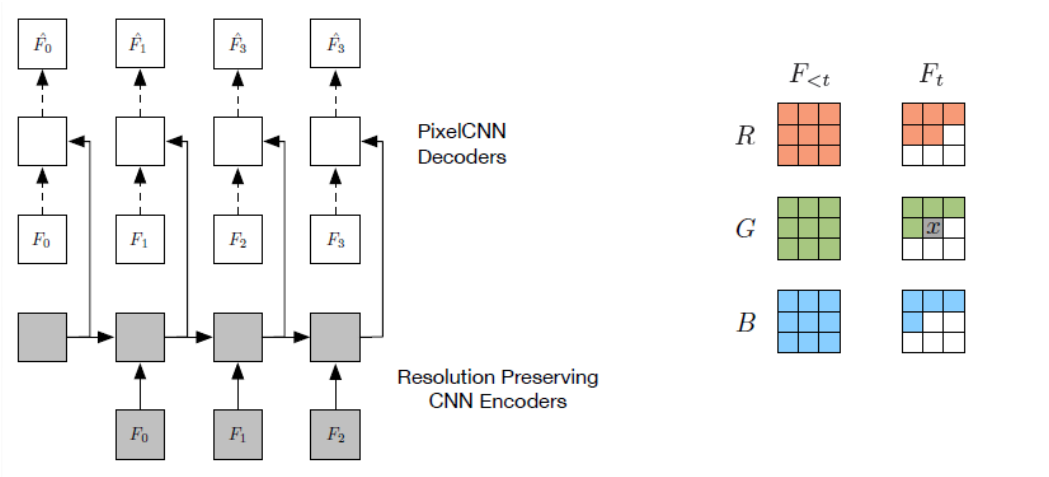


**Figure 4.16:** Kalchbrenner et al. Video Pixel Network. *Left*: The recurrent architecture of the model. *Right*: Dependency map of the pixel $x$ (coloured pixels). While a normal model only conditions on previous frames, the generation of a particular pixel in the VPN is also dependent on already produced pixels of the current frame. Source: [49]

### Video Ladder Networks

In this paper by Cricri & Ni et al. [18] they propose a simple autoencoder architecture for frame prediction. The encoder-decoder model is enhanced with lateral recurrent (Convolutional LSTM) and feedforward lateral connections, which form a recurrent residual block. Passing layer-wise information to the corresponding step in the decoder allows for a stratified representation of the data. Independent representations emerge as higher layers only need to further abstract without handling lower level details. They show very good qualitative results on the moving MNIST dataset and close to state-of-the-art quantitative results.

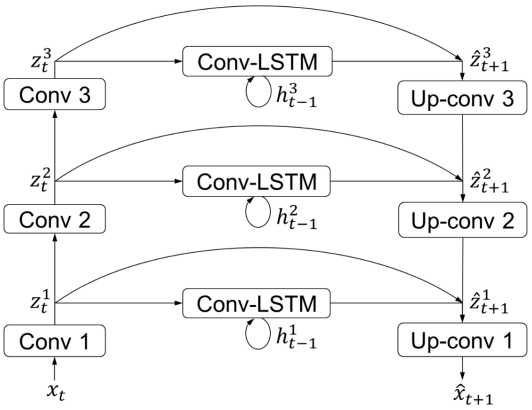


**Figure 4.17:** Cricri & Ni et al. proposed AE architecture. The residual recurrent connections allow for better layered features. Source: [18]

They predict one frame at a time using 10 frames as input (unrolling the encoder before predicting). During both, training and testing, the model uses its own predictions to shift the

input window and generate further future frames.

### Transformation-based models of Video Sequences

Amersfoort et al. [119] proposed a model which predicted transformations of patches instead of working at pixel level. They extracted affine transformations between overlapping patches in consecutive frames and fed that to a CNN that predicted the next transformations. The loss was applied directly at this level before using the the prediction to warp the last frame in order to produce the pixel wise representation. Every pixel in the resulting prediction is an average of the overlapping predicted transforms. They also stressed that using the model predictions to reach further in the future during training yields better results. Many models only do so at test time causing errors to accumulate, as the model hasn't been made robust enough to its own errors. Finally, they proposed to use the Inception Score for measuring resutls. They trained a model with a labeled dataset of ground truth sequences. Then compared classification accuracy obtained with ground truth test set (as upper bound) versus generated sequences.

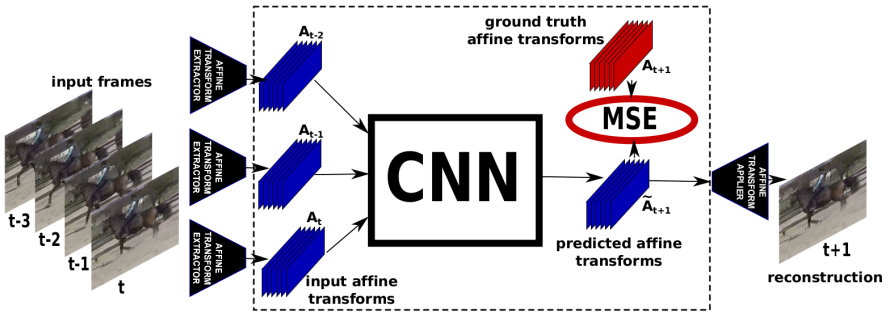


**Figure 4.18:** Amersfoort et al. proposed model for predicting frames at an Affine Transformation level. A sequence of motions is fed into the model in order to extrapolate the next one and produce the future frame. Source: [119]

As it has been mentioned, instead of working at pixel level or whole images, they extracted an affine transforms for different overlapping patches on the image and then averaged the predicted pixel values. For both datasets they used $16 \times 16$ patches to predict $8 \times 8$ patches. Furthermore, in the case of UCF101 [107], they used a network that worked at the resolution pointed in Table 4.1 as "patches" in order to produce full resolution predictions at $240 \times 320$.

### Video Frame Synthesis using Deep Voxel Flow

Liu et al. [62] propose Deep Voxel Flow for interpolation and prediction. They built a model that tries to overcome the blurriness of hallucinating whole frames from scratch using Optical Flow without supervision. In order to do so, their architecture is formed by a Convolutional AE with skip connections between encoder and decoder that predicts 3D voxel flow, and then a volume sampling layer (CNN) that synthesizes the desired frame by applying the predicted transformations. Then, as this architecture seemed troubled by larger motions, they made the model multi-scale so it predicted at different scales which where combined to form the final flow. This allowed to predict the correct transformations even for larger motions. They used the

model for both interpolation and extrapolation (prediction). The model, once trained for these tasks, was fine-tuned and tested in other tasks, such as Optical Flow prediction, view synthesis, action classification and slow motion effect on HD videos.

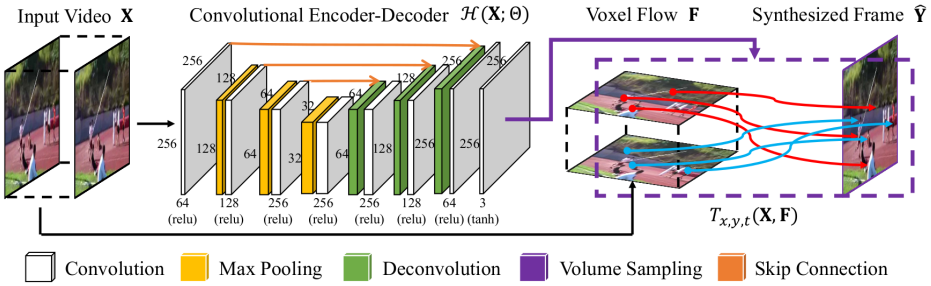


**Figure 4.19:** Liu et al. proposed architecture to predict pixel flows in order to generate future frames. Source: [62]

Despite the resolution included in Table 4.1, they used a multi-scale architecture. In other words, the network receives as input the frame at three different sizes, being the bigger one the explicited there.

### Generating the Future with Adversarial Transformers

Vondrick et al. [132] propose predicting transformations instead of directly pixels as a way of disentangling appearance from movement. In order to do so, they use a Convolutional Network to predict the transformations needed to convert the input sequence into the next frames, thus generating a prediction. To ensure realistic predictions they use an adversarial loss applied to the whole generated sequence. The discriminator consists on a spatio-temporal Convolutional Network that sees the whole sequence: input ground truth and prediction. This setting tries to guarantee that the predicted future is coherent with the input sequence. They test against several baselines and perform a system analysis to untangle what the model has learned. They also stress the idea that the performance in this task should be measured by the quality of the predictions and not by the closeness of those to the ground truth. Finally, they use their model as a pretraining for an object classification task.

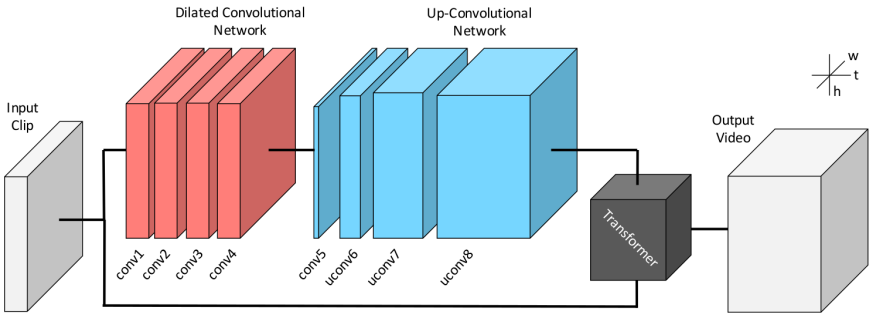


**Figure 4.20:** Vondrick et al. proposed architecture to predict the needed transformation to warp the last frame in the input sequence into a future sequence. Source: [132]

### The Pose Knows: Video Forecasting by Generating Pose Futures

In this paper by Walker et al. [135] they propose a new approach to video prediction. Instead of predicting directly pixel intensities or transformations to apply to the last frames they propose predicting at feature space using LSTM VAEs and Convolutional GANs. In particular, they propose a model for human motion prediction from videos which works at pose space. Their architecture is built in three separate parts: first, a Convolutional Network extracts high level features from the last frame and a pose detector is used to extract humans from the sequence. Both, the pose sequence and the last frame are fed to an LSTM AE where the decoder is trained to reconstruct the input. Then, a second LSTM AE is used to decode abstract pose representations into a future pose sequence conditioned on a random latent vector. Finally, the predicted pose velocities and the last frame are fed into a CNN GAN that generates the future image sequence. They use skip connections to help preserve low level details through the model. Another important aspect is they use the loss directly at pose level when training, not at pixel level. Finally, they evaluate the model at both pose and video level and compared against several baselines.

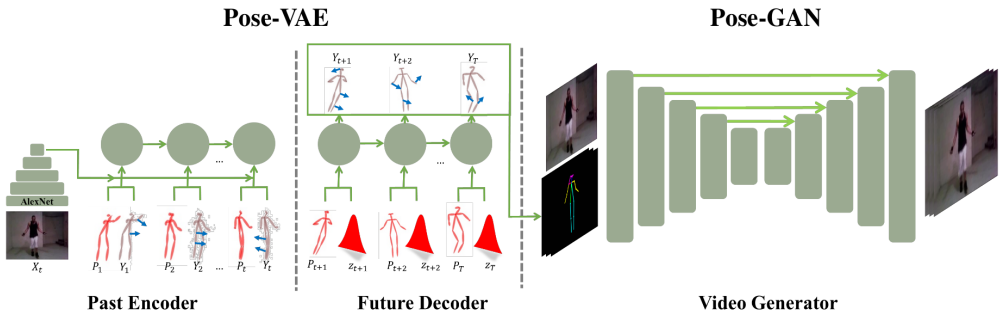


**Figure 4.21:** Walker et al. proposed architecture. First the past encoder takes a sequence of poses and the last frame. Then the state of the encoder is passed to the future decoder to produce a future pose sequence. Finally, the video generator takes the last frame of the input sequence and the predicted poses to generate the next frames. Source: [135]

They used a pretrained model by [12] to extract pose features from the frames and use them as the data to predict from instead of working directly at pixel level. The amount of frames used as input and output (see Table 4.1) were not stated explicitly in the paper but deduced from the supplementary material. Also, it is not clear the way in which they exploit the recurrence capabilities of the model for pose prediction, weather it is one at a time or encoder and decoder are alternatively unrolled.

### Unsupervised Learning of Disentangled Representations from Video

Denton et al. [24] propose a model for video prediction that separately models content (almost static visual information of the video) from pose (dynamically changing content of the video): the Disentangled Representation Network (DRNET). For this purpose, they use two networks that are trained separately: a Convolutional AE with two encoders and an LSTM to predict future representations. The two encoders produce distinct feature representations of content and pose for each frame. In order to ensure this separate representation they impose an Adversarial constraint on the pose encoder and a similarity loss on the content encoder. For the former,

the discriminator is a classifier that has to establish if two pose representations come from the same video or not. Thus the pose encoder has to generate representations that the discriminator is not able to tell apart ensuring that the content is not contained in it. For the latter the use a similarity loss to force content representations from the same sequence to be close. Finally, the pose representations are fed into the LSTM that makes predictions in latent pose space and the decoder generates future image frames based on the predictions and the encoded content. The training procedure has two steps: first, the CNN AE is trained to encode and reconstruct frames from the data. Then, the LSTM is trained to predict pose vectors using "ground truth" latent pose vectors extracted with the trained encoder. They perform different experiments on real and synthetic datasets and the model is able to successfully predict long distances into the future.

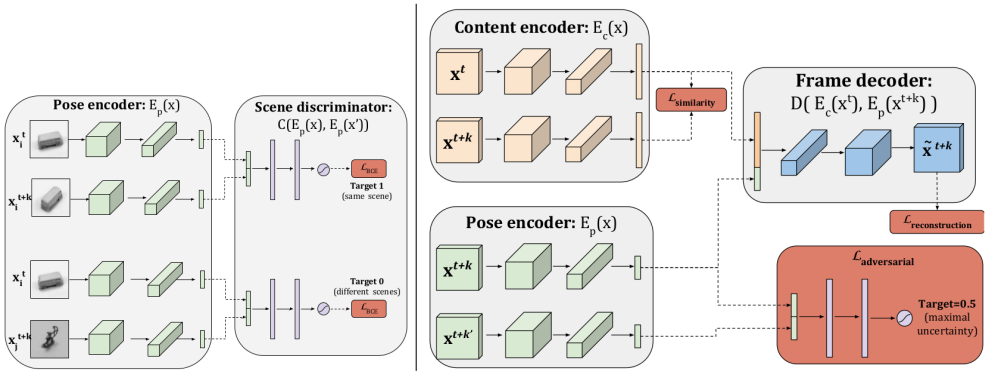


**Figure 4.22:** Denton et al. proposed model parts. *Left* displays the scene discriminator that tells apart if two pose vectors come from the same scene or not. *Right* shows the overall model architecture. Source: [24]

They use four different datasets for their experiments. However, they only report prediction results on moving MNIST [110] and KTH [101], using NORB [59] and SUNCG [106] for assessing their model capabilities on extrapolating pose and content latent vectors in order to generate completely new sequences and frame views.

### Decomposing Motion and Content for Natural Video Sequence Prediction

In this video prediction scheme by Villegas et al. (a) [126] they propose the Motion-Content Network (MCNet) for separating content and motion using difference images. In order to do so, they employ an Autoencoder architecture with two steam encoder: first a CNN followed by a Convolutional LSTM that uses motion images to handle movement; second, for encoding visual information, a CNN that uses the last frame of the sequence. Then a Convolutional decoder takes a combination of both encoded representations to output the next predicted frame. In some experiments they also try using residual connections from each encoder directly to the decoder, thus reducing the information loss produced by the pooling layers and yielding better results. Finally, they fit their predictor into a GAN environment to sharpen the generated frames. They perform various experiments using different datasets for training and testing in order to assess how well does the network generalize.

They use KTH [101] for train and test but also use sequences from the Weizmann Action dataset [11] for testing in order to assess how well the model generalizes. For color experiments,

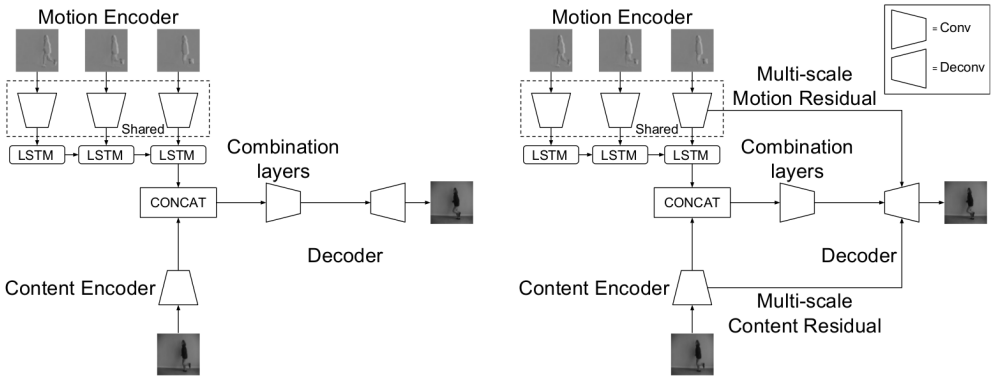


**Figure 4.23:** Villegas et al. (a) proposed architectures with and without residual connections. Source: [126]

they used the UCF101 [107] dataset for training and testing. Also, and again in order to test the model generalizability, they trained the model on Sports-1M [50] and tested on UCF101.

### Dual Motion GAN for Future-Flow Embedded Video Prediction

Liang et al. [61] propose using two Wasserstein GANs to encourage good content and motion prediction separately. For this purpose they use an AE structure with a single encoder (CNN and ConvLSTM layers) and two decoders (CNN): the first predicts frames and the second predicts pixel flows. Each of these decoders represent a generator in the GAN architecture. The two discriminators are trained to separate real from generated images and flows, respectively. The final generated frame is a combination of the flow and image prediction. Another important aspect of the model is that both decoders profit from the learning of the other: a flow warping and a flow estimator (CNN AE) are used respectively to obtain the image from the predicted flow and vice versa. These are also fed to the discriminators during learning, ensuring that the pixel level predictor produces sequences with proper motion and that the flow predictor is coherent at pixel level. As ground truth for the flow predictor they use Optical Flows extracted from the real frames using Epic Flow [95] They perform multiple experiments on a variety of different complexity datasets and compare against other state-of-the-art models. Also, they test the model (after some fine-tuning) for the task of flow prediction and action classification.

### Learning to Generate Long-term Future via Hierarchical Prediction

Villegas et al. (b) [127] propose a model for video prediction oriented to sequences containing humans. Their main contribution is making predictions at latent representation level instead of predicting at pixel level. Also, instead of a recurrent approach that accumulates errors by using feedback predictions, their proposal takes advantage of working at latent space to predict multiple time-steps before generating the image sequence. First, poses are extracted using a pretrained model by Newell et al. [76]. Then an LSTM AE observes a sequence of poses and predicts the next ones. Finally, an image generator takes the last frame, its corresponding pose and a predicted pose in order to generate a future frame by analogy making: the pose transfor-

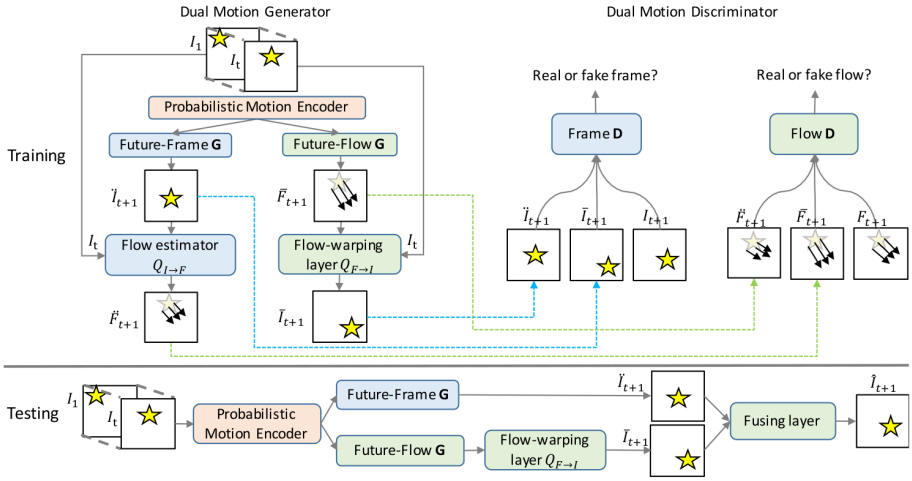


**Figure 4.24:** Liang et al. proposed Dual GAN architecture. They use the adversarial setting for training (*above*) and a fusing layer during prediction time in order to combine the strenghts of both decoders (*below*). Source: [61]

mation is transferred to the last frame via a CNN AE. They perform several experiments on real data and show promising results on transferring the predicted 2D poses to the original videos. Another important feature is they train not only using traditional losses (euclidean distance) at pixel level and an adversarial setting, but also a loss directly on the feature space (using AlexNet [55] as ground truth feature space). Furthermore, they use their predicted sequences for an action classification task to asses how representative were the generated videos of the specific actions.

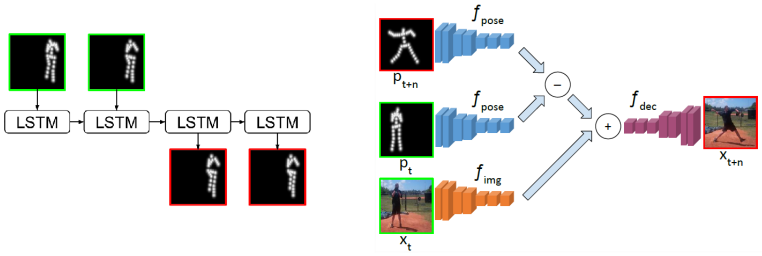


**Figure 4.25:** Villegas et al. (b) proposed architecture. A recurrent autoencoder is used to perform predictions at pose level (*left*) while the CNN AE is used to apply the predicted poses to the last frame in order to generate the future (*right*). Source: [127]

### Folded Recurrent Neural Networks for Future Video Prediction

In the proposal by Oliu et al. [84], they work with a new form of AE with shared states between encoder and decoder: the Folded RNN. This architecture uses GRUs that implement a double mapping between inputs and outputs, allowing for bidirectional flow of the information. A series of these bijective GRUs builds up as an AE architecture in which encoder and decoder states are shared, thus enabling the network for a layered abstract representation of the content. Their model uses the encoder for all frames and then every prediction is generated: first a CNN is

used to project the input frames to a latent space and fed into the recurrent encoder; when all the input sequence has been encoded the decoder is executed as many times as predicted frames are wanted. A final CNN performs deconvolution on the output to form the new frames. They stress the strength of their model for not accumulating errors during predictions as them are not fed back into the network. They compare against several baselines and other state-of-the-art models obtaining promising results.

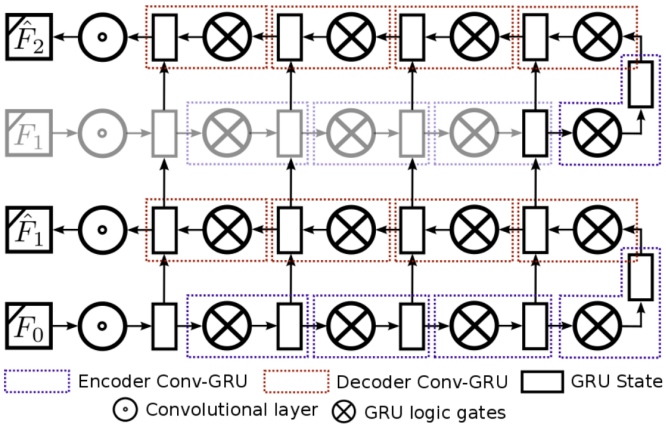


**Figure 4.26:** Oliu et al. folded RNN. Once the sequence has been encoded the decoder may be unrolled to produce multiple frames without the need of using predictions as feedback. Source: [84]

## 4.2 Comparative

Table Table 4.1 shows a comparative among different model features. In general the frame rate, the amount of input and output frames and the resolution are tied to the specific dataset (unless otherwise stated) in the same row. In particular, the amount of input and output frames has been difficult to asses specially in the case where recurrent models are used. In general, the table shows the amount of frames that the model predicted during train time without seeing new ground truth. Many times, however, recurrent models predict only one step ahead. In some cases the models were trained on its own predictions or predicted further than trained for. Specifics of each model and the coding used for the information display in the table can be read in the notes made on every model above and the table description. Regarding the use of GANs, it has been addressed in two different ways: as model and as loss. On one hand, a GAN is considered to be part of the model if the proposal uses it integrated with the rest of the architecture. On the other hand, if an adversarial setting was used by just using a discriminator to guide the generator gradients during training, the GAN has been considered a loss. Thus, *Adv.* is used in the loss column to indicate the use of an adversarial setting for sharpening the predictions and *GAN* is used in the model columnt to point out the use of such networks for the model basic architecture.

| | Model | Dataset | Data type | FPS | # Input frames | # Pred. frames | Extra features | Resolution | Pretrained model | Loss Function | Code |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Michalski et al. [70] (2014) | Pyramid of Gated AE | NORB videos [69], Bouncing Balls [114] | N* [**pd**]<br>A [**pd**] | - | 2, 3<br>5 | 1, 2<br>20 (56) | No | $13 \times 13 \times 1$ | Yes | L2 | No |
| Ranzato et al. [93] (2014) | rNN+kNN, rCNN+kNN | UCF-101 [107], van Hateren [42] | N [**pd**]<br>N [**pd**] | - | 12 | 1 | No | $8 \times 8 \times 1$ **[p]** | No | CE | No |
| Zhao et al. [141] (2014) | Inspired by: GBM & GAE | MNIST digits [58], Face Images [40], Whitened Images [85], NORB [59] | A* [**pd**]<br>N* [**pd**]<br>N* [**pd**]<br>N* [**pd**] | - | 2<br>5 | 1<br>1 | No | $13 \times 13 \times 1$<br>$21 \times 21 \times 1$ | No | L2 | No |
| Srivastava et al. [110] (2015) | LSTM AE | moving MNIST [110], Sports-1M [50], UCF-101 [107], HMDB-51 [56] | A [**pd**]<br>N [**pr**]<br>N [**pd**]<br>N [**cl**] | -<br>30<br>-<br>- | 10<br>16<br>16<br>- | 10/100<br>13<br>13<br>- | No<br>Yes<br>No<br>No | $64 \times 64 \times 1$<br>$224 \times 224 \times 3$<br>$32 \times 32 \times 3$<br>- | Convnet by [104] on ImageNet [22] | CE, L2 | [111] |
| Goroshin & Mathieu et al. [39] (2015) | CNN AE | YouTube Videos [38], NORB [59] | N* [**pd**]<br>N* [**pd**] | - | 2 | 1 | No | $32 \times 32 \times 3$<br>$32 \times 32 \times 1$ | No | Gaussian dist., L2 **[f]** | No |
| Shi et al. [103] (2015) | Conv. LSTM AE | moving MNIST [110], Radar Echo **[o]** | A [**pd**]<br>A [**pd**] | - | 10<br>5 | 10<br>15 | No | $64 \times 64 \times 1$<br>$100 \times 100 \times 1$ | No | CE | [102] |
| Oh et al. [79] (2015) | CNN AE, CNN AE + LSTM | ALE [6] **[o]** | A [**pd**] | 15 | 4<br>11 | 1<br>1 | Yes | $210 \times 160 \times 3$ | No | L2 | [80] |
| Klein et al. [53] (2015) | CNN + Dynamic Conv. Layer | Radar images from: Tel Aviv **[o]** , Davenport **[o]** , Kansas City **[o]** | A [**pd**]<br>A [**pd**]<br>A [**pd**] | - | 4 | 1 | No | $250 \times 250 \times 1$<br>$70 \times 70 \times 1$ [**p**] | No | Euclidean | No |
| Mathieu et al. [67] (2015) | MS CNN | Sports-1M [50], UCF-101 [107] | N [**pr**]<br>N [**cr**] | - | 4 , 8<br>4 | 1 , 8<br>1 | No | $32 \times 32 \times 3$ | No | L2, L1, Adv., Binary CE, Gradient Difference **[f]** | [68] |

| | Model | Dataset | Data type | FPS | # Input frames | # Pred. frames | Extra features | Resolution | Pretrained model | Loss Function | Code |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Lotter et al. [63] (2015) | Conv. AE+ LSTM | Rotating Faces **[o]**, Bouncing Balls [113] | A **[pd]**<br>A **[pd]** | - | 5<br>5-15 | 1<br>1 | No | $150 \times 150 \times 1$<br>$20 \times 20 \times 1$ | No | MSE, Adv. **[f]** | No |
| Pătrăucean et al. [91] (2016) | CNN AE + Conv. LSTM + Optical Flow | moving MNIST [110], HMDB-51 [56], PROST [99], ViSOR [124] | A **[pd]**<br>N **[pr]**<br>N **[cr]**<br>N **[cr]** | - | 1 | 1 | No | $64 \times 64 \times 1$<br>-<br>-<br>- | No | Binary CE | [90] |
| Finn et al. [29] (2016) | Conv. LSTM AE | Robotic Pushing [29], Human3.6M [46] | N **[pd]**<br>N **[pd]** | -<br>10 | 2<br>10 | 8/18<br>10/20 | Yes<br>No | $64 \times 64 \times 3$<br>$64 \times 64 \times 3$ | No | MSE | [30] |
| Lotter et al. [65] (2016) | Conv. LSTM + CNN | Rotating Faces **[o]**, KITTI [34], Caltech Pedestrian [25] | A **[pd]**<br>N **[pr]**<br>N **[cr]** | - | 10 | 1,5 | No | $64 \times 64 \times 1$<br>$128 \times 160 \times 3$<br>$128 \times 160 \times 3$ | No | L1 | [64] |
| Brabandere & Jia et al. [20] (2016) | Recurrent Conv. AE + CNN | moving MNIST [110], Highway Driving **[o]** | A **[pd]**<br>N **[pd]** | - | 10<br>3 | 10<br>3 | No | $64 \times 64 \times 1$<br>- | No | Binary CE, Euclidean | [19] |
| Zhou et al. [142] (2016) | Conv. AE, Conv. AE + LSTM | YouTube Time Lapse Videos [142], ImageNet [22] | N **[pd]**<br>N **[pr]** | - | 1<br>2<br>1 | 1<br>1<br>4 | Yes<br>No<br>No | $64 \times 64 \times 3$<br>- | No | Binary CE, pixel MSE, gradient MSE, Adv. | No |
| Kalchbrenner et al. [49] (2016) | CNN+ Conv. LSTM+ PixelCNN [120] | moving MNIST [110], Robotic Pushing [29] | A **[pd]**<br>N **[pd]** | - | 10<br>2 | 10<br>10/18 | No<br>Yes | $64 \times 64 \times 1$<br>$64 \times 64 \times 3$ | No | Sigmoid CE, Softmax CE | No |
| Criri & Ni et al. [18] (2016) | Conv. LSTM AE | moving MNIST [110] | A **[pd]** | - | 10 | 10 | No | $64 \times 64 \times 1$ | No | Binary CE | No |
| Amersfoort et al. [119] (2017) | CNN + Affine Transform Extractor | moving MNIST [110], UCF-101 [107] | A **[pd]**<br>N **[pd]** | - | 4<br>4 | 6<br>8 | No | $64 \times 64 \times 1$<br>$64 \times 64 \times 3$ **[p]** | No | MSE | No |

| | Model | Dataset | Data type | FPS | # Input frames | # Pred. frames | Extra features | Resolution | Pretrained model | Loss Function | Code |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Liu et al. [62] (2017) | MS CNN + MS Conv. AE | UCF-101 [107] THUMOS-15 [36] | A **[pd]** N **[cr]** | - | 2 | 1,2,3 | No | $64 \times 64 \times 3$ $128 \times 128 \times 3$ $256 \times 256 \times 3$ | No | L1 with TV regularization | No |
| Vondrick et al. [132] (2017) | CNN AE + CNN) | Flickr Videos [131, 117], Pascal VOC [27] | N **[pd]** N **[cl]** | - | 4 | 12 | No | $64 \times 64 \times 3$ | No | Binary CE, Adv. **[f]** | No |
| Walker et al. [135] (2017) | LSTM AE + CNN AE | UCF-101 [107] | N **[pd]** | - | 12 | 20 | Yes | $64 \times 80 \times 3$ | Pose detector by Cao et al. [12] | Binary CE, L2, Adv. **[f]** | No |
| Denton et al. [24] (2017) | CNN + GAN + LSTM | moving MNIST [110], NORB [59], SUNCG [106], KTH [101] | A **[pd]** A* **[pd]** A* **[pd]** N **[pd]** | - | 5 - - 10 | 10/500 - - 20/100 | No | $64 \times 64 \times 1$ - - $128 \times 128 \times 1$ | No | CE, L2 **[f]** | [23] |
| Villegas et al. [126] (2017) | CNN LSTM + CNN AE | KTH [101], Weizmann [11], Sports-1M [50], UCF-101 [107] | N **[pd]** N **[cr]** N **[pr]** N **[pd]** | - | 10 10 4 4 | 10/20 10/20 1 1 | No | $128 \times 128 \times 1$ $128 \times 128 \times 1$ $240 \times 320 \times 3$ $240 \times 320 \times 3$ | No | L2, Adv., Gradient Difference **[f]** | [125] |
| Liang et al. [61] (2017) | GAN + CNN AE + Conv. LSTM | KITTI [34], Caltech Pedestrian [25], YouTube clips **[o]**, UCF-101 [107], THUMOS-15 [36] | N **[pr]** N **[cr]** N **[cr]** N **[pd]** N **[cr]** | - | 10 | 1/5 | No | - | No | L1, Adv., EPE **[f]** | No |
| Villegas et al. [127] (2017) | LSTM AE + CNN AE | Penn Action [140], Human3.6M [46] | N **[pd]** N **[pd]** | - | 10 10 | 32/64 64/128 | Yes | - - | Pose detector by Newell et al. [77], AlexNet (CNN) [55] | MSE pixel, MSE feature, Adv. **[f]** | No |

| | Model | Dataset | Data type | FPS | # Input frames | # Pred. frames | Extra features | Resolution | Pretrained model | Loss Function | Code |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Oliu et al. [84] (2017) | CNN + GRU AE | moving MNIST [110], | A [**pd**] | - | 10 | 10 | No | $64 \times 64 \times 1$ | No | L1 | [83] |
| | | UCF-101 [107], | N [**pd**] | 25 | 10 | 10 | | $64 \times 80 \times 1$ | | | |
| | | KTH [101] | N [**pd**] | 12'5 | 10 | 10 | | $64 \times 85 \times 3$ | | | |

**Table 4.1:** Predictive models comparative: In the Model column, comma separated elements indicate different tested proposals, an addition ($+$) indicates an architecture built using those elements. (**[p]** indicates patch size, **[o]** indicates own-designed dataset (not made publicly available), **AE** means Autoencoder and **MS** means multi-scale; in the Data Type column **A** and **N** indicate Artificial and Natural sequences respectively (**N***/**A*** means sequences built using static natural/artificial images, see notes on that model for specific information), [**pr**] is Pretrain Data, [**pd**] is data for Prediction (train and test), [**cl**] is data used for a supervised Classification task and [**cr**] is Cross Data; for Loss Function, **CE** is Cross Entropy loss and **Adv.** is Adversarial loss; **[f]** indicates the use of a fused loss using one or more combinations of the ones listed). A blank space (**-**) indicates that no implicit mention to this information was made in the paper. Regarding amount of frames, $x/y$ indicates that the models were trained to predict $x$ frames but tested predicting up to $y$ frames, $x$-$y$ indicates a variable number of frames and $x$,$y$ indicates that experiments were carried out for both $x$ and $y$ number of frames.

# Chapter 5
# Experimental setting

This section describes the experiments performed on the models that have available code online. The objective of this tests is to use some common ground prediction task and the same evaluation protocol on different models to assess which features may provide better forecasting abilities. First of all, the experiment per se is described and the different used datasets are layed out. Then, the tools used to perform the experiments are presented. Finally, the selected models and the different code changes that had to be made to adapt them to the data are discussed. After that, the next sections contain extensive comment on the obtained results.

## 5.1 The experiment

As has been discussed in the previous sections, every proposal for video prediction has an slightly different objective and approaches the task of future frame generation from a mildly distinct standpoint. For this reason, the focus of this experiment is for all the models to perform a prediction of a fixed amount of frames given the same ground truth input. In particular, to test the quality of 10 predicted frames given the 10 previous ones with the same resolution for all models. This is done with complete disregard to how actually each models performs the task: weather it is using a recurrent environment to produce one frame at a time or if all the data is fed at a time; each one uses a different loss (pixel-wise, adversarial, feature regularizations, etc.) and optimizes the learning procedure with different procedures (generally Adam [51] or RMSProp [118]), and some use curriculum learning. In general the models per se were unchanged. As it will be discussed more thoroughly in 5.3 the main modifications made to the code were adapting the data input functions as to read from the used data format and the test functions to use the proposed metrics.

For the experiments, three different datasets were chosen that represented an increasing level of complexity for the task of video prediction: Moving MNIST, KTH and UCF101.

The most simple dataset is **Moving MNIST** [110]. It depicts two white digits randomly sampled from the MNIST Digits dataset [58] on a black background. The two numbers move in the frame with linear velocities and can bounce off the walls (invert the velocity vector). This dataset is used to test the model ability to understand some simple motion, invariance to occlusion (as both digits can cross their paths) and physical bouncing dynamics. They can be generated on the fly for virtually unlimited sequences. However, for this experiment a fixed set of 1M sequences was created for training and 10K for testing. The sequences are 20 frames long (10 used as input to predict the following 10), grayscale and have a $64 \times 64$ resolution.

**Figure 5.1:** An example sequence of the Moving MNIST dataset. First row shows the input and the second one the ground truth future frames to be predicted by the models.

The second used dataset is **KTH** [101]. It represents a more challenging task than Moving MNIST as it contains humans performing different actions. In particular, it consists in 600 videos of around 15-20 seconds each depicting 25 different subjects performing 6 actions (walking, jogging, running, boxing, hand waving and hand clapping) in 4 different settings (3 outdoor and 1 indoor). The videos are all grayscale, at a resolution of $120 \times 160$ pixels and 25 fps. The dataset has been split into subjects 1 to 16 for training, and 17 to 25 for testing, resulting in 383 and 216 sequences, respectively. The frame size is reduced to $64 \times 80$ using bilinear interpolation, removing 5 pixels from the left and right borders before resizing. In many examples the sequence depict the subject leaving the scene and hence many frames are *empty* and have not been removed. It is important to note that even though this dataset has enough complexity, the fact that the background is generally blank and static allows for the models to focus in correctly modeling human motion. Thus, it is a well suited medium level dataset.

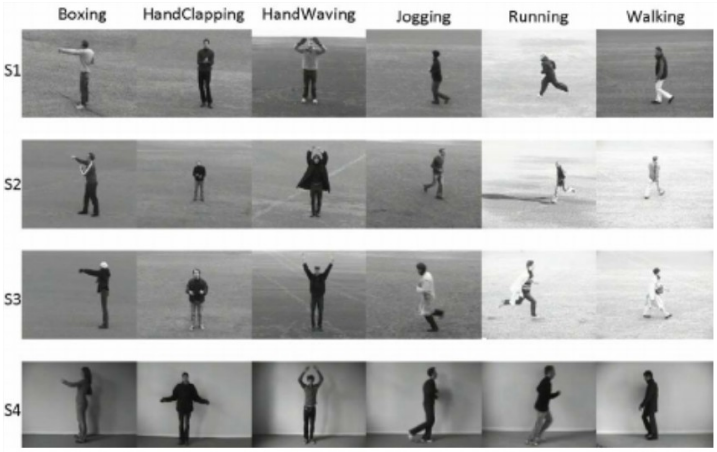


**Figure 5.2:** Example of KTH frames for the different activities. Source: [57]

Finally, the most challenging dataset considered is **UCF101** [107], displays sequences of humans performing 101 different actions (such as Military Parade, Golf Swing, Haircut, Horse Riding, Walking, Playing guitar, etc.) in RGB. It shows a high intraclass variability, containing not only human motion but also interaction of humans with different objects and other humans, different sports, etc. The standard split has been used with 9950 training sequences and 3361 test sequences. The original resolution is $240 \times 320 \times 3$ and 25 fps but is has been reduced to $64 \times 85 \times 3$ using the same procedure as in KTH. Furthermore, in order to increase motion between consecutive frames, one of every two frames was removed, hence halving the frame rate to $12'5$ fps. This dataset exhibits complex motions and backgrounds, camera motion and in

general is with different the most complex one considered.


**Figure 5.3:** Some examples of frames of the UCF101 dataset. Source: [108]

It is also important to note the sampling procedure that was used. While Moving MNIST sequences were created with length 20, the videos contained in KTH and UCF101 are usually longer. For this reason during training, a random subsequence is selected at each epoch. For KTH 20 frames from a random point are taken as input. In the case of UCF101 every other frame was taken to constitute a sub-sequence with halved frame rate. In this sense, every epoch had as many sequences as videos the training split of the dataset. Also, this can be considered as data augmentation as the amount of possible different training sequences gets greatly increased. For testing, all possible sub-sequences of length 20 were considered (with the necessary frame rate reduction of UCF101). For all the models that allowed for this parameter tuning, a batch size of 12 was used. If the code included different proposed architectures the one that reported best results in the paper was used. Apart from that, all models architectures and learning parameters were unchanged. To assess the quality of the results three different metrics were used: MSE, PSNR and DSSIM. While MSE may be seen in section 2.8 at the beginning of this document, the other two are next detailed.

### Peak Signal to Noise Ratio

PSNR is generally used to measure the quality of reconstruction of lossy compression codecs for images. It establishes a ratio between a signal and the power of the noise that affects its representation. In other words, this metric can be used to asses how an image has been affected by different perturbations, with higher values indicating a better preserved image. In the case of frame prediction it indicates how *noisy* a frame is with respect to the actual expected future. The metric value for a given $t_i$ prediction and the $y_i$ expected ground truth frame is:

$$PSNR(y_i,t_i) = 20\log_{10}(MAX_I) - 10\log_{10}(MSE(y_i,t_i)) \tag{5.1}$$

where $MAX_I$ is the maximum possible value of a pixel intensity (generally 1 for float values or 255 integers) and MSE is the Mean Squared Error seen in 2.8.

### Structural DisSimilarity

DSSIM is a variation of the SSIM index [137]. It is a method for measuring the similarity between two images. While metrics such as MSE or PSNR estimate absolute errors, SSIM is a

perception based model. In other words, it not only considers pixel values, but also other visual information in the image such as structure, luminance and contrast. It has proven to be more consistent with human quality perception. It is calculated with a sliding window approach. Take $T$ as a window in the predicted frame $t_i$ and $Y$ the equivalent window in the ground truth frame $y_i$. SSIM considers the average $\mu$, variance $\sigma^2$ and covariance $\sigma_{TY}$ to calculate the similarity among the two images:

$$SSIM(T,Y) = \frac{(2\mu_T\mu_Y + C_1) + (2\sigma_{TY} + C_2)}{(\mu_T^2 + \mu_Y^2 + C_1)(\sigma_T^2 + \sigma_Y^2 + C_2)} \tag{5.2}$$

where $C_1 = (k_1 L)^2$, $C_1 = (k_1 L)^2$ are variables to stabilize the division, $L$ is the range of possible pixel values and $k_1 = 0.01$, $k_2 = 0.03$ are the default values for two parameters of the model. For measuring the results of the different video prediction models tested in the present experiment, a sliding Gaussian window of size $11 \times 11$ and $\sigma = 1.5$ was used.

While SSIM tells how similar an image is to the expected one, DSSIM is a distance metric derived from it:

$$DSSIM(y_i, t_i) = \frac{1 - SSIM(y_i, t_i)}{2} \tag{5.3}$$

## 5.2 Tools

All the codes are generally written in Python with Tensorflow [2] or Lua with Torch7 [17]. However, it is common that every code has different dependencies at specific versions. For this reason, in order to keep a clean environment and be able to run all different codes, the Docker Engine[1] was used. Docker is a tool designed to build specific environments for applications. They use containers as opposed to virtual environments. As stated in their website: "A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings." This allows for a piece of code to run in any system able to run docker. As every available proposal of video prediction uses different versions system libraries such as Cuda and CuDNN, being able to use different environments for each of them was a big relief. For this reason, a Docker image was built for every code in order to run the experiments on the University server as well as perform the necessary modifications and tests of the code in a local level. All docker configuration files may be seen in the github of the author[2] of this project. Also, all the experiments were performed in the UB [DEPARTAMENTO Y TAL] server on a *NVIDIA GeForce GTX 1080 Ti*.

Regarding the data format, HDF5 [116] files were used. The Hierarchical Data Format is, as stated on their website, "a versatile data model that can represent very complex data objects and a wide variety of metadata". Some of the main advantages of this format is that it allows for structured data to be stored in files with no size limit, allows for any datatype, and most importantly "HDF5 data I/O functions can operate on selected subsets of the data, reducing transferred data volume and improving access speed.". This means that there is no need to load whole sequences, only the desired frames can be loaded thus reducing the memory requirements for executing an experiment. In particular, the whole of the three datasets was stored in six separate files (train and test for every dataset) containing the whole videos in NumPy [121]

[1] https://www.docker.com/

[2] https://github.com/javierselva/spatiotemporal

matrices. This specific format was chosen as it allowed access to the sequences metadata (such as the amount of frames) without the need of actually loading the sequences.

## 5.3 Selected models

In this section the models included in the comparative are discussed and some specific changes made to the code are detailed. First of all, not all the models with code available made it to the comparative. Patraucean et al. [90], Oh et al. [80], Brabandere et al. [19] and Denton et al. [23] were not included in the comparison as the available code of these proposals could not fit the experimental settings used here. Some were simplified versions of the paper models, lacked the ability to generalize to datasets not provided in their repositories or were too limited for use in real world data.

Having said that, the models that are then included for the comparative are: Srivastava [111], Mathieu [68], Finn [30], Lotter [64], Villegas [125] and Oliu [83]. The remaining of this section is focused on highlighting different details each implementation and some highlights of the modifications made. From a general standpoint, the models were adapted as to take command line parameters, read from the HDF5 files with the appropriate sampling, perform testing with the established metrics, save periodic checkpoints of the model during training and write samples as images for display purposes. However, some of the implementations already included some of this functionalities.

### 5.3.1 Srivastava

For this proposal, the composite model with conditional decoder was used. Both decoders (reconstruction and predictor) were conditioned on ground truth frames during training and previous predicted frames during testing. The batch size used was 12. It is a fully connected model, which makes the amount of parameters needed to explode as the frame size gets bigger. The input layer alone needed $64 \times 85 \times 3 = 16320$ neurons for the UCF101 dataset. This completely hampered using the number of layers that the model would have needed to model such complex data.

The code is written in Python and uses the ProtoBuf library[3] to handle the different parameters. Two separated files are used to establish model parameters and data parameters. The trained models were saved in H5 files. For Moving MNIST 2 layers with 4096 neurons each were used. For KTH and UCF101 only one layer with 2048 hidden neurons was possible due to memory requirements of the architecture. The needed input hidden neurons were also set depending on the dataset. Even though in general is not the purpose of this experiments to perform any modifications to such architectural parameters in this case it was necessary for the model to account for the whole input. On the contrary, the rest of the models use convolutions which can deal with the input with disregard of its size. Remaining parameters were left with the default values. The *lstm_combo.py* file was modified to account for different input parameters (to select training, testing or sample display). A test function was implemented based on the already present validation function. The *data_handler.py* contained the classes employed to deal with data reading and batch preparation. In particular the class *VideoPatchDataHandler* was modified to embody

[3] https://github.com/google/protobuf

the data processing for the three datasets. The code was adapted in order for it to perform the different established sequence sampling during train and test. The model already included a function for saving a sample sequence as images. This class was later refined and reused (with the necessary specific modifications) for the rest of the codes as it properly reflected the sampling procedure. Finally, a checkpoint class was created to account for automatic saves of the model and to report the training error in order to check if the learning process had converged.

### 5.3.2 Mathieu

This was probably the most challenging implementation to deal with. For both the building of the Docker image and the preparation of the code itself. This model uses Lua with Torch7 and several other libraries.The batch size was 12 and the amount of input and output frames was set to 10. It is worth mentioning that the model parameters (as the number and size of the convolutional filters used for the multi scale architecture) was not modified. As this model works with whole volumes of concatenated frames (for input and output) some fine tuning to these parameters may have had an influence on the results. However it was not the purpose of this study to perform such tweakings on the architecture of the models. The rest of learning parameters were also left as default. As they used a multi scale architecture, the implementation showed some problems when reducing the dimensionality of a frame with an odd size, in particular the width dimension of UCF101. For this reason the dimensions of that dataset had to be reduced to $64 \times 80$ by removing the excess pixels.

Regarding the modifications of the code, first of all, most of the parameters were hardcoded inside the model in the form of a parameter vector. An independent class was created to account for a more general setting in which to fit the three different datasets and its parameters. The most changes were made to the file *train_iclr_model.lua*. Some parameters were transformed into command line variables. For instance, the GPU device to be used, the modality in which the code should execute (train, test or display) or the path to the dataset file. A completely new test function was created as the included one depended too much in the original data format. Also, a function that saved a sample sequence as images was written. Finally, the data handler prepared for Srivastava was adapted and translated to Lua in order to perform the sampling of sequences. Afterwards it was further modified to fit the needs of the training implementation and other format details and fitted inside the *data_reader.lua* file.

### 5.3.3 Finn

This code is implemented in Python using Tensorflow. Together with the implementation of Villegas it was the most clean and straightforward to adapt code. It is important to mention that this code has been included in the research models of the Tensorflow library. Again, batch size was set to 12 and the length of the sequence was set to 20 frames. As this model allows for action conditioning, it was set to *false*. They propose three different models but report better results using CDNA which was the one selected for this experiment. They use schedule sampling for training the model. The rest of parameters including the number of masks to fuse the multiple predictions and the learning rate were left as default. It is important to note that this implementation is not able to deal with panoramic images, hence the input was cropped horizontally and all datasets were tested at size $64 \times 64$.

Different parameters for the experimental procedure were added to the model in order to control the behaviour of the code from the command line. For instance a parameter to control the mode of the current execution (train, test or display). The function used for training in *prediction_train.py* was reused as the base for building two separate functions for testing and data display. Finally the *prediction_input.py* file was modified following the data reader functions created for Srivastava in order for the model to read from the HDF5 files and perform the appropriate sampling. Also, here the crop necessary to adapt the frames to this architecture is performed. An automatic process to periodically save model checkpoints was already implemented.

### 5.3.4 Lotter

This code built using Python and Keras [15] was tested on top of Tensorflow. They had three separate files for train, test and settings which were all combined in one single file as separate functions. Then this file was replicated three times, one for each proposed dataset, to allow to keep most parameters as hardcoded instead of turning them in variables. Batch size was set to 12. Also, a function to save sequence samples with predictions was created.

### 5.3.5 Villegas

Similar to Finn, they used Python and Tensorflow for their implementation. They used a separate file for every dataset they tested. However, *train_s1m.py* was used to create a generic version of their model that accepted input parameters to control the different data settings used in the experiments. For instance the dataset, the GPU to use or the mode (train, test or display). The batch size was set to 12 and the input and target number of frames both to 10. The way in which the data reading was implemented in this code made it easier to adapt it to the current experiments sampling and data format than to reuse the implementation made for Srivastava as in other models. For this purpose the function *get_minibatches_idx* in the *utils.py* file was adapted to read from the HDF5 files and some slight changes were made to the training function in the file mentioned above. In a similar fashion of the problem found in the implementation of Mathieu, Villegas use multi scale residual connections between encoder and decoder. When dividing the size of the input data used by the UCF101 dataset to calculate some size parameters for the model the implementation could not cope. For this reason the dimensions of this dataset where reduced to $64 \times 80$ by removing the excess pixels.

Finally, based on the modified training file, two separate files were created for the testing process and for the display of example sequences (which was partly implemented). The implementation already included an automatic checkpoint handling process.

### 5.3.6 Oliu

This implementation was written in Python with Tensorflow. They use a separate file for the training code of each dataset. The repository also contains code for the baseline models that they used in the experiments: the Recurrent Ladder Networks [45]. Based on those, Oliu propose a modification of their model that uses bridge connections instead of the shared state. This allowed for even (slightly) better results than the proposed model for some of the experiments. However,

this was at the expense of a much greater computational power. For this reason that proposal was not included in the present comparison.

There was no need for adapting this model as their test setting was a collaboration between Oliu and the survey and experimental contents of the present project. They used the same amount of input and predicted frames, batch size, evaluating metrics, datasets and sampling procedure.

# Chapter 6
# Results

This brief chapter contains some quantitative and qualitative results of the experiments that were carried out. The performance of each model by time step is reported in Figure 6.1. Each column displays the results for a given dataset with each row being a different metric. First of all it is important to note that while MSE and PSNR are related metrics and the values showed by both go by hand, sometimes DSSIM disagrees with them. Secondly, as it has been stated in previous sections of this document, the predictions generally degrade as they get away from the last seen ground truth. This is due to uncertainty increasing with time. However, there are some exceptions to this. Lotter and Villegas were not able to learn the necessary dynamics in order to predict correct frames on the Moving MNIST dataset. They were limited to produce a completely black background on every time step. Because the background of this dataset is homogeneous, it is easier for the models to predict such black images when the training begins. That causes the learning to get stuck in a local minima that prevents further improvement. Another special case where the predictions do not seem to degrade over time is Finn. In particular for Moving MNIST (very similar to Villegas) but also to some degree in KTH. Not only does this proposal keep producing sharp results over time, it too produces the best. Regarding the results of Finn in KTH it is important to highlight the great distance it sets from the rest of the tested models. Nevertheless, there is a reason for this. As it was stated in chapter 5, the available implementation was limited to use completely square images. Even though this affects UCF101 too, it has a particularly high repercussion on the results obtained in the KTH dataset: the videos include sequences of the actors leaving the scene. Given that the backgrounds in this dataset are fairly plain it is easy in general to predict correctly those empty frames that show no motion at all. As Finn crops the input images to be squared, every frame has t less pixels on each side compared to the rest of the models. In KTH this translates as more frames where there are no actors in the scene, which could account for some of the results Finn obtained whith this data. Despite this, the qualitative results shown in Figure 6.2 still depict really sharp predictions for this model on KTH.

It also can be seen that Srivastava is a great distance away from the rest of the models, specially for real data (KTH and UCF101). As stated in chapter 5 the ammount of parameters needed for this fully connected model exploded quickly with greatly dimensional data. This did not allow for the tested architectures to be big enough in order to properly model the input and produce reasonable results. It can be seen in Figure 6.2 how the predictions completely lose their form in UCF101 and keep completely still in KTH (only increasing the bluriness). Mathieu seems to perform the worst on the simplest dataset. While keeping reasonable predictions on UCF101 and KTH, it hugely distorts MNIST digits, leaving nothing but white stains on a black

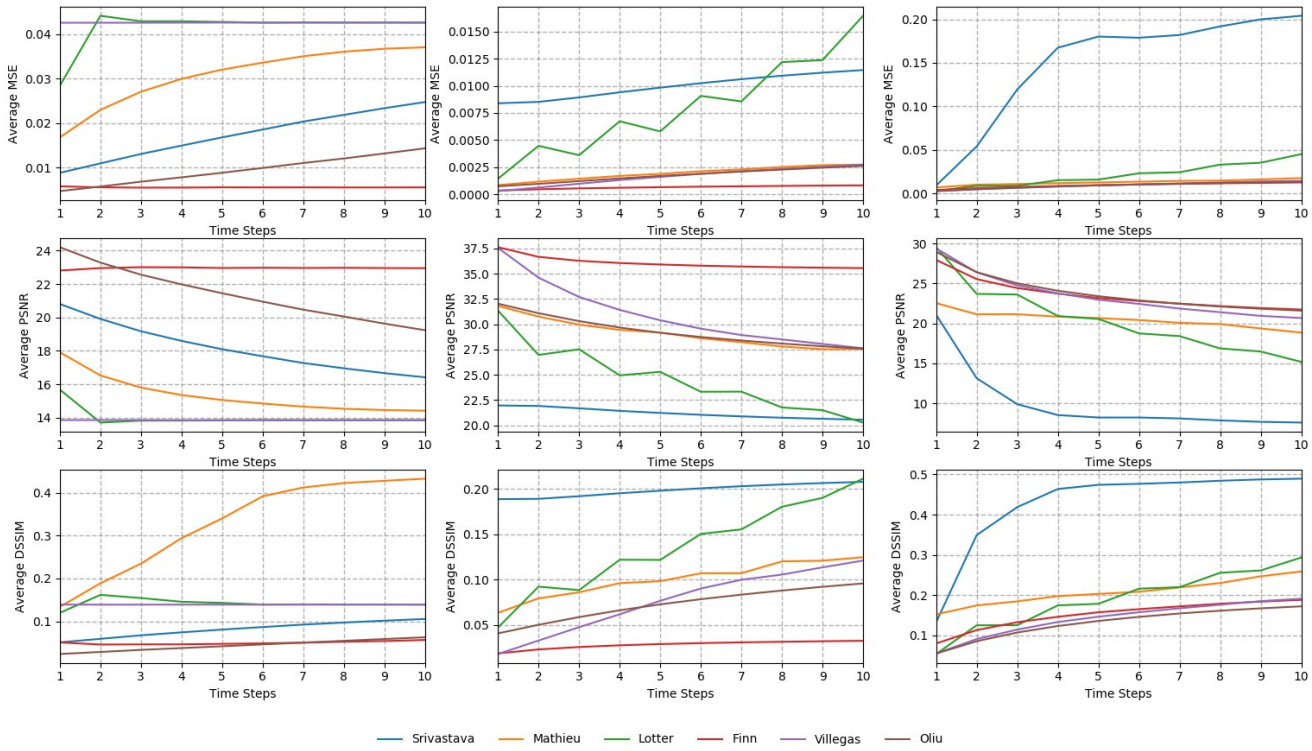


**Figure 6.1:** Comparative of results obtained on Moving MNIST (*Left*), KTH (*Center*) and UCF101 (*Right*) depending on the number of time steps since the last seen ground truth frame. From top to bottom: MSE, PSNR and DSSIM

background. Finally, it is worth of metion the stepped shape that exhibit the prediction results of Lotter. That helps the first generated frames to maintain a low prediction error. However, they end up generating artifacts that are further distorted with time. All these fenomena will be discussed in the following chapter.

| Model | Moving MNIST | | | KTH | | | UCF101 | | |
|---|---|---|---|---|---|---|---|---|---|
| | **MSE** | **PSNR** | **DSSIM** | **MSE** | **PSNR** | **DSSIM** | **MSE** | **PSNR** | **DSSIM** |
| **Srivastava** | 0.01737 | 18.183 | 0.08164 | 0.00995 | 21.220 | 0.19860 | 0.14866 | 10.021 | 0.42555 |
| **Mathieu** | 0.03071 | 15.361 | 0.32770 | 0.00194 | 29.097 | 0.10018 | 0.01287 | 20.492 | 0.20730 |
| **Lotter** | 0.04137 | 14.017 | 0.14201 | 0.00807 | 24.635 | 0.13588 | 0.02124 | 20.398 | 0.19013 |
| **Finn** | **0.00561** | **22.986** | 0.04985 | **0.00065** | **36.101** | **0.02790** | 0.00940 | 23.601 | 0.15196 |
| **Villegas** | 0.04254 | 13.857 | 0.13896 | 0.00165 | 30.946 | 0.07657 | 0.00940 | 23.457 | 0.14150 |
| **Oliu** | 0.00947 | 21.386 | **0.04376** | 0.00175 | 29.299 | 0.07251 | **0.00908** | **23.872** | **0.13055** |

**Table 6.1:** Average results over 10 time steps.

It is clear from Table 6.1 that Finn is the ultimate winner only challenged by Oliu, which in turn is closely followed by Villegas (except in Moving MNIST). It is important to highlight the fact that different metrics represent the results differently. While Finn has higher MSE and PSNR in Moving MNIST, Oliu performed better regarding DSSIM. Similarly Villegas got higher PSNR

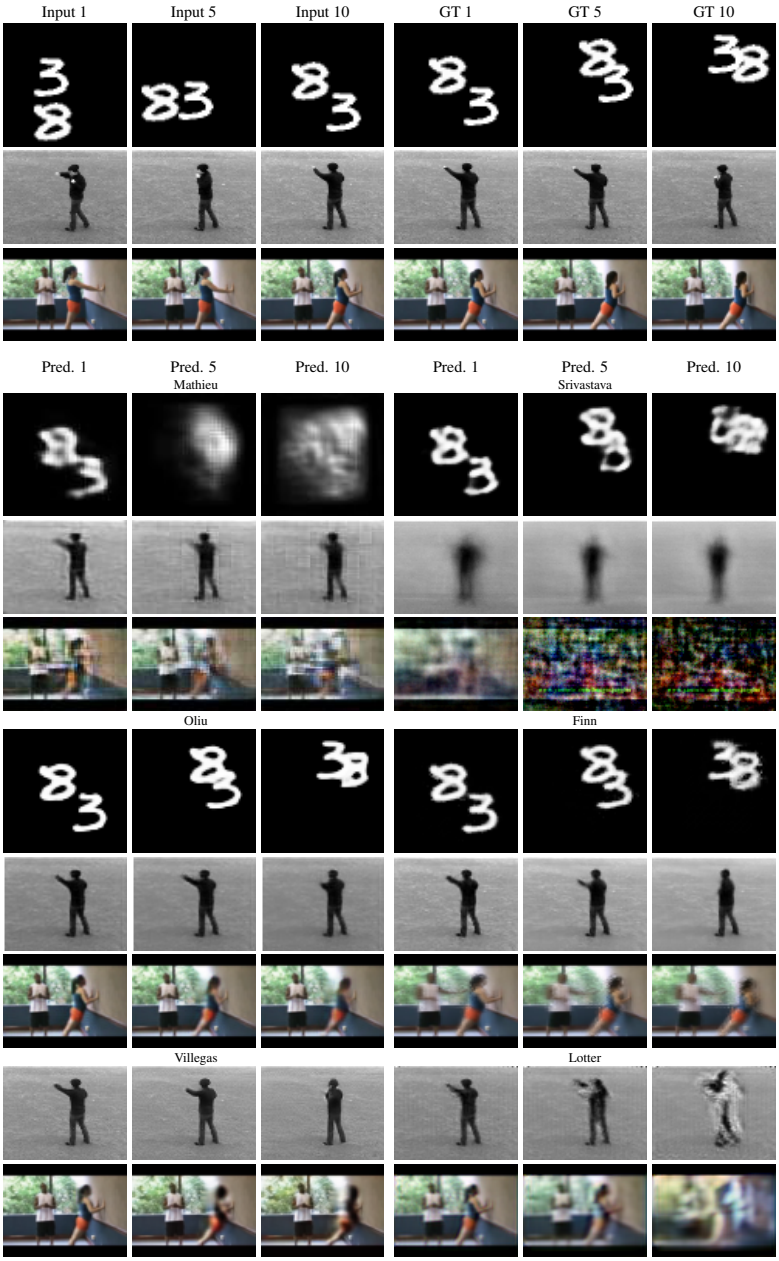


**Figure 6.2:** Predictions at 1, 5, and 10 time steps from the last ground truth frame for all models and considered datasets.

than Oliu but obtained worst results in DSSIM. In summary, Finn, Oliu and Villegas produce very close results (except for the inability of Villegas to produce valid frames in the Moving MNIST and the fact that Finn is using a cropped central version of KTH). These models have used some interesting techniques that were detailed in chapter 3: using feedback during training, avoid feedback at all or disentangling motion and content are some examples. The explicit use that the models make of them and the relationship to specific results will be discussed in the following chapter.

In Figure 6.2 one example is shown for each dataset. The first, fifth and tenth predicted

frames are used for a simple comparison. For a more extensive presentation of the results see Appendix A. It contains the per frame results used to build Figure 6.1 in table form and qualitative results with more complete predicted sequences for every model and dataset. The Moving MNIST frames generated by Villegas and Lotter were not included as they were totally black. It can be seen that while Oliu and Finn accurately predict the future adding just a bit of blurriness. Villegas is capable of good predictions in KTH where the background is easy to model but finds it harder on UCF101. Lotter makes a good first prediction but artifacts appear early and are quickly magnified by the model. Mathieu greatly distorts the predictions, producing almost no motion. Srivastava works fine for Moving MNIST, copies a blurry version of the input for KTH and is not able to deal with UCF101.

# Chapter 7
# Discussion

The focus of this section is to analyse the obtained results in relation with the models architecture and the implementation of the different techniques for improved prediction. Even though the topics discussed here may be understood by the evidence provided in the previous chapter, it is recommended to check the more extensive results of Appendix A. It will help provide a better appreciation of the small details. Specially for qualitative evaluation.

**Moving MNIST** is the most simple proposed dataset. The objective of using this dataset was to assess weather the model was capable of learning simple motion dynamics. Even though Srivastava, Finn and Oliu successfully solved this task, Mathieu distorted the prediction from an early stage and Lotter and Villegas learned to predict black frames. The three successful models all include a way of providing some sort of identity function from input to prediction to start with. Srivastava uses the composite model to reconstruct the input at the same time, thus forcing the model to learn features that enable the data to be transformed back from feature space. Finn implicitly predict transformations over the last frame pixels. Finally Oliu share states between encoder and decoder in a layerwise fashion, so at the beginning of training the first layers contain either the input frame or a convolved representation of the same. Regarding the failing models, despite all being (fairly) good on the other datasets, they lack any of these capabilities and it is much easier for the losses to guide towards whole black frames. Moving MNIST shows most of the frame being a static, slate black screen, thus predicting the whole frame as black is correct for most of the pixels. Oliu et al. [84] showed how performing pretraining on KTH helps these models learn some inital weights to later train effectively on Moving MNIST.

When comparing the models that do predict, Oliu and Finn produce the finest results. While the generated frames by Oliu look sharper around the digits edges, Finn is slightly more stable over time. At least regarding PSNR and MSE, because DSSIM likes them both practically the same. Actually, the fact that this happens helps explain the idea that pixel distance metrics produce blurry results. Being Finn generations slightly blurrier and still obtaining higher MSE and PSNR values demonstrates it. The predictions by Oliu are sharper but probably slightly deviated from the actual ground truth (by just some pixels). Finn solves this by blurring and hence spanning the digits a bit further, helping the prediction to be *closer* to the ground truth. Despite this, DSSIM, which takes into account the actual structure of the produced frame, regards them both as good results. Both of them are able to predict up to the end of the sequence without error propagation. Finn by using feedback during training with curriculum learning and Oliu by unrolling the decoder. They are also capable of separating the two digits, still recognizing them after the trajectories cross. Regarding Srivastava, even though proposing to use a CNN to extract the features to be fed to the prediction model, the used implementation did not include

it. Also, despite having the decoder unrolled to predict without the need to use the encoder, the pixel wise loss being the only supervision causes notably blurry predictions. Specially as uncertainty increases further in the future. Finally, the use of single or two layer architecture keeps the model from learning more complex dynamics needed to properly handle, for instance, occlusion invariance. This causes the digits to distort when they cross paths. Nevertheless, given the problematics that seem to have arisen from such a simple dataset, they generate fairly good predictions.

**KTH** is the first big step towards a more complex dataset. The simpler model of Srivastava is unable to predict at all. A single LSTM fully connected layer is not able to account for the complexity of this more rich data. As can be seen in the sequences (Appendix A) it can barely copy the last frame and keep blurring over time. This demonstrates that the task of frame prediction requires a minimum of feature abstraction. Probably the same model with the ImageNet CNN would have been able to produce some reasonably results on this data. Despite the fact that Mathieu also produces very blurry predictions, it seems to have learnt some motion dynamics. The model understands where in the frame is motion happening and diverges the blurriness towards that direction. For instance, in sequences where a person is running, the distorted blob seems to follow the direction correctly. Other example is when a subject is standing still and moving the arms: the model distorts the arm region over time but leaves the rest untouched. In this sense it seems like the model knows where the motion is happening but it can get over the uncertainty in order to produce a sharper prediction. Even though it has been demonstrated that an adversarial loss helps produce better predictions, it can only account for doing so up to a point. GANs are here to help, but they are not responsible for teaching the model proper motion prediction dynamics. A model needs to be able to produce some reasonable predictions in order to enable a discriminator to guide it. Lotter produce a reasonable first prediction. However, they train the model on ground truth frames only, and when the model has to use feedback prediction during testing it greatly magnifies appeared artifacts. Having the model not learned to be tolerant to such errors it further increases them. In some way, the model by Lotter seems to be able to swing back and forth forming steps that keep the error similar to the one in the previous prediction for every two. Even though qualitatively it looks like a gradual degradation, the results by all the metrics display this behaviour. Lotter degrades so much the prediction by enhancing the artifacts that the generated frames end up too far from ground truth, resulting in results worst than Mathieu.

Regarding the other three models, they again seem to produce better results. Specially Finn, for whom part of the credit may be attributed to the fact that the model is working with smaller predictions. However, that could explain overall good results (on average). But they go further, showing some qualitative results not reached by the rest of the models. Villegas and Oliu, who in contrast use smaller models, attain still very good results. However, Finn is better able to produce predictions that do not degrade over time. Even though the disentangled model of Villegas produces better results in the first frames, they degrade quickly as feedback predictions are difficult to account for. Even though Villegas used them during training, using an unrolled decoder as in Oliu seems to give them a higher persistence to errors over time. This is reflected on Oliu performing better for the last frames of the sequence when compared to Villegas. Specially with the DSSIM metric, although PSNR shows too a quicker degradation of results by the latter. Here it is again possible to see the main contrast between image difference metrics and DSSIM. While Mathieu and Oliu obtain very similar results on PSNR for KTH, qualitative results demonstrate

that the latter predicts much better, which is greatly reflected by the dissimilarity metric.

Finally, **UCF101** represents the most challenging task. Including more diverse content both in moving elements and background, colored images and camera motion. Srivastava was definitely not able to go this far. The fully connected architecture was a bottleneck and the amount of parameters that such an architecture would have needed to model this sequences successfully would have been too much. Mathieu and Lotter behave in a very similar way to what they did for KTH. Mathieu predicts frames which depict almost no motion, with increased blurriness over time in the regions that are moving. It is highly possible that a slightly bigger and better designed architecture for the CNN model would have allowed this proposal to produce better results. Lotter still generates and intensifies artifacts over the predictions, ending up the sequence with an almost completely distorted frame.

By contrast Finn, Oliu and Villegas seem to come closer for this one. This actually means that Finn is no longer producing results so remarkable, but just as good as Oliu. In fact, for this dataset the sharpness that the latter seemed to have in Moving MNIST is highlighted again. While Villegas ends up accumulating blurriness with increasing uncertainty over the predictions, Finn and Oliu produce rather consistent results over time. The fact that Finn is no longer producing such good results as it did on KTH may have something to do with background handling. Moving MNIST and KTH have for all the sequences a very much similar (if not the same) background: plain, displaying almost no features and not moving. This allows for a straightforward separation of moving elements from static parts, which results in very accurate predictions. Even though it was finally not included for comparison, it is worth mentioning here the work of Denton et al. [24] who also tested on KTH and Moving MNIST obtaining impressive results for over 100 time steps. However, their two stream encoder for separately modeling motion and content extracted such features from static frames. The fact that it was so successful on all the experiments they perform may have been related with this. The model learns a representation not so explicit of motion but of the parts that usually move in the specific data it has been trained on. This is more difficult to do in UCF101. However, Finn does extract motion features in a recurrent manner, which has allowed them to produce very good results still. Nevertheless, Oliu obtains the better results in this dataset.

In general Finn and Oliu seem the better models for longer sequences. One profits from a scheduled learning with feedback predictions while the other completely avoids re encoding predictions. The fact that Finn shows better results in Moving MNIST and KTH is probably due to the motion/content disentanglement they do thanks to the masked transformation predictor. Allowing to separate background from foreground really boosts up prediction results. Specially in datasets with simple background as those two. Both Finn and Villegas explicitly separate motion and content, and both use feedback predictions during training. The main difference is that Finn uses curriculum learning, which seems important to gradually teach the model to work with longer sequences and better handle uncertainty. Also, the way in which content and motion are dealt with. On one hand Villegas uses difference images and a content convoluted representation which are concatenated before prediction. On the other hand Finn uses Convolutional LSTM AE to predict different patch transformations and several masks to apply the appropriate motion to the correct pixels. Furthermore, it explicitly handles background and foreground by using a specific mask and allows for hallucinating new pixels. It also uses residual connections between encoder and decoder. In summary, it seems like they have done everything that could improve prediction quality. It is possible that Villegas is losing some information when combining both

content and motion features before generating the prediction. It is, nevertheless, important to highlight that the model by Finn is quite big with a huge amount of parameters, while Villegas uses a more moderate architecture.

Finn and Oliu may be regarded as some kind of Ying-Yang. On one hand, the former advocated for an engineering approach and built a complex machine that uses several components (Conv. LSTM AE with recurrent connections to predict several motion patches which are later dynamically masked to form a prediction which can also hallucinate pixels) and is trained gradually to longer sequences and learns to get robust to its own predictions. The model is, in the end, using a lot of parameters, which take longer to train. On the other hand, Oliu decided to build a simpler yet efficient model which completely shared information between encoder and decoder, and some Convolutional layers to extract the necessary features. Also the training procedure and loss used were simpler. Nevertheless, both models obtain comparable results.

# Chapter 8
# Conclusions and future work

The present document has presented an extensive and comprehensive analysis of the problem of future frame prediction. The SOTA has been thoroughly examined to include every proposal made to solve this problem. There are many different approaches to video prediction. Recurrent models and Convolutional Networks have proven to be necessary and essential in order to properly model the time and space in this task. Even though the proper way of combining these two elements to produce good results may vary, it is clear that inspiration in the human brain offers an interesting research direction. For this reason, techniques that allow the separation of different levels of information in the data seem to work better. In general, Auto Encoders allow for representing bottom-up and top-down processes of the Predictive Coding process and any kind of information sharing between the different layers of these two allow for feature stratification. Also, different used techniques for training the models such as scheduled sampling or pre training for reconstruction have proven effective to stabilize the learning procedure.

However, there is in general a lack of common test ground for this kind of models. Not only it is difficult to asses the results, as future frame prediction is a highly uncertain task (due to its multi modality), but it is not very common that these models compare effectively against each other. Given the high variability of approaches for this task it results very complex to measure their achievements from a single point of view. For this reason, it might be necessary to start valuing video prediction by its effectiveness to perform different tasks and comparing to other approaches that try to solve those tasks too. For example generalization capabilities to learn unsupervised features useful for other tasks or an explicit use of future visual prediction for agent action planning.

After the completion of this project there are many doubts still in the air. Are GAN really useful for frame prediction or they only ensure sharp looking results? Can this task really be used to improve the visual system of autonomous agents or, on the contrary, it is overdoing it? May architectures for video prediction be extended to other kinds of future prediction?

For all these unanswered questions there is much future work to be done on future frame generation:

- First of all, further test and compare the models that could not be used for comparison in the current project in order to learn what their proposals have to offer.
- Secondly, use all the knowledge learned during the development of this survey to propose and build an effective model of video prediction.
- The use of such model for solving different applications on a real world environment.

- Development of a valid way of measuring future prediction results.

# References


[1] The liberated pixel cup. http://lpc.opengameart.org. [Online; accessed 03-May-2017].

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. 2015.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. December 2017.

[5] João Ascenso, Catarina Brites, and Fernando Pereira. Improving frame interpolation with spatial motion smoothing for pixel domain distributed video coding. 01 2005.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253–279, jun 2013.

[7] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS'15, pages 1171–1179, Cambridge, MA, USA, 2015. MIT Press.

[8] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In International Conference on Machine Learning, ICML, 2009.

[9] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2):157–166, Mar 1994.

[10] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, Advances in Neural Information Processing Systems 19, pages 153–160. MIT Press, 2007.

[11] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. In Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1, volume 2, pages 1395–1402 Vol. 2, Oct 2005.

[12] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In CVPR, 2017.

[13] Silvia Chiappa, Sébastien Racanière, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. CoRR, abs/1704.02254, 2017.

[14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. CoRR, abs/1406.1078, 2014.

[15] François Chollet et al. Keras. https://github.com/keras-team/keras, 2015.

[16] Francois Chollet. Building autoencoders in keras, 2016. [Online; accessed 13-Jan-2018].

[17] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In BigLearn, NIPS Workshop, 2011.

[18] Francesco Cricri, Xingyang Ni, Mikko Honkala, Emre Aksu, and Moncef Gabbouj. Video ladder networks. CoRR, 2016.

[19] Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Code: Dynamic filter networks, 2016. [Online; accessed 27-Dec-2017].

[20] Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. In NIPS, 2016.

[21] Hanneke Den Ouden, Peter Kok, and Floris De Lange. How prediction errors shape perception, attention, and motivation. Frontiers in Psychology, 3:548, 2012.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09, 2009.

[23] Emily L Denton and vighnesh Birodkar. Code: Unsupervised learning of disentangled representations from video, 2016. [Online; accessed 06-Jan-2018].

[24] Emily L Denton and vighnesh Birodkar. Unsupervised learning of disentangled representations from video. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 4417–4426. Curran Associates, Inc., 2017.

[25] Piotr Dollár, Christian Wojek, Bernt Schiele, Pietro Perona, and Tu Darmstadt. Pedestrian detection: A benchmark. In In CVPR, 2009.

[26] Vincent Dumoulin. Convolution arithmetic, 2016. [Online; accessed 13-Jan-2018].

[27] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. International Journal of Computer Vision, 88(2):303–338, Jun 2010.

[28] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Push dataset, 2016. [Online; accessed 13-Jan-2018].

[29] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 64–72. Curran Associates, Inc., 2016.

[30] Chelsea Finn, Ian J. Goodfellow, and Sergey Levine. Code: Unsupervised learning for physical interaction through video prediction, 2016. [Online; accessed 26-Dec-2017].

[31] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. CoRR, abs/1506.06825, 2015.

[32] Peter Földiák. Learning invariance from transformation sequences. Neural Computation, 3(2):194–200, 1991.

[33] Z. Gan, C. Li, R. Henao, D. Carlson, and L. Carin. Deep temporal sigmoid belief networks for sequence modeling. In NIPS, 2015.

[34] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. International Journal of Robotics Research (IJRR), 2013.

[35] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc., 2014.

[36] A. Gorban, H. Idrees, Y.-G. Jiang, A. Roshan Zamir, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes. http://www.thumos.info/, 2015.

[37] R. Goroshin, J. Bruna, J. Tompson, D. Eigen, and Y. LeCun. Unsupervised learning of spatiotemporally coherent metrics. In 2015 IEEE International Conference on Computer Vision (ICCV), pages 4086–4093, Dec 2015.

[38] R. Goroshin, J. Bruna, J. Tompson, D. Eigen, and Y. LeCun. Unsupervised learning of spatiotemporally coherent metrics. In 2015 IEEE International Conference on Computer Vision (ICCV), pages 4086–4093, Dec 2015.

[39] Ross Goroshin, Michael Mathieu, and Yann LeCun. Learning to linearize under uncertainty. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS'15, pages 1234–1242, Cambridge, MA, USA, 2015. MIT Press.

[40] Nicolas Gourier, Daniela Hall, and James L. Crowley. Estimating face orientation from robust detection of salient facial structures. In FG NET WORKSHOP ON VISUAL OBSERVATION OF DEICTIC GESTURES, 2004.

[41] Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton. Wikimedia commons: Peephole long short-term memory, 2017. [Online; accessed 13-Jan-2018].

[42] J. H. van Hateren and A. van der Schaaf. Independent component filters of natural images compared with simple cells in primary visual cortex. Proceedings: Biological Sciences, 265(1394):359–366, Mar 1998.

[43] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Comput., 9(8):1735–1780, November 1997.

[44] Andrew Hollingworth. Constructing visual representations of natural scenes: the roles of short- and long-term visual memory. Journal of experimental psychology. Human perception and performance, 30 3:519–37, 2004.

[45] Alexander Ilin, Isabeau Prémont-Schwarz, Tele Hotloo Hao, Antti Rasmus, Rinu Boney, and Harri Valpola. Recurrent ladder networks. CoRR, 2017.

[46] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. IEEE Transactions on Pattern Analysis and Machine Intelligence, 36(7):1325–1339, jul 2014.

[47] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems 28, pages 2017–2025. Curran Associates, Inc., 2015.

[48] Nishchal K. Verma. Future image frame generation using artificial neural network with selected features. pages 1–8, 10 2012.

[49] Nal Kalchbrenner, Aäron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. CoRR, 2016.

[50] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In CVPR, 2014.

[51] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. CoRR.

[52] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. CoRR, 2013.

[53] B. Klein, L. Wolf, and Y. Afek. A dynamic convolutional layer for short range weather prediction. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4840–4848, June 2015.

[54] A. Kosaka and A. Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 3, pages 2177–2186, Jul 1992.

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.

[56] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: a large video database for human motion recognition. In In Proceedings of the International Conference on Computer Vision (ICCV), 2011.

[57] Ivan Laptev and Barbara Caputo. Recognition of human actions, 2005. [Online; accessed 14-Jan-2018].

[58] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In Intelligent Signal Processing, pages 306–351. IEEE Press, 2001.

[59] Y. LeCun, Fu Jie Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004., volume 2, pages II–97–104 Vol.2, June 2004.

[60] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object Recognition with Gradient-Based Learning, pages 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[61] Xiaodan Liang, Lisa Lee, Wei Dai, and Eric P. Xing. Dual motion gan for future-flow embedded video prediction. In Proceedings of the IEEE International Conference on Computer Vision, 2017.

[62] Ziwei Liu, Raymond Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video frame synthesis using deep voxel flow. In Proceedings of International Conference on Computer Vision (ICCV), October 2017.

[63] William Lotter, Gabriel Kreiman, and David Cox. Unsupervised learning of visual structure using predictive generative networks. CoRR, abs/1511.06380, 2015.

[64] William Lotter, Gabriel Kreiman, and David Cox. Code: Deep predictive coding networks for video prediction and unsupervised learning, 2016. [Online; accessed 28-May-2017].

[65] William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. ICLR, 2016.

[66] William Lotter, Gabriel Kreiman, and David D. Cox. Unsupervised learning of visual structure using predictive generative networks. CoRR, abs/1511.06380, 2015.

[67] Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. 2015.

[68] Michaël Mathieu, Camille Couprie, and Yann LeCun. Code: Deep multi-scale video prediction beyond mean square error, 2016. [Online; accessed 24-May-2017].

[69] Roland Memisevic and Georgios Exarchakis. Learning invariant features by harnessing the aperture problem. In Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13, pages III–100–III–108. JMLR.org, 2013.

[70] V. Michalski, R. Memisevic, and Kishore Konda. Modeling deep temporal dependencies with recurrent "grammar cells". In Advances in Neural Information Processing Systems 27, 2014.

[71] Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Generalizable features from unsupervised learning. ICLR'17, 2016.

[72] Roni Mittelman, Benjamin Kuipers, Silvio Savarese, and Honglak Lee. Structured recurrent temporal restricted boltzmann machines. In Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, ICML'14, pages II–1647–II–1655. JMLR.org, 2014.

[73] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In NIPS Deep Learning Workshop. 2013.

[74] Hossein Mobahi, Ronan Collobert, and Jason Weston. Deep learning from temporal coherence in video. In Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, pages 737–744, New York, NY, USA, 2009. ACM.

[75] D. Mumford. On the computational architecture of the neocortex. Biological Cybernetics, 66(3):241–251, Jan 1992.

[76] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked Hourglass Networks for Human Pose Estimation, pages 483–499. Springer International Publishing, Cham, 2016.

[77] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. CoRR, abs/1603.06937, 2016.

[78] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ICCV '15, pages 1520–1528, Washington, DC, USA, 2015. IEEE Computer Society.

[79] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-Conditional Video Prediction using Deep Networks in Atari Games. In C Cortes, N D Lawrence, D D Lee, M Sugiyama, R Garnett, and R Garnett, editors, Advances in Neural Information Processing Systems (NIPS) 28, pages 2845–2853. Curran Associates, Inc., 2015.

[80] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. CODE: Action-Conditional Video Prediction using Deep Networks in Atari Games, 2015. [Online; accessed 24-May-2017].

[81] Christopher Olah. Understanding convolutions, 2014. [Online; accessed 13-Jan-2018].

[82] Christopher Olah. Understanding lstm networks, 2015. [Online; accessed 13-Jan-2018].

[83] Marc Oliu, Javier Selva, and Sergio Escalera. Code: Folded recurrent neural networks for future video prediction, 2017. [Online; accessed 07-Jan-2018].

[84] Marc Oliu, Javier Selva, and Sergio Escalera. Folded recurrent neural networks for future video prediction, 2017.

[85] Bruno A. Olshausen and David J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? Vision Research, 37(23):3311 – 3325, 1997.

[86] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In CVPR, 2014.

[87] Randall O'Reilly, Dean Wyatte, and John Rohrlich. Learning through time in the thalamocortical loops. CoRR, 07 2014.

[88] R. B. Palm. Prediction as a candidate for learning deep hierarchical models of data, 2012. Supervised by Associate Professor Ole Winther, owi@imm.dtu.dk, DTU Informatics, and Morten Mørup, mm@imm.dtu.dk, DTU Informatics.

[89] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. CoRR, 2016.

[90] Viorica Pătrăucean, Ankur Handa, and Roberto Cipolla. Code: Spatio-temporal video autoencoder with differentiable memory, 2016. [Online; accessed 28-May-2017].

[91] Viorica Pătrăucean, Ankur Handa, and Roberto Cipolla. Spatio-temporal video autoencoder with differentiable memory. In International Conference on Learning Representations (ICLR) Workshop, 2016.

[92] Silvia L. Pintea, Jan C. van Gemert, and Arnold W. M. Smeulders. Déjà Vu: Motion Prediction in Static Images, pages 172–187. Springer International Publishing, 2014.

[93] Marc'Aurelio Ranzato, Arthur Szlam, Joan Bruna, Michaël Mathieu, Ronan Collobert, and Sumit Chopra. Video (language) modeling: a baseline for generative models of natural videos. CoRR, abs/1412.6604, 2014.

[94] Rajesh Rao and Dana H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. 2:79–87, 02 1999.

[95] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. In Computer Vision and Pattern Recognition, 2015.

[96] A. J. Robinson and Frank Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK, 1987.

[97] H. Sakaino. Spatio-temporal image pattern prediction method based on a physical model with time-varying optical flow. IEEE Transactions on Geoscience and Remote Sensing, 51(5):3023–3036, May 2013.

[98] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. CoRR, 2016.

[99] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof. Prost: Parallel robust online simple tracking. In 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 723–730, June 2010.

[100] Paul Sastrasinh. Dense realtime optical flow on the gpu, 2011. [Online; accessed 13-Jan-2018].

[101] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local svm approach. In Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3 - Volume 03, ICPR '04, pages 32–36, Washington, DC, USA, 2004. IEEE Computer Society.

[102] Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Code: Convolutional lstm network: A machine learning approach for precipitation nowcasting, 2015. [Online; accessed 14-Dec-2017].

[103] Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems (NIPS) 28, pages 802–810. Curran Associates, Inc., 2015.

[104] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. ICLR'15, 2014.

[105] William R. Softky. Unsupervised pixel-prediction. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, Advances in Neural Information Processing Systems 8, pages 809–815. MIT Press, 1996.

[106] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[107] K. Soomro, A. Roshan Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. In CRCV-TR-12-01, 2012.

[108] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101 - action recognition data set, 2011. [Online; accessed 14-Jan-2018].

[109] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15:1929–1958, 2014.

[110] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In Francis Bach and David Blei, editors, Proceedings of the 32nd International Conference on Machine Learning (ICML-15), volume 37 of Proceedings of Machine Learning Research, pages 843–852, Lille, France, 07–09 Jul 2015. PMLR.

[111] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Code: Unsupervised learning of video representations using lstms, 2015. [Online; accessed 24-May-2017].

[112] Ilya Sutskever and Geoffrey Hinton. Learning multilevel distributed representations for high-dimensional sequences. In Marina Meila and Xiaotong Shen, editors, Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, volume 2 of Proceedings of Machine Learning Research, pages 548–555, San Juan, Puerto Rico, 21–24 Mar 2007. PMLR.

[113] Ilya Sutskever and Geoffrey Hinton. Learning multilevel distributed representations for high-dimensional sequences. In Marina Meila and Xiaotong Shen, editors, Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, volume 2 of Proceedings of Machine Learning Research, pages 548–555, San Juan, Puerto Rico, 21–24 Mar 2007. PMLR.

[114] Ilya Sutskever, Geoffrey E Hinton, and Graham W. Taylor. The recurrent temporal restricted boltzmann machine. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, Advances in Neural Information Processing Systems 21, pages 1601–1608. Curran Associates, Inc., 2009.

[115] Ernő Téglás, Edward Vul, Vittorio Girotto, Michel Gonzalez, Joshua B. Tenenbaum, and Luca L. Bonatti. Pure reasoning in 12-month-old infants as probabilistic inference. Science, 332(6033):1054–1059, 2011.

[116] The HDF Group. Hierarchical data format version 5, 2000-2010.

[117] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. The new data and new challenges in multimedia research. CoRR, abs/1503.01817, 2015.

[118] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[119] Joost R. van Amersfoort, Anitha Kannan, Marc'Aurelio Ranzato, Arthur Szlam, Du Tran, and Soumith Chintala. Transformation-based models of video sequences. CoRR, 2017.

[120] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In ICML, volume 48 of JMLR Workshop and Conference Proceedings, pages 1747–1756. JMLR.org, 2016.

[121] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. Computing in Science Engineering, 13(2):22–30, March 2011.

[122] N. K. Verma and Shimaila. Generation of future image frames using adaptive network based fuzzy inference system on spatiotemporal framework. In 2012 IEEE Applied Imagery Pattern Recognition Workshop (AIPR), pages 1–8, Oct 2012.

[123] N. K. Verma and S. Singh. Generation of future image frames using optical flow. In 2013 IEEE Applied Imagery Pattern Recognition Workshop (AIPR), pages 1–7, Oct 2013.

[124] Roberto Vezzani and Rita Cucchiara. Video surveillance online repository (visor): an integrated framework. Multimedia Tools and Applications, 50(2):359–380, 2010.

[125] Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. Code: Decomposing motion and content for natural video sequence prediction, 2016. [Online; accessed 06-Jan-2018].

[126] Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. Decomposing motion and content for natural video sequence prediction. 06 2017.

[127] Ruben Villegas, Jimei Yang, Yuliang Zou, Sungryull Sohn, Xunyu Lin, and Honglak Lee. Learning to generate long-term future via hierarchical prediction. 2017.

[128] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th International Conference on Machine Learning, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

[129] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Code: Generating videos with scene dynamics, 2016. [Online; accessed 28-May-2017].

[130] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 613–621. Curran Associates, Inc., 2016.

[131] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. CoRR, abs/1609.02612, 2016.

[132] Carl Vondrick and Antonio Torralba. Generating the future with adversarial transformers. 07 2017.

[133] Dana Vrajitoru. Teaching section of dana's vrajitoru website, 2011. [Online; accessed 13-Jan-2018].

[134] Vedran Vukotić, Silvia-Laura Pintea, Christian Raymond, Guillaume Gravier, and Jan Van Gemert. One-step time-dependent future video frame prediction with a convolutional encoder-decoder neural network. In Netherlands Conference on Computer Vision (NCCV), 2016.

[135] Jacob Walker, Kenneth Marino, Abhinav Gupta, and Martial Hebert. The pose knows: Video forecasting by generating pose futures. In The IEEE International Conference on Computer Vision (ICCV), Oct 2017.

[136] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. CoRR, abs/1505.00687, 2015.

[137] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing, 13(4):600–612, April 2004.

[138] William F. Whitney, Michael Chang, Tejas Kulkarni, and Joshua B. Tenenbaum. Understanding visual concepts with continuation learning. arXiv preprint arXiv:1502.04623, 2016.

[139] Tianfan Xue, Jiajun Wu, Katherine L Bouman, and William T Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In NIPS, 2016.

[140] W. Zhang, M. Zhu, and K. G. Derpanis. From actemes to action: A strongly-supervised representation for detailed action understanding. In 2013 IEEE International Conference on Computer Vision, pages 2248–2255, Dec 2013.

[141] Mingmin Zhao, Chengxu Zhuang, Yizhou Wang, and Tai Sing Lee. Predictive encoding of contextual relationships for perceptual inference, interpolation and prediction. In ICLR'15, 2014.

[142] Yipin Zhou and Tamara L. Berg. Learning temporal transformations from time-lapse videos. In ECCV, 2016.

Appendix A

# Apendix A

This Appendix contains more extensive results of the performed experiments. First, detailed per frame results by every model on all the datasets.

## A.1 Per frame results

| | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 | avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MSE | | | | | | | | | | | |
| Srivastava | 0.00885 | 0.01097 | 0.01308 | 0.01496 | 0.01680 | 0.01856 | 0.02035 | 0.02185 | 0.02335 | 0.02473 | 0.01737 |
| Mathieu | 0.01681 | 0.02298 | 0.02704 | 0.02993 | 0.03201 | 0.03356 | 0.03500 | 0.03602 | 0.03668 | 0.03701 | 0.03071 |
| Lotter | 0.02838 | 0.04408 | 0.04285 | 0.04285 | 0.04270 | 0.04255 | 0.04258 | 0.04255 | 0.04256 | 0.04254 | 0.04137 |
| Finn | 0.00582 | 0.00560 | 0.00553 | 0.00554 | 0.00560 | 0.00558 | 0.00559 | 0.00557 | 0.00557 | 0.00559 | **0.00561** |
| Villegas | 0.04251 | 0.04252 | 0.04252 | 0.04254 | 0.04255 | 0.04253 | 0.04254 | 0.04255 | 0.04255 | 0.04255 | 0.04254 |
| Oliu | 0.00475 | 0.00578 | 0.00686 | 0.00784 | 0.00887 | 0.00994 | 0.01105 | 0.01207 | 0.01319 | 0.01435 | 0.00947 |
| PSNR | | | | | | | | | | | |
| Srivastava | 20.809 | 19.916 | 19.177 | 18.601 | 18.103 | 17.681 | 17.276 | 16.960 | 16.671 | 16.421 | 18.183 |
| Mathieu | 17.921 | 16.530 | 15.805 | 15.358 | 15.061 | 14.850 | 14.664 | 14.539 | 14.461 | 14.421 | 15.361 |
| Lotter | 15.684 | 13.711 | 13.828 | 13.831 | 13.843 | 13.857 | 13.853 | 13.855 | 13.855 | 13.855 | 14.017 |
| Finn | 22.812 | 22.958 | 23.011 | 23.002 | 22.963 | 22.980 | 22.965 | 22.976 | 22.961 | 22.954 | **22.986** |
| Villegas | 13.860 | 13.859 | 13.860 | 13.858 | 13.856 | 13.858 | 13.856 | 13.855 | 13.855 | 13.856 | 13.857 |
| Oliu | 24.208 | 23.287 | 22.566 | 21.983 | 21.455 | 20.949 | 20.471 | 20.060 | 19.634 | 19.242 | 21.386 |
| DSSIM | | | | | | | | | | | |
| Srivastava | 0.05095 | 0.05916 | 0.06735 | 0.07426 | 0.08072 | 0.08661 | 0.09239 | 0.09707 | 0.10150 | 0.10544 | 0.08164 |
| Mathieu | 0.13374 | 0.18872 | 0.23448 | 0.29395 | 0.33991 | 0.39162 | 0.41217 | 0.42259 | 0.42772 | 0.43275 | 0.32770 |
| Lotter | 0.11971 | 0.16172 | 0.15431 | 0.14562 | 0.14292 | 0.13912 | 0.13945 | 0.13909 | 0.13920 | 0.13899 | 0.14201 |
| Finn | 0.05083 | 0.04581 | 0.04613 | 0.04626 | 0.04735 | 0.04863 | 0.05007 | 0.05234 | 0.05397 | 0.05647 | 0.04985 |
| Villegas | 0.13905 | 0.13885 | 0.13891 | 0.13894 | 0.13889 | 0.13886 | 0.13898 | 0.13908 | 0.13910 | 0.13899 | 0.13896 |
| Oliu | 0.02375 | 0.02854 | 0.03336 | 0.03762 | 0.04180 | 0.04612 | 0.05047 | 0.05444 | 0.05871 | 0.06275 | **0.04376** |

**Table A.1:** Quantitative results on the Moving MNIST dataset, relative to the number of time steps since the last input frame and on average over all time steps.

| | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 | avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MSE | | | | | | | | | | | |
| Srivastava | 0.00839 | 0.00852 | 0.00893 | 0.00940 | 0.00983 | 0.01024 | 0.01061 | 0.01093 | 0.01121 | 0.01145 | 0.00995 |
| Mathieu | 0.00085 | 0.00117 | 0.00144 | 0.00170 | 0.00189 | 0.00213 | 0.00230 | 0.00252 | 0.00269 | 0.00274 | 0.00194 |
| Lotter | 0.00144 | 0.00447 | 0.00361 | 0.00673 | 0.00580 | 0.00907 | 0.00856 | 0.01218 | 0.01237 | 0.01645 | 0.00807 |
| Finn | 0.00035 | 0.00047 | 0.00055 | 0.00061 | 0.00066 | 0.00071 | 0.00074 | 0.00078 | 0.00081 | 0.00083 | **0.00065** |
| Villegas | 0.00030 | 0.00063 | 0.00098 | 0.00132 | 0.00161 | 0.00189 | 0.00214 | 0.00234 | 0.00254 | 0.00274 | 0.00165 |
| Oliu | 0.00074 | 0.00097 | 0.00122 | 0.00147 | 0.00170 | 0.00190 | 0.00210 | 0.00228 | 0.00246 | 0.00262 | 0.00175 |
| PSNR | | | | | | | | | | | |
| Srivastava | 21.974 | 21.922 | 21.691 | 21.439 | 21.234 | 21.048 | 20.892 | 20.762 | 20.653 | 20.559 | 21.220 |
| Mathieu | 31.829 | 30.763 | 29.972 | 29.453 | 29.189 | 28.624 | 28.232 | 27.808 | 27.539 | 27.564 | 29.097 |
| Lotter | 31.360 | 26.974 | 27.532 | 24.952 | 25.300 | 23.329 | 23.340 | 21.776 | 21.508 | 20.280 | 24.635 |
| Finn | 37.634 | 36.681 | 36.290 | 36.071 | 35.922 | 35.810 | 35.724 | 35.658 | 35.607 | 35.565 | **36.101** |
| Villegas | 37.575 | 34.621 | 32.709 | 31.430 | 30.401 | 29.575 | 28.940 | 28.509 | 28.061 | 27.640 | 30.946 |
| Oliu | 32.044 | 31.106 | 30.320 | 29.683 | 29.165 | 28.749 | 28.400 | 28.097 | 27.829 | 27.596 | 29.299 |
| DSSIM | | | | | | | | | | | |
| Srivastava | 0.18878 | 0.18911 | 0.19203 | 0.19530 | 0.19809 | 0.20076 | 0.20307 | 0.20497 | 0.20655 | 0.20788 | 0.19868 |
| Mathieu | 0.06334 | 0.07920 | 0.08591 | 0.09606 | 0.09819 | 0.10693 | 0.10700 | 0.12005 | 0.12081 | 0.12467 | 0.10018 |
| Lotter | 0.04704 | 0.09218 | 0.08831 | 0.12188 | 0.12168 | 0.15040 | 0.15538 | 0.18037 | 0.19019 | 0.21140 | 0.13588 |
| Finn | 0.01850 | 0.02283 | 0.02552 | 0.02738 | 0.02874 | 0.02979 | 0.03064 | 0.03131 | 0.03188 | 0.03235 | **0.02790** |
| Villegas | 0.01778 | 0.03261 | 0.04741 | 0.06162 | 0.07656 | 0.09009 | 0.09973 | 0.10550 | 0.11346 | 0.12094 | 0.07657 |
| Oliu | 0.04057 | 0.05004 | 0.05858 | 0.06605 | 0.07262 | 0.07830 | 0.08335 | 0.08787 | 0.09200 | 0.09571 | 0.07251 |

**Table A.2:** Quantitative results on the KTH dataset, relative to the number of time steps since the last input frame and on average over all time steps.

| | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 | avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MSE | | | | | | | | | | | |
| Srivastava | 0.00908 | 0.05399 | 0.11943 | 0.16735 | 0.18014 | 0.17885 | 0.18194 | 0.19184 | 0.19989 | 0.20404 | 0.14866 |
| Mathieu | 0.00694 | 0.01004 | 0.01080 | 0.01186 | 0.01264 | 0.01351 | 0.01442 | 0.01472 | 0.01609 | 0.01762 | 0.01287 |
| Lotter | 0.00274 | 0.00878 | 0.00874 | 0.01523 | 0.01589 | 0.02313 | 0.02436 | 0.03307 | 0.03527 | 0.04516 | 0.02124 |
| Finn | 0.00391 | 0.00601 | 0.00748 | 0.00861 | 0.00955 | 0.01032 | 0.01099 | 0.01161 | 0.01215 | 0.01263 | 0.00940 |
| Villegas | 0.00268 | 0.00482 | 0.00655 | 0.00812 | 0.00940 | 0.01040 | 0.01150 | 0.01261 | 0.01373 | 0.01443 | 0.00942 |
| Oliu | 0.00274 | 0.00481 | 0.00652 | 0.00795 | 0.00920 | 0.01029 | 0.01122 | 0.01204 | 0.01273 | 0.01334 | **0.00908** |
| PSNR | | | | | | | | | | | |
| Srivastava | 21.077 | 13.125 | 9.876 | 8.515 | 8.212 | 8.212 | 8.094 | 7.849 | 7.670 | 7.577 | 10.021 |
| Mathieu | 22.539 | 21.133 | 21.140 | 20.840 | 20.658 | 20.421 | 20.069 | 19.910 | 19.362 | 18.846 | 20.492 |
| Lotter | 29.537 | 23.693 | 23.612 | 20.923 | 20.554 | 18.746 | 18.403 | 16.859 | 16.477 | 15.180 | 20.398 |
| Finn | 27.945 | 25.541 | 24.425 | 23.729 | 23.176 | 22.806 | 22.468 | 22.163 | 21.926 | 21.730 | 23.601 |
| Villegas | 29.389 | 26.389 | 24.759 | 23.765 | 22.959 | 22.440 | 21.854 | 21.401 | 20.940 | 20.671 | 23.457 |
| Oliu | 28.942 | 26.411 | 25.011 | 24.086 | 23.402 | 22.878 | 22.453 | 22.111 | 21.830 | 21.593 | **23.872** |
| DSSIM | | | | | | | | | | | |
| Srivastava | 0.13221 | 0.34915 | 0.41874 | 0.46370 | 0.47410 | 0.47660 | 0.47985 | 0.48424 | 0.48746 | 0.48940 | 0.42555 |
| Mathieu | 0.15196 | 0.17420 | 0.18437 | 0.19734 | 0.20279 | 0.20826 | 0.21891 | 0.22985 | 0.24673 | 0.25862 | 0.20730 |
| Lotter | 0.05345 | 0.12480 | 0.12507 | 0.17436 | 0.17825 | 0.21586 | 0.21961 | 0.25540 | 0.26117 | 0.29331 | 0.19013 |
| Finn | 0.07961 | 0.11293 | 0.13254 | 0.14593 | 0.15718 | 0.16498 | 0.17189 | 0.17852 | 0.18363 | 0.18813 | 0.15196 |
| Villegas | 0.05502 | 0.09051 | 0.11414 | 0.13304 | 0.14620 | 0.15725 | 0.16712 | 0.17599 | 0.18492 | 0.19083 | 0.14150 |
| Oliu | 0.05446 | 0.08526 | 0.10694 | 0.12300 | 0.13569 | 0.14580 | 0.15421 | 0.16115 | 0.16700 | 0.17195 | **0.13055** |

**Table A.3:** Quantitative results on the UCF-101 dataset, relative to the number of time steps since the last input frame and on average over all time steps.

## A.2 Qualitative Examples

Secondly, some example of predicted sequences by the tested models. These results are better seen and compared on a computer zooming in.

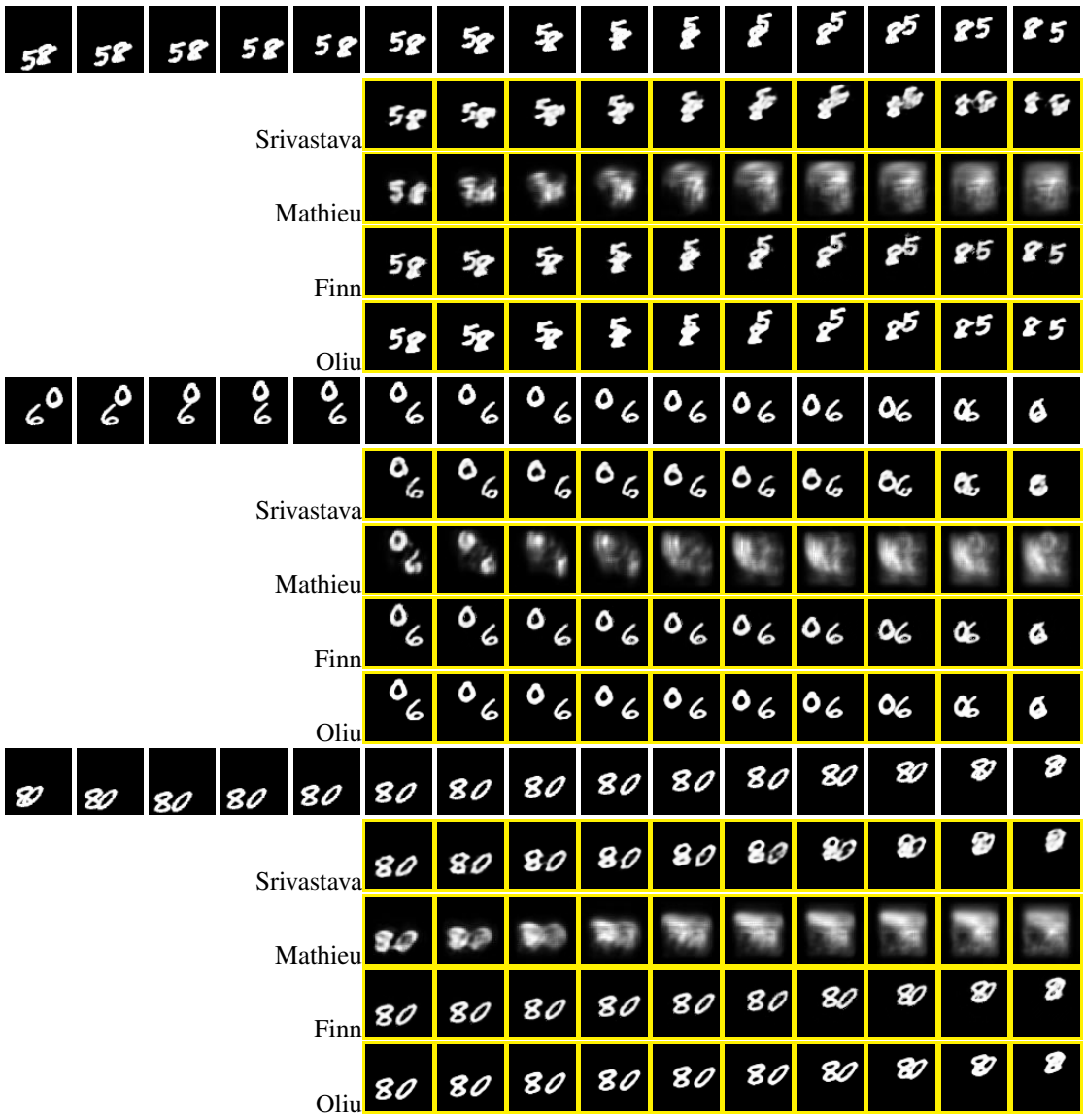


**Figure A.1:** MMNIST predictions for the tested models. First row per sequence shows last 5 inputs and target frames. Yellow means prediction.

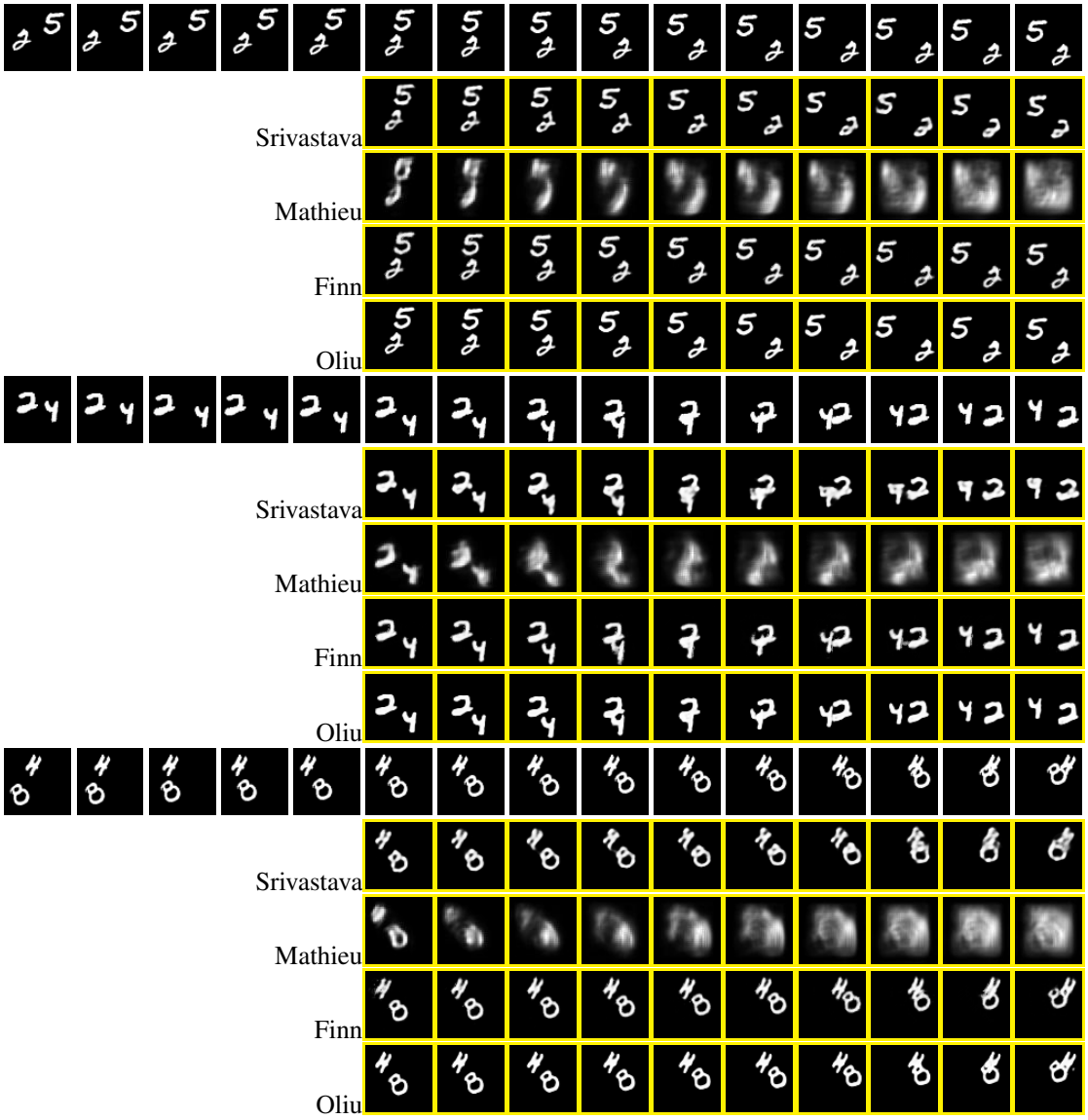


**Figure A.2:** MMNIST predictions for the tested models. First row per sequence shows last 5 inputs and target frames. Yellow means prediction.

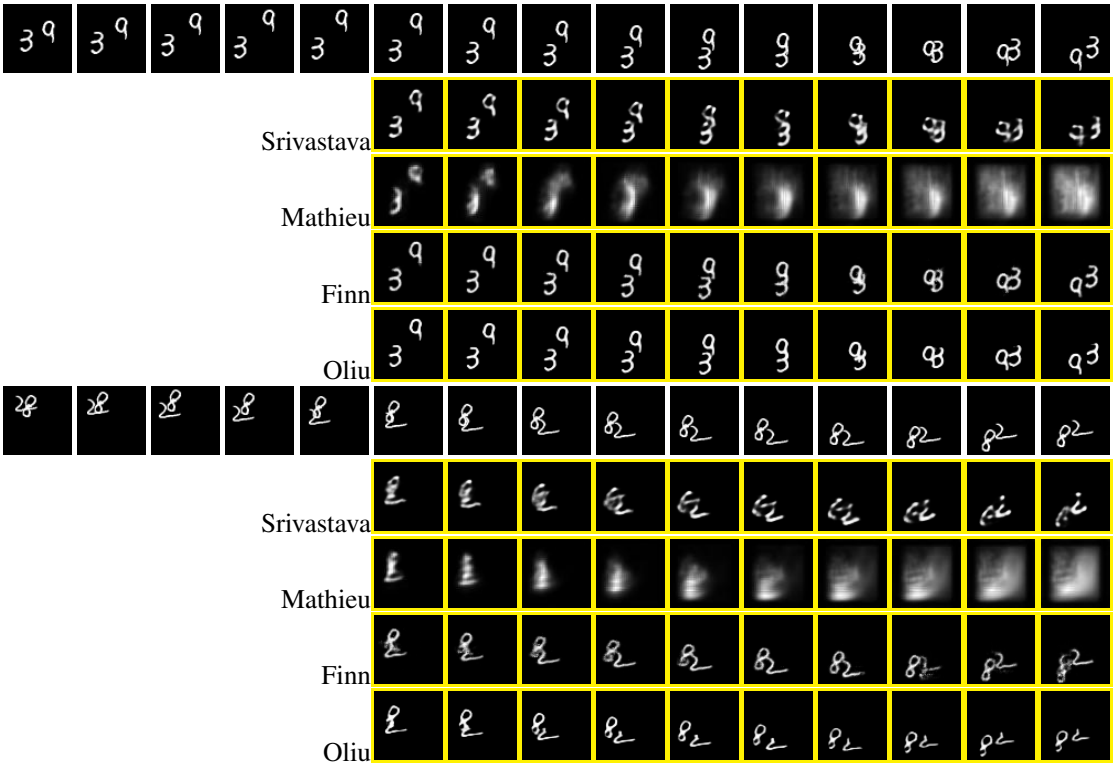


**Figure A.3:** MMNIST predictions for the tested models. First row per sequence shows last 5 inputs and target frames. Yellow means prediction.

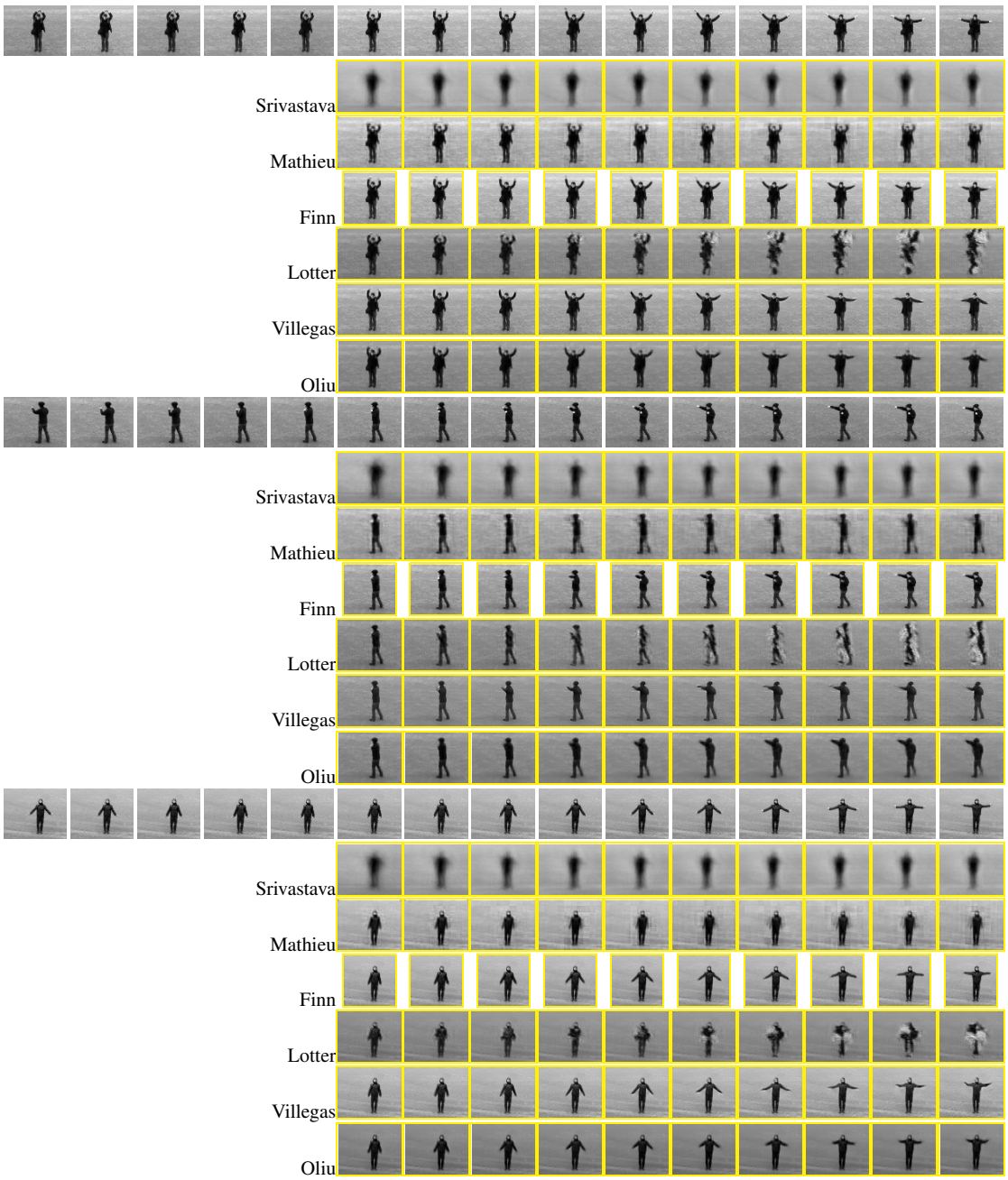


**Figure A.4:** KTH predictions for the tested models. First row for each sequence shows last 5 inputs and target frames. Yellow frames are model predictions.


**Figure A.5:** KTH predictions for the tested models. First row for each sequence shows last 5 inputs and target frames. Yellow frames are model predictions.

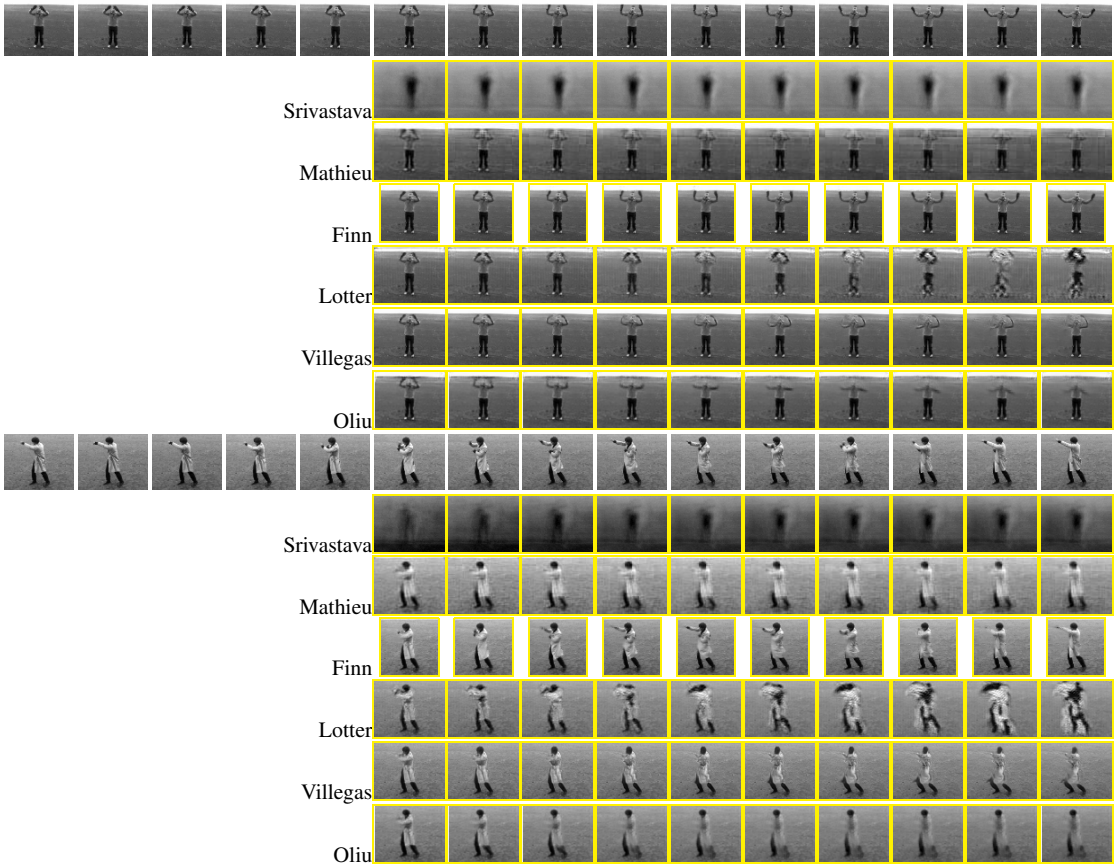


**Figure A.6:** KTH predictions for the tested models. First row for each sequence shows last 5 inputs and target frames. Yellow frames are model predictions.

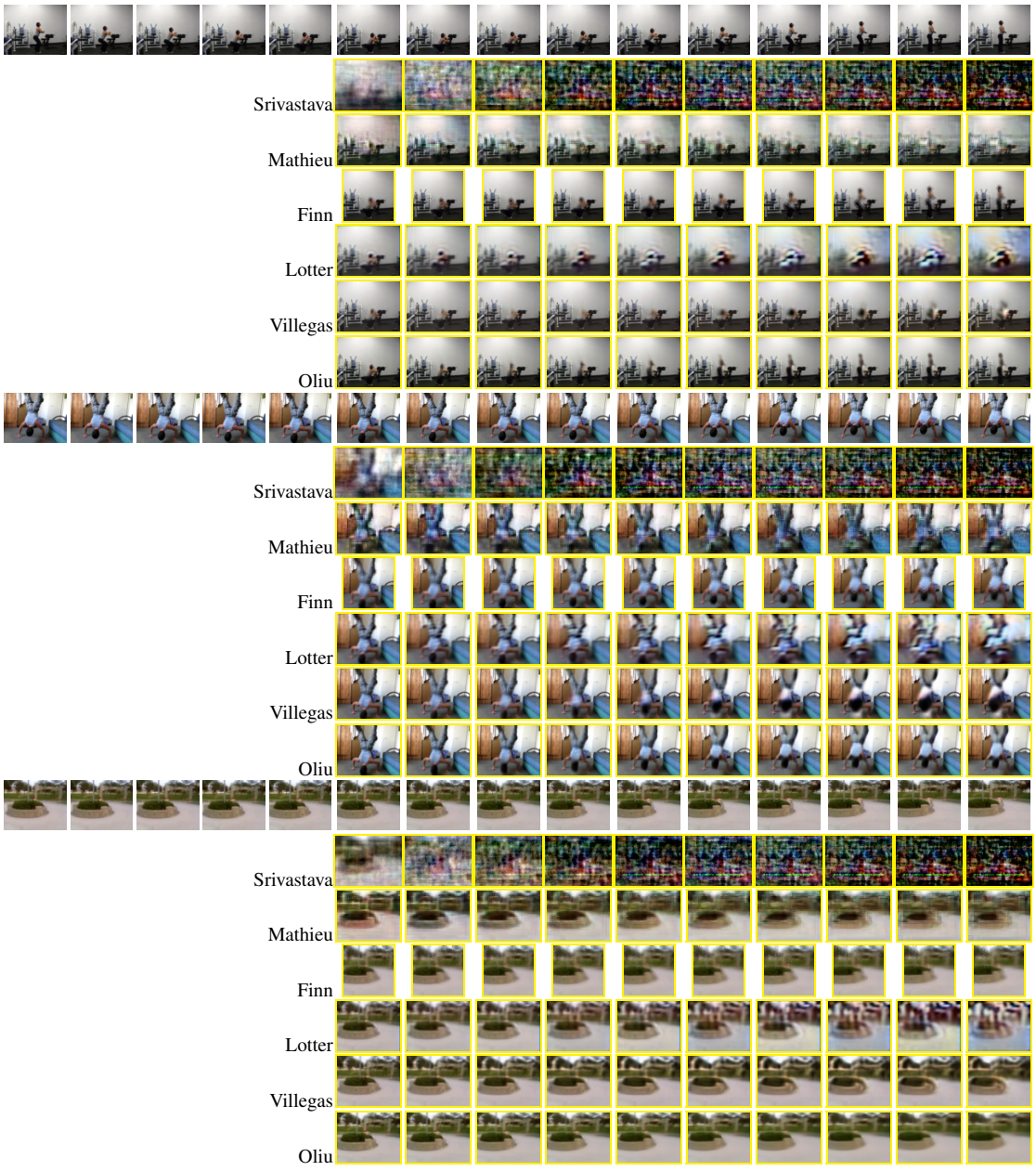


**Figure A.7:** UCF101 predictions for the tested models. First row for each sequence shows last 5 inputs and target frames. Yellow frames are model predictions.

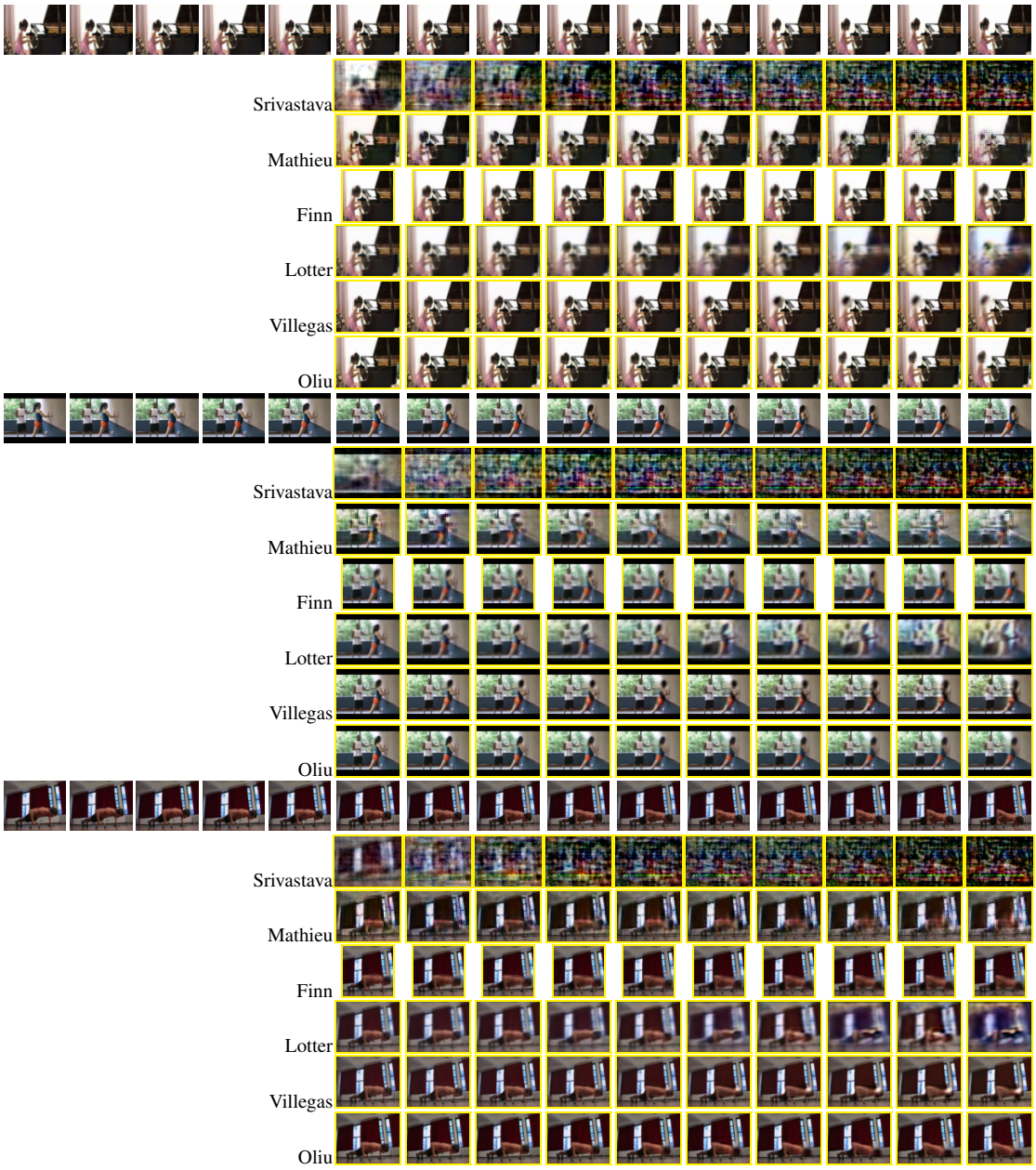


**Figure A.8:** UCF101 predictions for the tested models. First row for each sequence shows last 5 inputs and target frames. Yellow frames are model predictions.

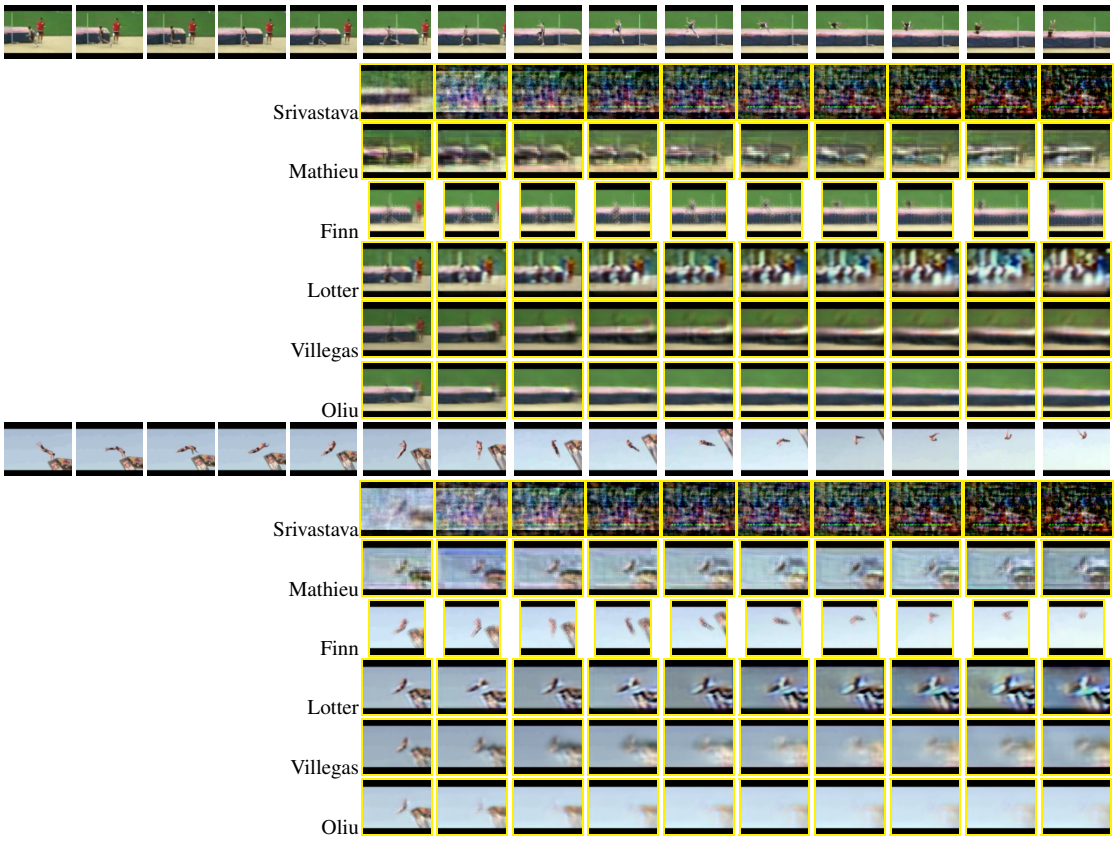


**Figure A.9:** UCF101 predictions for the tested models. First row for each sequence shows last 5 inputs and target frames. Yellow frames are model predictions.

# Apendix B

The present work is focused on analysing the SOTA for video prediction. Such analysis and the gathered experimental results obtained were used in the proposal by Oliu et al. in order to set a solid ground for comparison against other models. The results of that colaborative work were submited as a Paper to the Conference on Computer Vision and Pattern Recognition (CVPR) of 2018. It has been included as results of the present Master Thesis in the present appendix, starting in the next page.

Some of the figures were used in both the current project and the CVPR submission. Also, the author of the present document was not directly involved in the design of the model proposed in the submission.

# Folded Recurrent Neural Networks for Future Video Prediction

Marc Oliu
Universitat Oberta de Catalunya
Centre de Visió per Computador
Rambla del Poblenou, 156, 08018 Barcelona
moliusimon@uoc.edu

Javier Selva
Universitat de Barcelona
Gran Via de les Corts Catalanes, 585, 08007 Barcelona
javier.selva.castello@est.fib.upc.edu

Sergio Escalera
Universitat de Barcelona
Centre de Visió per Computador
Gran Via de les Corts Catalanes, 585, 08007 Barcelona
sergioescalera@cvc.uab.edu

## Abstract

*Future video prediction is an ill-posed Computer Vision problem that recently received much attention. Its main challenges are the high variability in video content, the propagation of errors through time, and the non-specificity of the future frames: given a sequence of past frames there is a continuous distribution of possible futures. This work introduces bijective Gated Recurrent Units, a double mapping between the input and output of a GRU layer. This allows for recurrent auto-encoders with state sharing between encoder and decoder, stratifying the sequence representation and helping to prevent capacity problems. We show how with this topology only the encoder or decoder needs to be applied for input encoding and prediction, respectively. This reduces the computational cost and avoids re-encoding the predictions when generating a sequence of frames, mitigating the propagation of errors. Furthermore, it is possible to remove layers from an already trained model, giving an insight to the role performed by each of layer and making the model more explainable. We evaluate our approach on three video datasets, outperforming state of the art prediction results on MMNIST and UCF101, and obtaining competitive results on KTH with 2 and 3 times less memory usage and computational cost than the best scored approach.*

## 1. Introduction

Future video prediction is a challenging task that recently received much attention due to its capabilities for learning in an unsupervised manner, making it possible to leverage large volumes of unlabelled data for video-related tasks such as action and gesture recognition, body pose recovery, and video segmentation. One of the main problems when approaching this task is the need of expensive models both in terms of memory and computational power in order to capture the variability present in video data. Another problem is the propagation of errors in recurrent models, which is tied to the inherent uncertainty of video prediction: given a series of previous frames, there are multiple feasible futures. This, left unchecked, results in a blurry prediction averaging the space of possible futures that propagates back into the network when predicting subsequent frames.

In this work we propose a new approach to recurrent auto-encoders (AE) with state sharing between encoder and decoder. We show how the exposed state in Gated Recurrent Units (GRU) can be used to create a bijective mapping between the input and output of each layer. Creating a stack of these layers allows for a bidirectional flow of information similar to an AE[1], but with many inherent advantages. This allows to reduce memory and computational costs during both training and testing: only the encoder or decoder needs to be executed for both input encoding and prediction,

[1] Code will be made publicly available after publication of the paper.

respectively. Furthermore, the representation is stratified so that only part of the input needs to be encoded at each layer: low level information that is not necessary to capture higher level dynamics is not passed to the next layer. It naturally provides an identity mapping of the input as starting point, facilitating the initial stages of training. The approach also mitigates the propagation of errors: while it does not solve the problem of blur, it prevents its magnification in subsequent predictions. Furthermore, an already trained network can be deconstructed in order to analyse the role of each layer in the final predictions, making the model more explainable. Since the encoder and decoder states are shared, the architecture can be thought of as a recurrent auto-encoder folded in half, with corresponding encoder and decoder layers overlapping. We call our method Folded Recurrent Neural Network (fRNN). In summary, our list of contributions is as follows:

- A new shared-state recurrent AE with low memory and computational costs.
- Mitigation of error propagation through time.
- It naturally provides an identity function during training.
- Model explainability through layer removal.
- Model optimisation through layer removal.
- Demonstration of representation stratification.

## 2. Related work

Much research has been done on future video prediction in recent years. In this section we review main works, organising them into non-recurrent and recurrent models.

**Non-recurrent approaches.** Most non-recurrent techniques focus on solving problems related to the probability distribution of possible futures or the introduction of a-priori information to guide the prediction into a more restricted output space. One of the first techniques tackling the problem of multiple possible futures was *Cross Convolutional Networks* (CCN) [18]. It predicts multiple possible future frames from a single image. The network generates a distribution of future motions based on the motion image between the last pair of frames and a multi-scale pyramid representing the last frame. A sampled frame motion is decoded into convolutional kernels and applied to the encoded pyramid of image frames to generate the future prediction.

Vondrick *et. al* [16] proposed a model based on convolutional adversarial networks to generate a video sequence from a given single frame. They use a two stream network to model background and moving elements separately in static camera videos. Amersfoort *et. al* [14] developed a model to predict movement instead of working at pixel level. They extracted affine transformations between overlapping patches in consecutive frames to be fed into a CNN. Applying the predicted new transformation to the last frame in the sequence allowed to generate the next frame.

Vukotić *et al.* [17] use a convolutional AE taking a single frame along with a time differential which is fed after encoding frame information. The goal is to predict a single future frame at a given time interval from the input one. *Deep Voxel Flow* (DVF) [5] uses a convolutional AE network with unpooling layers in the decoder and skip connections between corresponding layers of the encoder and decoder. It feeds triplets of consecutive frames as input, predicting the next frame as output. During training, it uses an L1 reconstruction loss with Total Variation regularisation to minimise target image blur and enforce coherence.

**Recurrent approaches.** Given the sequential nature of future video prediction, recurrent approaches form the bulk of the scientific literature in the topic. Most works focus on the prediction blur and network capacity problems. Ranzato *et al.* [10] was one of the first recurrent proposals for frame prediction. They realised common losses introduced blur to the predictions and stressed the need for specialised ones. They adapted a RNN model from NLP to predict video sequences. They quantized image patches into a dictionary and reduced the generation problem to predicting the index in the dictionary for each image region of the next frame.

Mathieu *et al.* [7] proposed an AE multi-scale architecture to predict future frames. They used an adversarial setting and proposed the Gradient Difference Loss to promote sharp predictions. Oh *et al.* [8] predict future frames on Atari games conditioning on the action taken by the player, reducing the space of possible futures. It uses a convolutional AE, where the representation is transformed by an action-specific transform before decoding. They make two proposals for the last layer of the encoder: a regular fully-connected layer and a recurrent one, obtaining better results with the recurrent model. Patraucean *et al.* [9] combinine a recurrent convolutional LSTM AE with optical flow. The approach predicts a flow map and uses it to wrap the previous frame. While this method is capable of learning long-range dependencies between frames, the use of optical flow to update the frame tends to introduce artefacts for large or complex motions. Srivastava *et al.* [13] also use a recurrent AE approach where an input sequence is encoded and its state is copied into the decoder. The decoder is then applied to generate a given number of frames. This model does not need to apply both the encoder and decoder at each time step, but is limited to a single recurrent layer for each part.

Brabandere & Jia *et al.* [4] developed the Dynamic Filter Network. This architecture uses a recurrent convolutional network to generate filters based on the input sequence. The filter is then applied to the last frame through another convolutional network in order to predict the next frame. Villegas *et al.* [15] propose a model treating motion and appearance separately. They use an autoencoder architecture where the encoder works separately for motion, using CNN + LSTM with difference images as input, and appearance, using a

CNN with the last frame of the sequence as input. The decoder receives a concatenation of the encoded representations and multi-scale residual connections coming from each branch. *PredNet* [6] is a recurrent residual network using convolutional LSTM layers. It is temporally recurrent like previous methods, but also contains multiple recurrent blocks at the frame level, each minimising the discrepancies from the previous block prediction.

*Convolutional Dynamic Neural Advection* (CDNA) introduced skip connections to recurrent convolutional AEs [2]. It uses the AE to generate a series of affine transform maps and compositing masks. Then, it generates multiple images with each affine transform map and fuses them using the compositing masks. The goal is to predict a sequence of future frames from a physical system, based on both the previous frames and the state of a robotic arm interacting with the scene. Similarly, *Video Ladder Networks* (VLN) [1] proposed a convolutional AE topology with skip connections between corresponding levels of the encoder and decoder. Pairs of convolutions are grouped into residual blocks. The information is passed horizontally between corresponding blocks, both by directly passing the output to the decoder block and by using a recurrent bridge layer. This topology is further extended with *Recurrent Ladder Networks* (RLN) [3], where the recurrent bridge connections are removed, and the residual blocks replaced by recurrent layers. This results in recurrent layers taking three inputs: cell state, output of the previous layer, and output of the corresponding layer from the previous encoding/decoding step.

Both VLN and RLN share some similarities with our approach: they both propose a recurrent AE with bridge connections between encoder and decoder. This allows for representation stratification, reducing the capacity needs of the subsequent layers. However, using skip connections instead of state sharing has some disadvantages: higher number of parameters and memory requirements, impossibility to skip the encoding/decoding steps (resulting in a higher computational cost) and reduced explainability due to not allowing layers to be removed after training. Finally, bridge connections do not provide an initial identity function during training. This makes it hard for the model to converge in some cases: when the background is homogeneous the model may not learn a proper initial mapping between input and output, but set the weights to zero and adjust the bias of the last layer, eliminating the gradient in the process.

# 3. Proposed method

We propose an architecture based on recurrent convolutional AEs to deal with network capacity and error propagation problems on the future video prediction task. It consists on a series of bijective GRUs that allow for a bidirectional flow of information between states by providing an extra set of gates. These are then stacked, forming an encoder and decoder using the forward and backward functions of the bijective GRUs (Figures 1, 2). We call it Folded Recurrent Neural Network (fRNN). Because of the state sharing between encoder and decoder, this topology has several advantages: stratification of the encoded information, lower memory and computational requirements compared to regular recurrent AEs, constrained propagation of errors, and increased explainability through layer removal.

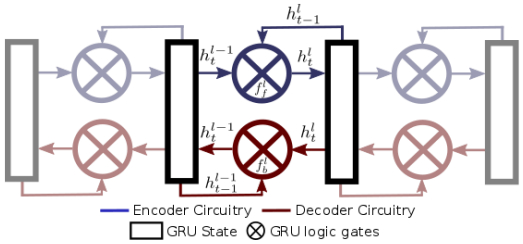


Figure 1. Scheme of a Bijective Gated Recurrent Unit. Blue and red correspond to forward and backward gates, respectively. Rectangles represent the recurrent state cell.

## 3.1. Bijective Gated Recurrent Units

GRUs have their state fully exposed to the output layer. This allows us to define a bidirectional mapping between input and output by replicating the logic gates of the GRU layer. Lets define the output of a GRU at layer $l$ and time step $t$ as $h^l_t = f^l_f(h^{l-1}_t, h^l_{t-1})$ given an input $h^{l-1}_t$ and its state at the previous time step $h^l_{t-1}$. A second set of weights can be used to define an inverse mapping $h^{l-1}_{t-1} = f^l_b(h^l_t, h^{l-1}_t)$ using the output of the forward function at the current time step to update its input, which is treated as the hidden state of the inverse function. This is illustrated in Figure 1. We will refer to this double mapping as bijective GRU (bGRU).

## 3.2. Folded Recurrent Neural Network

By stacking multiple bGRUs, a recurrent AE is obtained. Given $n$ bGRUs, the encoder is defined by the set of forward functions $E = \{f^1_f, \ ..., \ f^n_f\}$ and the decoder by the set of backward functions $D = \{f^n_b, \ ..., \ f^1_b\}$. This is illustrated in Figure 2, and is equivalent to a recurrent AE with shared states, with 3 main advantages: 1) It is not necessary to feed the predictions back into the network in order to generate the following predictions. Because the states are shared, the decoder already updates all the states except for the bridge state between encoder and decoder. The bridge state is updated by applying the last layer of the encoder before generating the next prediction. The shadowed area in Figure 2 shows the section of the computational graph that is not required when performing multiple sequential predictions. For the same reason, when considering multiple sequential elements before prediction, only the encoder is required; 2) Because the network updates its states from the higher level representations to the lowest ones during prediction, errors introduced at a given layer during gen-

eration are not propagated back into deeper layers, leaving the higher-level dynamics unaffected; 3) The model implicitly provides a noisy identity model during training, as it is shown in Figure 8, when all bGRU layers are removed. The input state of the first bGRU layer is either the input image itself or, when first applying a series of convolutional layers, an over-complete representation of the input. A noise signal is introduced to the representation by the backward function of the untrained first bGRU layer. This provides the model with an initial identity model. As we show in Section 4.3, this helps the model to converge in some datasets like MMNIST: when the same background is shared across instances, it prevents the model from killing the gradients by adjusting the biases to match the background and setting the weights to zero.

This approach shares some similarities with VLN and RLN. As with them, part of the information can be passed directly between corresponding layers of the encoder and decoder, not having to encode a full representation of the input into the deepest layer. However, our model implicitly passes the information through the shared recurrent states, making bridge connections unnecessary. When compared against an equivalent recurrent AE with bridge connections, this results in a much lower computational and memory cost. More specifically, the number of weights in a pair of forward and backward functions is equal to $3(\overline{h^{l-1}}^2 + \overline{h^l}^2 + 2\overline{h^{l-1}}\ \overline{h^l})$ in the case of bGRU, where $\overline{h^l}$ corresponds to the state size of layer $l$. When establishing bridge connections as in the case of VLN and RLN, that value is increased to $3(\overline{h^{l-1}}^2 + \overline{h^l}^2 + 4\overline{h^{l-1}}\ \overline{h^l})$. This corresponds to an increase of 44% in the number of parameters when one state has double the size of the other, and of 50% when they have the same size. Furthermore, both the encoder and decoder must be applied at each time step. Overall, the memory usage is doubled and the computational cost increased by a factor of between 2.88 and 3.

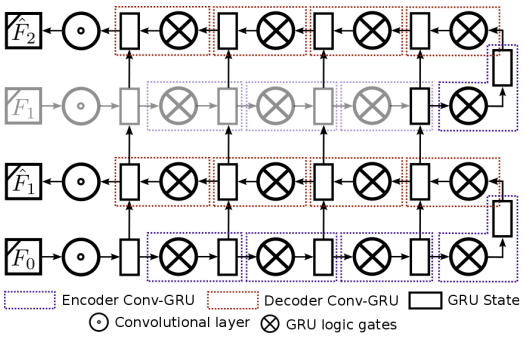


Figure 2. Folded Recurrent Neural Network topology. The recurrent states of the encoder and decoder are shared, resulting in a bidirectional mapping between states.

### 3.3. Training Folded RNNs

In a regular recurrent AE, a ground truth frame is introduced at each time step by applying both encoder and decoder. The output is used as a supervision point, comparing it to the next ground truth frame in the sequence. This implies all predictions are at a single time step from the last ground truth prediction. Here, we propose a training approach for Folded RNNs that exploits the ability of the topology of skipping the model encoder or decoder at a given time step. First $g$ ground truth frames are shown to the network by passing them through the encoder. The decoder is then applied $p$ times, producing $p$ predictions. This approach results in only half the memory requirements: either encoder or decoder is applied at each step, never both. This has the same advantage as the approach by Srivastava [13], where recurrently applying the decoder without further ground truth inputs encourages the network to learn video dynamics. This also prevents the network from learning an identity model, that is, copying the last input to the output.

## 4. Experiments

### 4.1. Data and evaluation protocol

We considered 3 datasets of different complexity in order to analyse the performance of the proposed method: Moving MNIST (MMNIST)[13], KTH [11], and UCF101 [12].

MMNIST consists of $64 \times 64$ greyscale sequences of length 20 displaying pairs of digits moving around the image. The sequences are generated by randomly sampling pairs of digits and trajectories. It contains a fixed test partition with 10000 sequences. We generated a million extra samples for training. KTH consists of 600 videos of 15-20 seconds from 25 subjects performing 6 actions in 4 different settings. The videos are all grayscale, at a resolution of $120 \times 160$ pixels and 25 fps. The dataset has been split into subjects 1 to 16 for training, and 17 to 25 for testing, resulting in 383 and 216 sequences, respectively. The frame size is reduced to $64 \times 80$ using bilinear interpolation, removing 5 pixels from the left and right borders before resising. UCF101 displays 101 actions, such as playing instruments, weight lifting or playing a wide range of sports. It is the most challenging dataset considered, showing a high intra-class variability. It contains 9950 training sequences and 3361 test sequences. Sequences are RGB at a resolution of $320 \times 240$ pixels and 25 fps. To increase motion between consecutive frames, one of every two frames was removed. As with KTH, the frame size is reduced to $64 \times 85$.

All methods are tested using 10 input frames to generate the following 10 frames. We use 3 common metrics for video prediction analysis: Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Dissimilarity (DSSIM). MSE and PSNR are objective measure-

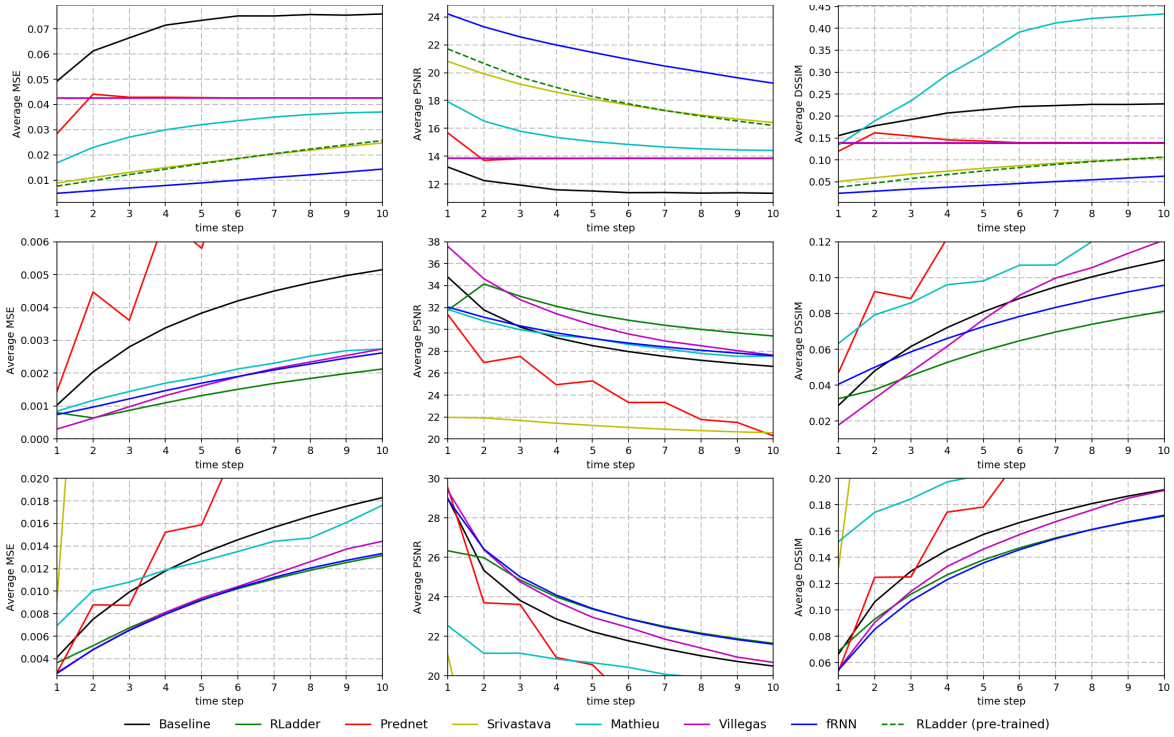


Figure 3. Quantitative results on the considered datasets in terms to the number of time steps since the last input frame. From top to bottom: MMNIST, KTH, and UCF101. From left to right: MSE, PSNR, and DSSIM.

| | MMNIST | | | KTH | | | UCF101 | | |
|---|---|---|---|---|---|---|---|---|---|
| | MSE | PSNR | DSSIM | MSE | PSNR | DSSIM | MSE | PSNR | DSSIM |
| Baseline | 0.06989 | 11.745 | 0.20718 | 0.00366 | 29.071 | 0.07900 | 0.01294 | 22.859 | 0.15043 |
| RLadder | 0.04254 | 13.857 | 0.13788 | **0.00139** | **31.268** | **0.05945** | 0.00918 | 23.558 | 0.13395 |
| Prednet [6] | 0.04137 | 14.017 | 0.14201 | 0.00807 | 24.635 | 0.13588 | 0.02124 | 20.398 | 0.19013 |
| Srivastava [13] | 0.01737 | 18.183 | 0.08164 | 0.00995 | 21.220 | 0.19860 | 0.14866 | 10.021 | 0.42555 |
| Mathieu [7] | 0.03071 | 15.361 | 0.32770 | 0.00194 | 29.097 | 0.10018 | 0.01287 | 20.492 | 0.20730 |
| Villegas [15] | 0.04254 | 13.857 | 0.13896 | 0.00165 | 30.946 | 0.07657 | 0.00940 | 23.457 | 0.14150 |
| fRNN | **0.00947** | **21.386** | **0.04376** | 0.00175 | 29.299 | 0.07251 | **0.00908** | **23.872** | **0.13055** |

Table 1. Average results over 10 time steps.

ments of reconstruction quality. DSSIM is a measure of the perceived quality. For DSSIM we use a Gaussian sliding window of size $11 \times 11$ and $\sigma = 1.5$.

### 4.2. Methods

To train the proposed method we used RMSProp with learning rate of 0.0001 and batch size of 12, selecting a random sub-sequence at each epoch. Initial weights were set using orthogonal initialisation, with biases set to 0. For testing, all possible sub-sequences of length 20 were considered. Our network topology consists of two convolutional layers followed by 8 bijective GRU layers, applying a $2 \times 2$ max pooling every 2 layers. We use deconvolution and nearest neighbours interpolation to invert the convolutional and max pooling layers, respectively. We train with L1 loss.

We compare against the following methods: Prednet [6], Srivastava *et al.* [13], Mathieu *et al.* [7] and Villegas *et al.* [15]. They are recurrent AEs, adapted for training on sequences of length 20. We used topologies and parameters defined by the authors. We also include a stub baseline model predicting the last input frame. We also design a second baseline (RLadder) to evaluate the advantages of using state sharing. RLadder has the same topology as the Folded RNN, but uses bridge connections between corresponding encoder and decoder layers. This is similar to how RLN [3] work, but using regular conv GRU layers in the decoder. Note that in order to keep the same state size on GRU layers, using bridge connections doubles the memory cost and almost triples the computational cost (Section 3.2).

### 4.3. Quantitative analysis

The first row of Figure 3 displays the results for the MMNIST dataset for the proposed method, baselines, and state of the art alternatives. The only methods to provide accurate predictions on this dataset are the fRNN models and the method by Srivastava *et al.*. Most other methods predict a black frame, with Mathieu *et al.* progressively blurring the digits. The problem with these methods is the loss of gra-

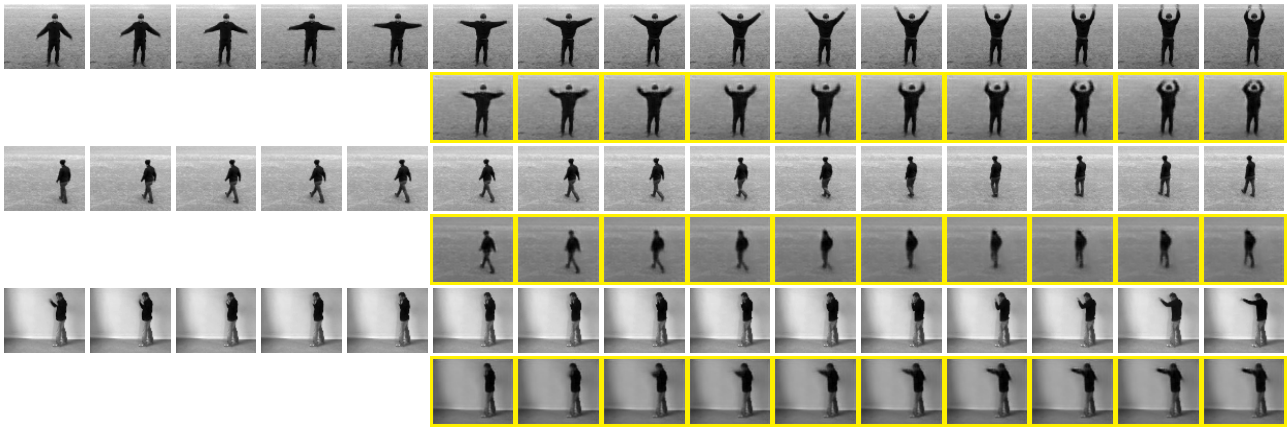

Figure 4. KTH fRNN predictions. First row for each sequence shows last 5 inputs and target frames. Yellow frames are model predictions.

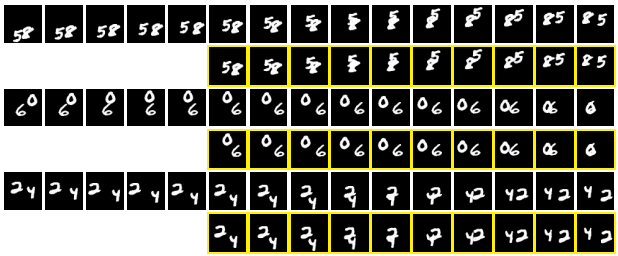

Figure 5. MMNIST fRNN predictions. First row per sequence shows last 5 inputs and target frames. Yellow means prediction.

dient during the first stages of training. On more complex datasets the methods start by learning an identity function, then refining the results. This is possible since in many sequences most of the frames remains unchanged. In the case of MMNIST, where the background is homogeneous, it is much easier for the models to set the weights of the output layer to zero and set the biases to match the background colour. This cuts the gradient and prevents further learning. Srivastava *et al.* use an auxiliary decoder to reconstruct the input frames, forcing the model to learn an identity function. This, as discussed at the end of Section 3.2, is naturally done by our method, that in turn gives an initial solution to improve on, preventing the models from learning a black image. In order to verify this effect, we pre-trained RLadder on the KTH dataset. While this dataset has completely different dynamics, the initial step to solve the problem remains: providing an identity function. Afterwards the model is fine-tuned on the MMNIST dataset. As it is shown in Figure 3 (dashed lines), this results in the model converging, with an accuracy comparable to Srivastava for the 3 evaluation metrics. With most methods not converging, fRNN performs best on all time steps and metrics, followed by Srivastava *et al.* and the pre-trained RLadder. Mean scores are shown in Table 1.

On the KTH dataset, as shown in Table 1, the best approach is our RLadder baseline followed by fRNN and Villegas *et al.* [15], both having similar results, but with Villegas *et al.* having slightly lower MSE and higher PSNR, and fRNN a lower DSSIM. While both approaches obtain comparable average results, the error increases faster over time in the case of Villegas *et al.* (second row in Figure 3).

In the case of the UCF101 dataset, as shown in Table 1, our fRNN approach is the best performing method for all 3 metrics. When looking at the third row of Figure 6, one can see that Villegas *et al.* starts out with results similar to fRNN on the first frame, but as in the case of KTH and MMNIST, the predictions degrade faster than with the proposed approach. Two methods display low performance in most cases. Prednet works well for the first predicted frame in the case of KTH and UCF101, but the error rapidly increases on the following predictions. This is due to a magnification of artefacts introduced on the first prediction, making the method unable to predict multiple frames without supervision. In the case of Srivastava *et al.* the problem is about capacity: it uses fully connected LSTM layers, making the number of parameters explode quickly with the state cell size. This severely limits the representation capacity for complex datasets such as KTH and UCF101.

Overall, for the considered methods, the proposed fRNN is the best performing on MMINST and UCF101, the later being the most complex of the 3 datasets. We achieved these results with a simple topology: apart from the proposed bGRU layers, we use conventional max pooling and layers with an L1 loss. There are no normalisation or regularisation mechanisms, specialised activation functions, complex topologies or image transform operators. In the case of MMNIST, fRNN shows the ability to find a good initial representation and converges to good predictions where most other methods fail. In the case of KTH, fRNN has an overall accuracy comparable to that of Villegas *et al.*, being more stable over time. It is only surpassed by the proposed RLadder baseline, a method equivalent to fRNN but with 2 and 3 times more memory and computational requirements.

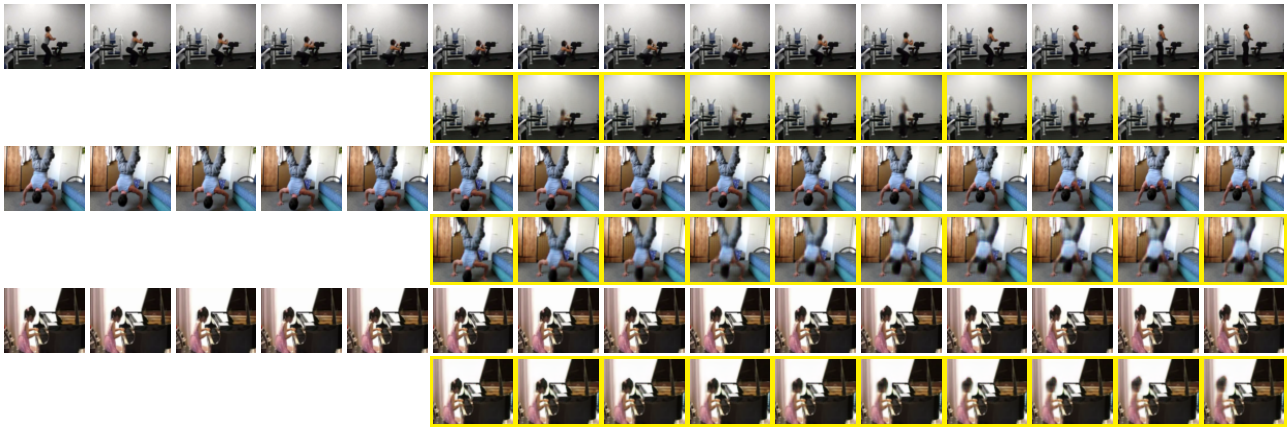

Figure 6. UCF fRNN predictions. First row for each sequence shows last 5 inputs and target frames. Yellow frames are model predictions.

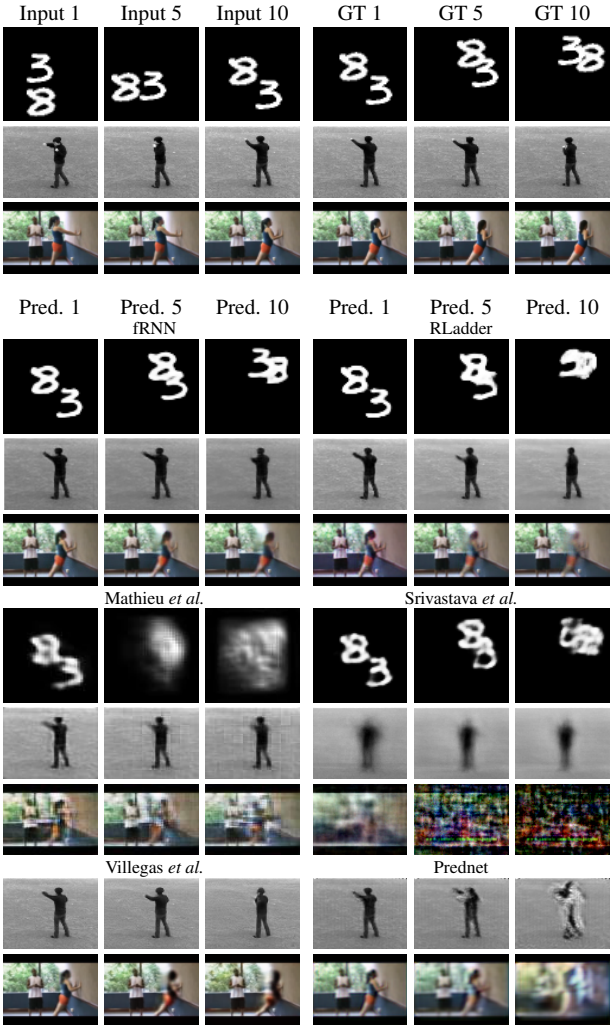


Figure 7. Predictions at 1, 5, and 10 time steps from the last ground truth frame. RLadder predictions on MMNIST are from the model pre-trained on KTH.

### 4.4. Qualitative analysis

In this section we evaluate our approach qualitatively on some samples from the three considered datasets.

Figure 5 shows the last 5 input frames from some MMNIST sequences along with the next 10 ground truth frames and their corresponding fRNN predictions. The ten predictions are generated sequentially without showing the previous ground truth/prediction to the network, that is, only using the decoder. As it can be seen, the digits maintain their sharpness across the sequence of predictions. Also, the bounces at the edges of the image are done correctly, and the digits do not distort or deform when crossing. This shows the network internally encodes the appearance of each digit, making it possible to reconstruct them after sharing the same region in the image plane.

Qualitative examples of fRNN predictions on the KTH dataset are shown in Figure 4. It shows three actions: hand waving, walking, and boxing. The blur stops increasing after the first three predictions, generating plausible motions for the corresponding actions while background artefacts are not introduced. Although the movement patterns for each type of action have a wide range of variability on its trajectory, bGRU gives relatively sharp predictions for the limbs. The first and third examples also show the ability of the model to recover from blur. The blur slightly increases for the arms while the action is performed, but decreases again as these reach the final position.

Figure 6 shows fRNN prediction examples from the UCF101 dataset. These correspond to two different physical exercises and a girl playing the piano. One common attribute for all images is the static parts do not lose sharpness over time, and when occluded and later revealed, the background is properly reconstructed. As for dynamic regions, the network correctly predicts actions with low variability, as shown in rows 1-2, where a repetitive movement is performed, and in last row, where the girl moves back to a correct body posture. Regarding the introduction of blur, these dynamic regions introduce it due to the uncertainty of the action, averaging the possible futures.

The first row also shows an interesting behaviour: while the woman is standing up, the upper body becomes blurry

due to uncertainty, but as the woman finishes her motion and ends up in the expected upright position, the frames sharpen again. Since the model does not propagate errors to deeper layers nor makes use of previous predictions for the following ones, the introduction of blur does not imply this blur will be propagated. In this example, while the middle motion could have multiple predictions depending on the movement pace and the inclination of the body while performing it, the final body pose has a lower uncertainty.

In Figure 7 we compare predictions from the proposed approach against the RLadder baseline and other state of the art methods. For the MMNIST dataset we did not consider Villegas *et al.* and Prednet since these methods fail to successfully converge and they predict a sequence of black frames. From the rest of approaches, fRNN obtains the best predictions, with little blur or distortion. The RLadder baseline is the second best approach. It does not introduce blur, but heavily deforms the digits after they cross. Srivastava *et al.* and Mathieu *et al.* both accumulate blur over time, but while the former does so to a smaller degree, the later makes the digits unrecognisable after five frames.

For KTH, Villegas *et al.* obtains outstanding qualitative results. It predicts plausible dynamics and maintains the sharpness of both the individual and background. Both fRNN and RLadder follow closely, predicting plausible dynamics, but not being as good as Villegas *et al.* at maintaining the sharpness of the individual.

On UCF101 the best prediction is obtained by our model, with little blur or distortion compared to the other methods. The second best is Villegas *et al.*, successfully capturing the movement patterns but introducing more blur and important distorsions on the last frame. When looking at the background, fRNN proposes a plausible initial estimation and progressively completes it as the woman moves. On the other hand, Villegas *et al.* modifies already generated regions as more background is uncovered, generating an unrealistic sequence regarding the background.

Srivastava and Prednet fail on both KTH and UCF101. Srivastava *et al.* heavily distort the frames. As discussed in Section 4.3, this is due to the use of fully connected recurrent layers, which constrains the state size and prevents the model from encoding relevant information on more complex scenarios. In the case of Prednet, it makes good predictions for the first frame, but rapidly accumulates artefacts.

### 4.5. Representation stratification analysis

In this section the stratification of the sequence representation among the bGRU layers is analysed. Because bGRU units allow for a bijective mapping between the states, it is possible to remove the bottom-most layers of an already trained network. This allows us to check how the predictions are affected, providing an insight on the dynamics captured by each layer. To the best of our knowledge, this is the first topology allowing for a direct observation of the behaviour encoded on each layer.

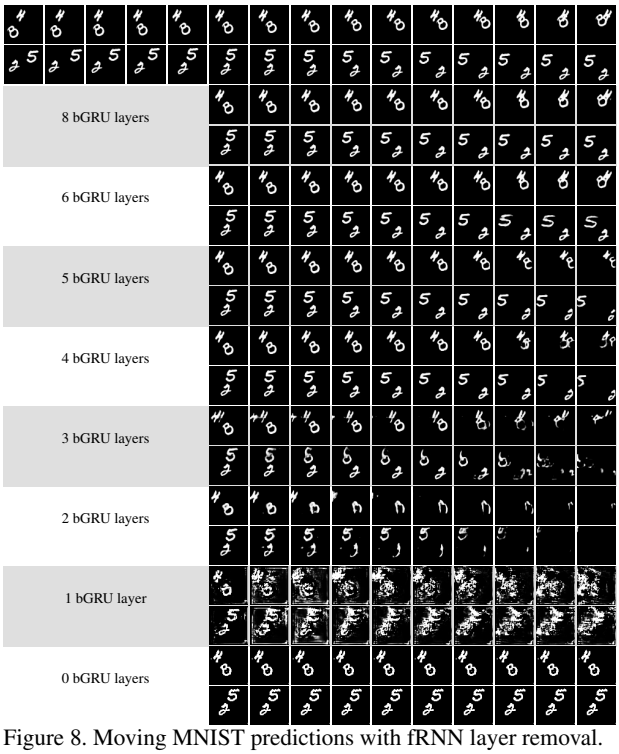

Figure 8. Moving MNIST predictions with fRNN layer removal.

Specifically, the same sequences are predicted multiple times, removing a layer each time. This is shown in Figure 8 for the MMNIST dataset. The analysed model consists of 10 layers: 2 convolutional layers and 8 bGRU layers. Firstly, removing the last 2 bGRU layers has no significant impact on prediction. This shows that, for this simple dataset, the network has a higher capacity than required. Further removing layers does not result in a loss of pixel-level information, but on a progressive loss of behaviours, from more complex to simpler ones. This means information at a given level of abstraction is not encoded into higher level layers. When removing the third deepest bGRU layer, the digits stop bouncing and keep their linear trajectories, exiting the image. This indicates this layer is in charge of encoding information on bouncing dynamics. When removing the next layer, digits stop behaving correctly on the boundaries of the image. Parts of the digit bounce while others keep the previous trajectory. While this also has to do with bouncing dynamics, the layer seems to be in charge of recognising digits as single units following the same movement pattern. When removed, different segments of the digit are allowed to move as separate elements. Finally, with only 3-2 bGRU layers the digits are distorted in various ways. With only two layers left, the general linear dynamics are still captured by the model. By leaving a single bGRU layer, the linear dynamics are lost.

According to these results, linear movement dynamics are captured at the pixel level on the first two bGRU layers. The next two start aggregating these movement patterns into single-trajectory components, preventing their distortion. Also the collision of these components with image bounds are detected. The fifth layer aggregates single-motion components into digits, forcing them to follow the same motion. This seems to have the effect of preventing bounces, likely due to only one of the components reaching the edge of the image. It is the sixth bGRU layer that provides a coherent bouncing pattern for the whole digit.

## 5. Conclusions

We presented Folded Recurrent Neural Networks, a new recurrent architecture for video prediction with lower computational and memory cost compared to equivalent recurrent AE models. This is achieved by using the proposed bijective GRUs, which horizontally pass information between the encoder and decoder. This eliminates the need for using the entire AE at any given step: only the encoder or decoder needs to be executed for both input encoding and prediction, respectively. It also facilitates the convergence by naturally providing a noisy identity function during training. We evaluated our approach on three video datasets, outperforming state of the art prediction results on MMNIST and UCF101, and obtaining competitive results on KTH with 2 and 3 times less memory usage and computational cost than the best scored approach. Qualitatively, the model can limit and recover from blur by preventing its propagation from low to high level dynamics. We also demonstrated stratification of the representation, topology optimisation, and model explainability through layer removal. Each layer has been shown to modify the state of the previous one by adding more complex behaviours: removing a layer eliminates its behaviours but leaves lower-level ones untouched.

## References


[1] F. Cricri, M. Honkala, X. Ni, E. Aksu, and M. Gabbouj. Video ladder networks. *arXiv preprint arXiv:1612.01756*, 2016. 3

[2] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems*, pages 64–72, 2016. 3

[3] A. Ilin, I. Prémont-Schwarz, T. H. Hao, A. Rasmus, R. Boney, and H. Valpola. Recurrent ladder networks. *arXiv preprint arXiv:1707.09219*, 2017. 3, 5

[4] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool. Dynamic filter networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 667–675. Curran Associates, Inc., 2016. 2

[5] Z. Liu, R. Yeh, X. Tang, Y. Liu, and A. Agarwala. Video frame synthesis using deep voxel flow. *arXiv preprint arXiv:1702.02463*, 2017. 2

[6] W. Lotter, G. Kreiman, and D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016. 3, 5

[7] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015. 2, 5

[8] 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*, pages 2863–2871, 2015. 2

[9] V. Patraucean, A. Handa, and R. Cipolla. Spatio-temporal video autoencoder with differentiable memory. *arXiv preprint arXiv:1511.06309*, 2015. 2

[10] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. Video (language) modeling: a baseline for generative models of natural videos. *CoRR*, abs/1412.6604, 2014. 2

[11] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: a local svm approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32–36. IEEE, 2004. 4

[12] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 4

[13] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning*, pages 843–852, 2015. 2, 4, 5

[14] J. R. van Amersfoort, A. Kannan, M. Ranzato, A. Szlam, D. Tran, and S. Chintala. Transformation-based models of video sequences. *CoRR*, abs/1701.08435, 2017. 2

[15] R. Villegas, J. Yang, S. Hong, X. Lin, and H. Lee. Decomposing motion and content for natural video sequence prediction. *ICLR*, 1(2):7, 2017. 2, 5, 6

[16] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 613–621. Curran Associates, Inc., 2016. 2

[17] V. Vukotić, S.-L. Pintea, C. Raymond, G. Gravier, and J. Van Gemert. One-step time-dependent future video frame prediction with a convolutional encoder-decoder neural network. *arXiv preprint arXiv:1702.04125*, 2017. 2

[18] 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*, pages 91–99, 2016. 2