Forecasting wave propagation with neural networks

Fotiadis Efstathios

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2019

Abstract

Wave propagation is formalised by partial differential equations. While numerical solvers exist their computational requirements are prohibiting for time-critical applications. In this work, we assess the role of neural networks as alternatives for predicting the evolution of such physical phenomena. We evaluate state-of-the-art recurrent architectures for spatio-temporal forecasting and show that they only perform well for a short horizon. Inspired by the deterministic nature of the system we propose a new inference method and demonstrate that it improves long-term predictions by a significant margin. Whether feed-forward networks can learn the dynamics of such a problem is also investigated. Our experiments demonstrate that purely convolutional networks perform at least as well as recurrent architectures. This could be seen as an indication that in physical systems like this, convolutions alone can exploit the inherent determinism and effectively learn the underlying dynamics without the need for memory modules. The generalisation capacity of the models is evaluated in various unseen datasets. Results suggest that models struggle most when the waves are faster or slower than the ones found in the training set. To increase robustnes we train with two different datasets. In the first, the wave speed is fixed and in the other, the model is trained to be reversible, i.e. predict the future as well as the past. Overall, this project proposes and evaluates methods that introduce a deterministic induction bias while it discusses their strengths and weaknesses.

Acknowledgements

Firstly, I would like to thank my supervisor Amos Storkey for his support as well as our fruitful discussions during this project. Next, I would like to express my thanks to Anil Bharath for bringing this interesting topic to my attention and Stef Garasto and Nick Antonopoulos for the feedback. Finally, I would also like to thank my family, friends and Ioanna for their support.

Table of Contents

Life can only be understood backwards;
but it must be lived forwards.

Søren Kierkegaard, Journals

Chapter 1

Introduction

1.1 Motivation

From the predictions of solar motion by prehistoric civilisations to the mathematicians of the Hellenistic world and from the groundbreaking work of Newton to modern science there is one connecting thread: the effort to understand the physical world. This long-standing endeavour is not only posed to quench an innate intellectual thirst, but it also has huge implications for our advancement as species in an ever-changing habitat. Understanding the world opens up the possibility to predict it and also shape it closer to a desired state.

Mathematics provides an elegant framework for abstracting and capturing the inner workings of physical systems. For example, partial differential equations (PDEs) are used to describe dynamical physical processes. Such processes are characterised by continuous state changes and are ubiquitous in science and engineering.

Fluid flow is a dynamical process governed by the Navier-Stokes equations which can be seen as Newton's second law of motion for fluids. Practical uses exist in diverse fields like aerodynamics, environmental engineering, biological flow estimation, computer animation, dissipation of pollutants and climate modelling [1]. Despite their importance, our understanding of the N-S equation is far from complete since there is no proof for the existence and smoothness of solutions in 3 dimensions. The Clay Mathematics Institute has named this one of the most important open mathematical challenges and offers a million-dollar prize for a solution [2].

Given that analytic solutions are so hard to find the most common approach is to approximate the solutions with numerical methods like Computational Fluid Dynamics (CFD). Depending on the complexity of the system, though, numerical simulations can

have high computational cost [3].

In system design, iterating quickly is very important and simulations can become a bottleneck, especially in the earlier stages of the process when the potential designs are too many. Speed is also relevant in biomedical applications like heart surgery, where it is crucial to minimise the time for designing patient-specific prosthetics to be implanted [4]. In cases like these, it is desirable to have faster tools even if this comes at the cost of some approximation accuracy.

Previous studies have shown that machine learning can be used to accelerate part of the fluid simulation process [5]. Deep learning (DL) techniques are hierarchical machine learning models where each layer captures increasingly more abstract representations of the previous layer. Over the recent years, DL has been successfully applied in many domains including object detection, video prediction, natural language processing and speech recognition [6]. With their potential to learn sophisticated representations, DL techniques provide a promising pathway to accurately forecasting physical systems without being computationally heavy. Furthermore, DL can be used in an end-to-end fashion from raw input to final result thus speeding up not only the simulation but also the visualisation part of traditional CFD tools. This report aims to broaden our understanding of the benefits and limitations of applying DL to approximate fluid flow and physical systems in general.

1.2 Problem statement

This project focuses on predicting fluid flow on shallow water where the horizontal scale is much higher than the the vertical scale [7]. The shallow water flow is described by the Saint-Venant equations, which are derived from the Navier-Stokes equations with integration over the depth. The non-conservative form is shown below:

$$\begin{aligned}
\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}\Big((H+h)u\Big) + \frac{\partial}{\partial y}\Big((H+h)v\Big) &= 0, \\
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - fv &= -g\frac{\partial h}{\partial x} - bu + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \\
\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + fu &= -g\frac{\partial h}{\partial y} - bv + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)
\end{aligned} \tag{1.1}$$

where h is the water height, H the reference height of the surface, u, v correspond to the horizontal velocity across the x and y axes respectively, g is the gravity acceleration, f is the Coriolis coefficient, b the viscosity coefficient and ν the kinematic viscosity.

The evolution of such a system is equivalent to wave propagation in water. Figure 1.1, shows an example of rendered waves on a medium with solid boundaries. Essentially this is a spatio-temporal forecasting problem and the main goal is to find a deep learning model that given a few consecutive frames would accurately predict future frames on a long-term horizon. The model should operate solely on raw visual input, without any information about the specifications of the physical system. Moreover, it should be able to generalise well on unseen settings that might be considerably different than the training data.

Selecting this problem was based on various aspects. Firstly, shallow water equations are used in a wide variety of transport phenomena like atmosphere and oceanic flows, flood modelling and ice sheet formation [7, 8, 9]. Furthermore, the equation can be used to model wave propagation that has applications in many fields including modelling cardiac arrhythmia [10]. Moreover, the problem is far from trivial as can be seen in Figure 1.1 where the complexity of the equations allows intricate patterns to emerge. Generalising in different boundary conditions (Figure 1.2) is even more challenging and solving this problem will shed more light to how DL can exploit determinism to approximate fluid flow and physical simulations in general.

(a) t=0 (b) t=20 (c) t=60

Figure 1.1: Wave at different time steps t

(a) Solid (b) Mirror

Figure 1.2: Boundary conditions

1.3 Organization

This report is organized as follows. Chapter 2 presents related work that has been done on spatio-temporal forecasting. Chapter 3 introduces some concepts that are important for the experimental design and the analysis of the results. Chapter 4 analyses the experiments conducted in this project while Chapter 5 summarises with our conclusions and proposals for future work.

Chapter 2

Related work

This chapter presents previous work that has been done in related spatio-temporal forecasting problems. We broadly consider three categories placed on an axis from generic to domain-specific: real-life video prediction techniques, forecast of physical systems and methods specialised for fluid flow estimation (Figure 2.1). All machine learning models have to make some assumptions of the systems they are trying to model [11] and more specialised architectures, due to higher induction bias, tend to perform better on a narrower set of problems but are usually less transferable.

2.1 Natural video prediction

Prediction of the future in a real-life video stream is quite challenging because the world entails a lot of variability and making strong assumptions regarding the underlying process can easily harm generalisation. Hence, most techniques in this category treat the environment as a stochastic process.

Convolutional Neural Networks CNNs are feed-forward models that have been successful in image-related tasks and have also been used for video prediction [12] . In [13] the authors use 2D CNN filters to learn the latent vectors of two consecutive frames and then extrapolate their difference to the future to generate a prediction. The same idea has also been explored with multi-scale 2D CNNs in [14] where an adversarial loss was further included to combat the blurriness in predictions that are usually associated with the Mean Square Error (MSE) loss. Using 3D convolutions allow to directly extract spatio-temporal correlations and eliminate further manipulations [15]. While recurrent models have traditionally be seen as superior to CNNs for sequence modelling, new studies have shown that dilated convolutions perform better than re-

current networks in various 1D tasks [16]. Inspired by this, in this project we evaluate a CNN based on 2D dilations.

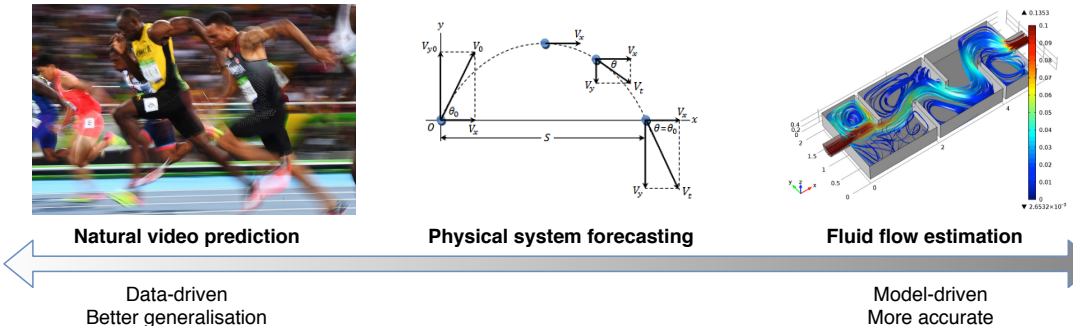


Figure 2.1: **The spectrum of techniques for spatio-temporal forecasting.** Generic, data-driven techniques can be applied to a wider range of problems but specialised model-driven architectures perform better in particular settings. The boundaries are not rigid.

Recurrent Models While CNNs are good at extracting info on the spatial domain, video prediction has a temporal element too. Recurrent networks like *Long-Short Term Memory* (LSTM) include an internal state that provides them with memory-like capabilities and have been used extensively in sequence modelling [17, 18]. These architectures are tailor-made for one-dimensional sequences and cannot be directly applied to spatio-temporal data. One way to apply recurrent neural networks to high-dimensional data is to first reduce the data vector form by some transform and then recurrently process the low-dimensional data [19, 20]. Recurrent approaches to video prediction usually follow the encoder-forecaster architecture where the encoder learns a latent representation of the input and the forecaster propagates it to the future (Figure 2.2 - right). In this project because all three recurrent models we evaluate are based on the encoder-forecaster framework. Srivastava et al. concatenated the input images in one vector and fed them to a stack of LSTMs to predict the next frame [19]. In LSTMs the number of network parameters grows exponentially with the input size, so concatenating the frames only works for smaller image sizes. To alleviate this problem various researchers have used a convolutional encoder to reduce the dimensionality [21, 22].

Combining the spatial capabilities of CNNs with the temporal modelling of LSTMs performs well but there are a couple of shortcomings. Firstly the spatial and temporal signals are processed in a linear fashion missing out on possible interactions. Secondly, even with the encoded input going through the LSTMs, the memory and com-

putations requirements of those models are significant. *Convolutional LSTM* (ConvLSTM) is a method to alleviate those problems by substituting the fully connected (FC) gates inside the LSTM with convolutional [23]. Whereas originally applied in predicting a stochastic physical process for short term meteorological phenomena, since then ConvLSTMs have been widely adopted in many state-of-the-art video prediction techniques [24, 25, 26, 27]. PredRNN introduced the *Causal LSTM*, a cell-based on ConvLSTM that adds a global memory variable that moves not only between states but also across layers [28]. PredRNN++ additionally adds a gateway unit that allows more efficient backpropagation through time (BPTT) [29]. While ConvLSTM and Causal LSTM are considered state-of-the-art for short-term prediction tasks, longer prediction horizons remain a challenge. Exploiting the good short-term predictions and the inherent determinism of wave propagation we propose a new inference method (Section 4.8).

Video Pixel Network follows a different approach and predicts the next frame one pixel at a time [30]. By using causal convolution layers and memory modules called Multiplicative Units, it learns a discrete joint distribution of the raw pixel values. VPN reaches state-of-the-art performance but misses some spatial correlations due to the 1D nature of its predictions on a 2D plane. PredCNN uses a similar approach but uses dilated causal convolutions to better capture long-distance spatial relationships [31].

Optical Flow Calculating the optical flow relies on the assumption that surfaces only undergo smooth changes from one frame to the next. Optical flow estimators have been part of many models for video prediction.

Disentangling the motion of objects from the background scene is another way to do forecasting [26]. FlowNet [32] is a seminal work that uses a *Convolutional-Deconvolutional Neural Network (CDNN)* to predict the optical flow between two frames. Many approaches to frame prediction include an optical flow module but are trained end-to-end. Convolutional Dynamic Neural Advection [24] learns affine convolutional transformations that are applied to each pixel of the input. The transformations are weighted by a mask that is calculated from the input frames by a different part of the network. Dynamic Filter Networks follow a similar approach but introduce and extra constraint allowing only one element of each filter to be non-zero [25]. Simplifying things even further, the authors of [33] extract optical flow and apply it to the input image through bi-linear sampling.

Generative models A major underlying assumption about natural video is stochasticity. Recent studies have proposed models that handle the inherent uncertainty by us-

ing generative models [13]. *Generative Adversarial Networks* (GANs) can implicitly represent a probability distribution of predictions [34]. Two CNN-based methods that have been discussed previously, use a combination of L2 loss with a GAN loss which leads to sharper predictions [14, 15]. Besides GANs, another method used to tackle uncertainty is the Variational Autoencoder (VAE) [35]. In VAEs instead of learning a fixed latent vector Z we learn a probability distribution of such vectors $p(z)$ which has been shown to alleviate the blurry prediction problem [36, 37, 38]

2.2 Forecasting physical systems

This section discusses approaches that either incorporate physical priors in their architecture or have been designed to predict a physical system.

Physics-inspired models The natural video prediction techniques that have been discussed so far are based on the assumption of stochasticity. Nevertheless, many real-world spatio-temporal data come from well-defined processes and using prior physical knowledge can lead to better statistical models [39]. Cressie et al. (2015) showed that auto-regressive Bayesian inspired by partial differential equations (PDEs) perform better in spatio-temporal prediction tasks [39] while Raissi et al. (2017) used Gaussian Processes with covariance functions derived from the temporal discretization of PDEs to quantify the uncertainty in noisy data [40].

Forecasting architectures can also be derived from generic physical laws, like the conservation of energy or motion laws, by apply appropriated constrains. Stewart et al. (2017) use NNs to predict the trajectory of a pillow in an unsupervised manner by exploiting the fact that an object moving under the effect of gravity traces a parabola [41]. NNs have also been applied to extract motion dynamics from a single frame by matching against a predetermined set of Newtonian motion scenarios [42]. These approaches introduce very strong assumptions and fail when they encounter unseen situations. Greydanus et al. (2019) proposed NNs inspired by Hamiltonian mechanics that can predict conservative physical systems but the effectiveness of this technique in large scale non-conservative models remains unknown.

Learning physical systems Learning the parameters of an underlying physical process is akin to system identification. Wu et al. (2015) use NNs to find the mass, density and friction from unlabeled videos and subsequently use a physics simulator to predict future states [44]. The drawback is that a physics engine is needed for simulations. Others train NNs to answer questions like how many degrees of freedom the

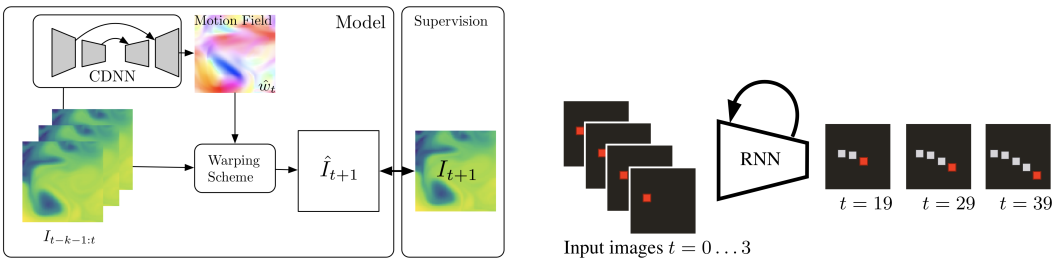


Figure 2.2: **Left:** Feed-forward CDNN, the output must be fed back to the model recursively. Image from [43] **Right:** In recurrent models the internal representation can be propagated to generate an output of arbitrary length. Image from [20]

system exhibits [45]. It is also possible to identify the PDEs that govern a system from data using Gaussian Processes [46] or NNs [47]. A major limitation of these methods is that they require a concrete formulation of the governing differential equations.

Object-centric methods introduce a relational inductive bias to capture the dynamics of the scene. Models with compositional structure that can capture pairwise interactions of rigid objects include CNNs [48], memory-based VAEs [49] and graph networks [50]. While these models are promising, they are limited in the amount of object that they can model while long-term predictions are still a challenge.

Intuitive physics Intuitive physics is a field in cognitive psychology, that studies how humans perceive the physical world and what makes us capable to answer simple questions like if an object is going to fall. DL has been developed to predict the effect of forces on parts of an image [51] or learn how stable a block of towers is [52]. Those methods, though, do not try to model the evolution of the environment. A popular choice in physical systems is the encoder-forecaster discussed earlier. Fragkiadaki et al. (2015) use a convolutional encoder and an LSTM forecaster to learn the dynamics of billiards [53] while Erhardt et al. (2017) use a similar architecture with a convectional encoder and decoder to predict the motion of sliding balls [20]. The problem with these methods is that the quality of predictions deteriorates quickly with time.

Intuitive physics are also very useful in real-life robotics because they provide a good first approximation for modelling the environment. Incorporating them in the scope reinforcement learning is an open research question [54].

2.3 Fluid flow approximation

Here we discuss the other side of the spectrum: models specifically designed to approximate fluid flow by making stronger assumptions like, for example, divergence-free velocity fields.

Eulerian and Langrangian approaches Simulation methods for fluid flow are divided into two broad categories: Eulerian methods which predict the velocity and pressure fields over a discrete mesh and Langragian methods that take into account the properties and interactions between the individual objects [55]. To make accurate predictions, traditional Langrangian methods like Smoothed Particle Hydrodynamics (SPH) simulate a large number of molecules. Because handling so many objects is hard for current DL architecture, Jeon et al. (2015) used regression forests and hand-crafted features to predict the velocity of each particle in the next time-step and speed-up the simulation [56].

Eulerian methods include a projection step where the pressure field is calculated by solving the Poisson equation. This is computationally expensive and machine and deep learning techniques can replace numerical computations speed-up the simulation process. Thompson et al (2017) [57] designed CNNs motivated by the linear system structure to speed-up the pressure projection step while Thurey et al. (2018) [1] estimate both the pressure and velocity fields using a U-net architecture to improve flow prediction around an airfoil.

Transport phenomena Fluid flow belongs to the class of transport phenomena, processes characterised by the exchange of physical quantities within a system. Wave propagation involved mass and energy exchange so related works are highly relevant to this project. In the work closest to this project, Sorteberg et al. (2019) [58] approximate the solution to wave propagation with based on CNN encoder/decoder and 3 LSTMs, showing promising results for long-term predictions. This work was the first to address the wave propagation problem we use it as the baseline. Deep learning has been also used to predict sea surface temperature by predicting the motion field with a CDNN and applying a Gaussian warping scheme to enforce smoothness [43] (Figure 2.2). A similar CDNN is evaluated in this project for wave propagation. GANs conditioned on boundary conditions have been proposed to infer solutions for steady flow in lid-driven cavity problems [59] but this work is only concerned about the steady-state. Furthemore, Seo and Liu (2019) demonstrate that physics-informed graph networks can be used to predict temperatures on a spatial grid [60].

Chapter 3

Background

In this chapter, we introduce some spatio-temporal forecasting concepts which are important for following our experimental design process, evaluation and discussion. We present the problem formulation of spatio-temporal forecasting and discuss relevant multi-step approaches. We also introduce two classes of networks that we will be evaluated, feed-forward and recurrent neural networks (RNNs), the nuances between refeeding and propagating, along with relevant building blocks for each class.

3.1 Spatio-temporal forecasting

Time-series forecasting is defined as the prediction of the next values in a sequence by using the values that came before. In this case, the data are one-dimensional and the values that are known and need to be predicted are scalars. Spatio-temporal forecasting extends this idea to two or higher-dimensional data. More specifically, the goal in forecasting wave patterns from image sequences is to use previous frames to produce a series of next frames. Each frame can be considered as a two-dimensional $M \times N$ rectangular grid represented by a tensor $\mathbf{X} \in \mathbb{R}^{M \times N}$. As it can be seen in Figure 1.1 each frame in the sequence corresponds to a specific time after the wave propagation has initiated. Given a sequence of previous L observations $\mathbf{X}_{t-L+1}, \mathbf{X}_{t-L+2}, \ldots \mathbf{X}_t$ we want to obtain the most likely length K sequence of future frames. The fundamental constraint is that only previous frames can be used. Formally the problem can be defined as follows:

$$\tilde{\mathbf{X}}_{t+1}, \ldots, \tilde{\mathbf{X}}_{t+K} = \underset{\mathbf{X}_{t+1}, \ldots, \mathbf{X}_{t+K}}{\arg\max}\ p\left(\mathbf{X}_{t+1}, \ldots, \mathbf{X}_{t+K} | \mathbf{X}_{t-L+1}, \mathbf{X}_{t-L+2}, \ldots, \mathbf{X}_t\right) \tag{3.1}$$

Finding the real conditional probability function of Equation 3.1 is a difficult task. In this project, we use deep convolutional and recurrent networks to find good approximate solutions without explicitly learning a conditional probability distribution. The task, then, becomes learning a network that will have the minimal expected loss, like MSE, between the predictions and the ground truth. This problem is different and more challenging than the typical single-step time series forecasting because the output is not a single element but a series of non i.i.d elements $\tilde{\mathbf{X}}_{t+1:t+K}$ and also the series is not only time-dependent but also spatially correlated.

3.2 Convolutional Neural Networks

Due to their unique ability in capturing spatial features, convolutional layers are used in virtually every spatio-temporal prediction architecture. In recent years, purely convolutional models have been successfully used in various spatiotemporal prediction tasks [32, 43, 1]. Furthermore, recent studies have demonstrated that convolutional networks can capture time-dependent correlations on 1D sequential data at least as good as baseline recurrent models [16]. In this project, we use concepts like convolution, deconvolution, dilation, dimensionality reduction and skip connections. We assume a basic knowledge of those concepts and we refer the reader to Section A.1 of the Appendix for detailed explanations.

3.3 Recurrent Neural Networks

The other major network architecture relevant to this project are Recurrent Neural Networks (RNNs). RNNs contain an inner state that propagates as the network unfolds in time. By sharing weights across time RNNs can capture long-term dependencies that would otherwise be impossible to do with a feed-forward net. This is very important for time-series prediction because an event depends on the events that preceded it.

A very popular form of recurrent networks is Long-Short Term Memory (LSTM) modules. [17]. LSTMs solved a major problem of previous RNNs models: the loss of temporal information due to vanishing gradients. Their major innovation was to include a cell state c_t which corresponds to longer-term memory and allows for more efficient backpropagation through time.

LSTMs have been extensively used in sequence modelling tasks like video prediction [19, 20, 18]. They also form a key building block in our baseline model for wave

propagation forecasting [58]. Next, we will present two recent variations of LSTM used in spatio-temporal prediction: ConvLSTMs and Causal LSTMs. For more about how original RNNs and LSTMs work we refer the reader to Section A.2 of the Appendix.

3.3.1 Convolutional LSTM

Despite their effectiveness in video prediction, LSTMs use fully connected gates which require a large amount of memory. The current GPU memory and memory bandwidth bottlenecks render training such models troublesome. Moreover, recurrent models unfold linearly and this limits the potential gains from hardware parallelization.

Another drawback of using LSTMs for video prediction is that the internal and cell states are lower-dimensional vectors that hold little analogy with the spatial characteristics of the signal. To address these drawbacks Xi et al. (2015) proposed Convolutional LSTM (ConvLSTM) that uses convolutional gates instead of fully connected ones in both the input-to-state and state-to-state transitions [23]. This reduces the number of parameters while at the same time exploits the structural information on the data. The information flow in an ConvLSTM cell is the same as in Equations A.3 but instead of matrices the inputs x_t, the cell state c_t, hidden state h_t and gates i_t, f_t, o_t are 3D tensors and the matrix multiplications are replaced by convolutional operations. The update equations for a ConvLSTM cell are the following:

$$\begin{aligned} i_t &= \sigma\left(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i\right) \\ f_t &= \sigma\left(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f\right) \\ \mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh\left(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c\right) \\ o_t &= \sigma\left(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o\right) \\ \mathcal{H}_t &= o_t \circ \tanh\left(\mathcal{C}_t\right) \end{aligned} \tag{3.2}$$

where $\circ$ is the Hadamard product and $*$ denotes the convolution operation. The way convolutions work in ConvLSTMs is illustrated in Figure A.3 of the Appendix.

Stacked ConvLSTMs (Figures 3.1) were initially built to address precipitation nowcasting, but have been used in other spatio-temporal sequence forecasting problems and nowadays are considered state-of-the-art in short-term video prediction [18].

3.3.2 Causal LSTM

Although the cell state in LSTMs captures considerable longer relationships over plain RNNs, vanishing gradients are still an issue and deep-in-time architectures have trouble with backpropagation through time [61]. Causal LSTMs have been designed to maintain a constant flow of gradients for longer time [29]. They achieve that by introducing using a spatial memory M_t^k separate from the temporal cell memory C_t^k, where the subscript t denotes the time step and the superscript k is the layer.

While the current temporal memory C_t^k depends on its previous state C_{t-1}^k the additional spatial memory M_t^k depends on its counterpart of the previous layer M_t^{k-1}. The new memory M_t^k is allowed to move both spatially from one layer to the other and also in time from the top layer of the previous state to the bottom layer of the next. Essentially the two memories are allowed to zigzag in different directions: vertically across stacked layers and horizontally through states. The difference in information flow between Causal and Convolutional LSTM is depicted in Figure 3.1. A Causal LSTM cell at time-step t and layer k has the folling update equations:

$$
\begin{aligned}
\begin{pmatrix} g_t \\ i_t \\ f_t \end{pmatrix} &= \begin{pmatrix} \tanh \\ \sigma \\ \sigma \end{pmatrix} W_1 * \left[\mathcal{X}_t, \mathcal{H}_{t-1}^k, \mathcal{C}_{t-1}^k \right] \\
\mathcal{C}_t^k &= f_t \odot \mathcal{C}_{t-1}^k + i_t \odot g_t \\
\begin{pmatrix} g_t' \\ i_t' \\ f_t' \end{pmatrix} &= \begin{pmatrix} \tanh \\ \sigma \\ \sigma \end{pmatrix} W_2 * \left[\mathcal{X}_t, \mathcal{C}_t^k, \mathcal{M}_t^{k-1} \right] \\
\mathcal{M}_t^k &= f_t' \odot \tanh \left(W_3 * \mathcal{M}_t^{k-1} \right) + i_t' \odot g_t' \\
o_t &= \tanh \left(W_4 * \left[\mathcal{X}_t, \mathcal{C}_t^k, \mathcal{M}_t^k \right] \right) \\
\mathcal{H}_t^k &= o_t \odot \tanh \left(W_5 * \left[\mathcal{C}_t^k, \mathcal{M}_t^k \right] \right)
\end{aligned}
\tag{3.3}
$$

A schematics of the Causal LSTM cell be seen in Figure A.4 of the Appendix. Overall, equipping the model with memories that traverse both longer and shorter routes, from input frames to the expected future predictions, grants it more flexibility in capturing spatial correlations and short-term dynamics. Experiments in the original paper demonstrate that stacked Causal LSTMs outperform ConvLSTM and other state-of-the-art models in short and longer-term predictions tasks.

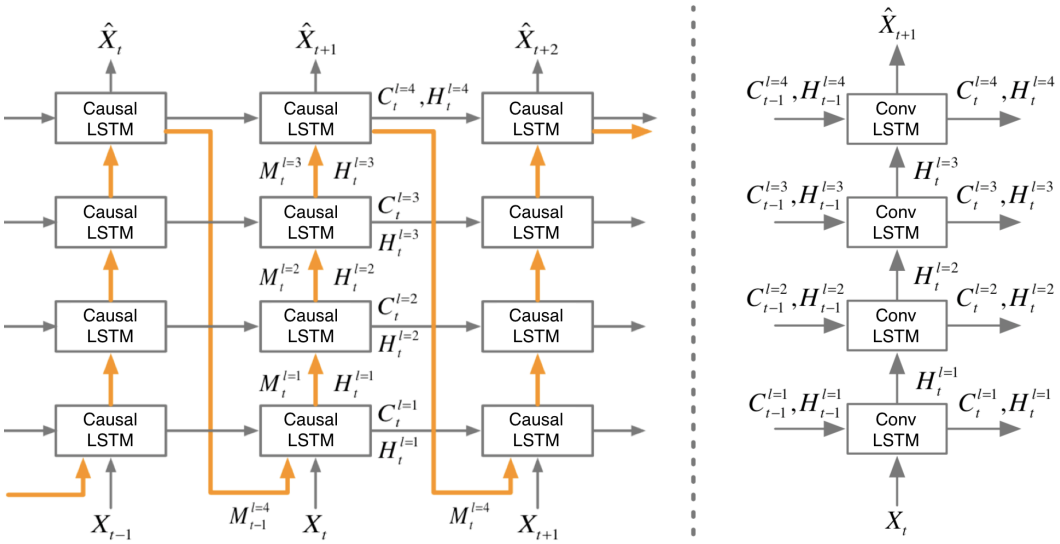


Figure 3.1: **Right:** The conventional ConvLSTM architecture has a hidden state H_t and cell state C_t. **Left:** Causal LSTMs contain an additional memory M_t^l, where t the state and l the layer, that moves ziz-zag across layers and states (orange arrow). Image adapted from [29].

3.3.3 Sequence to Sequence Modelling

In contrast to feed-forward networks that produce a fixed-length prediction, recurrent models can be quite flexible when it comes to the length of the predicted sequence, which is known as sequence-to-sequence modelling. Initially used for machine translation this framework is now widely applied in sequential data of unknown length [62]. The approach uses an encoder-forecaster structure. The encoder processes the input at each time-step and converts it to a latent representation in the form of the internal states if the recurrent module. This representation is copied over to the recurrent decoder to produce the desired output sequence. By unfolding the internal state in time the decoder can produce an arbitrarily long sequence. The recurrent models we evaluate use this encoder-forecaster approach as can be seen in Figures 4.1, 4.7 and 4.9.

Chapter 4

Experiments

This section presents our experimental evaluations. First, we describe the dataset, metrics and the common setup that has been used across experiments. Two datasets were developed for training the networks and six more datasets to test generalization. Two evaluation metrics are also presented. Next, five models are evaluated: an LSTM based encoder-forecaster as a baseline, a fully convolutional NN with dilations, two state-of-the-art models based on ConvLSTM and Causal LSTM and a Convolutional-Deconvolutional NN. We also propose an inference method for recurrent models that leads to significantly lower long-term prediction error. Lastly, we explore the generalisation capabilities of the models in unseen datasets and identify, through further experiments, that model struggle with different wave speeds. We propose two method to enhance generalisation by enforcing the models to be reversible and present our findings.

4.1 Dataset acquisition

All the datasets for this project were created by using Scikit-FDiff, a fast numerical solver, to simulate the Saint-Venant equations (Equation 1.1) under solid boundary conditions (Figure 1.1). The Coriolis force and viscosity were neglected and kinematic viscosity was set to $10^{-6} m^2/s$ which is close to water viscosity at 20°C. The reference height H is set to 10 m while the size of the square water tank is randomly selected in each run from 10 to 20m. An initial excitation of a Gaussian droplet with varied strength and location was applied in each simulation. The time step was set to 0.01 sec while each sequence had a total length of 100 frames or 1 sec. A total of 3,000 sequences were rendered with lighting azimuth of 45 degrees and an altitude of 20

Dataset Name	Initial Condition	Depth	Tank Size	Illum. Azimuth	Sequences
Original	Droplet	10	[10, 20]	45	3000
Double Drop	Double Droplet	10	[10, 20]	45	500
Lines	Line wave	10	[10, 20]	45	500
Opposite Illum.	Droplet	10	[10, 20]	135	500
Random Illum.	Droplet	10	[10, 20]	Random	500
Shallow Depth	Droplet	5	[10, 20]	45	500
Small Tank	Droplet	10	[5, 10]	45	500
Big Tank	Droplet	10	[20, 40]	45	500
Fixed Tank	Droplet	10	10	45	3000

Table 4.1: The original dataset was used for training, model selection and evaluation. Models were also trained with the fixed tank dataset to study the effect of tank size. All the other datasets were used to evaluate the generalisation capabilities of the models.

degrees. Each frame was saved as a 184×184 image.

We also created 7 datasets to evaluate the generalization capabilities of the networks. The first two have different initial conditions: two droplets and linear waves. The third dataset has opposite illumination azimuth (135°) and the fourth was rendered with random illumination azimuth. The fifth dataset has shallower reference height H. The last two datasets were created to study the effect of tank size, or equivalently the wave speed. One of them uses a bigger and the other a smaller range of tank sizes. To study further the effect of tank size we created an extra dataset with constant tank size and used it to train and test the models. A summary of all 9 datasets can be found in Table 4.1.

4.2 Metrics and evaluation

Which error metric is best in spatio-temporal forecasting is not always straightforward and depends on the nature of data. In this project, we use two metrics: Mean Square Error (MSE) and the Structural Similarity Index.

The MSE between a target Y and a prediction $\hat{Y}$ is computed pixelwise as:

$$MSE = \frac{1}{T \times m \times n} \sum_{t=1}^{T} \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij}^{t} - \hat{y}_{ij}^{t})^2 \tag{4.1}$$

where T is the length of the sequence, N, M the dimensions of each frame and $y, \hat{y}$

are the pixel values of the target and prediction respectively. MSE was used both for training and evaluating our models.

One problem with MSE when assessing image quality is that two predictions with similar MSE might vary greatly in content. SSIM was designed to improve on that aspect over traditional methods like MSE and peak signal-to-noise ratio (PSNR) [63]. It can be viewed as a quality measure that captures the perceived similarity between two images, more akin to how a human would compare them. The SSIM index is calculated between two windows x, y with the following formula:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{4.2}$$

where μ, σ^2 are the mean and variance of the pixel values and c_1, c_2 two variable to avoid division by zero. The SSIM between two images is computed by averaging the measure over various windows.

4.3 Experimental setup

4.3.1 Data pre-processing

Each data sequences has 100 frames of dimensions 184×184 while the pixel values range from 0 to 1. In our experiments, the frames were resized down to 128×128 to reduce training time and memory footprint. We also used image normalisation which is known to improve performance on image prediction tasks [64]. Initial experiments showed that normalising the pixels values to zero mean and standard deviation 1 works best. Note that the normalising values are computed from the training set alone and applied to the validation and test sets. Data augmentation techniques like horizontal and vertical flips were employed on a per sequence basis. From the 3000 sequences of the original dataset, 70% were used for training, 15% for validation and 15% for testing.

4.3.2 Model selection methodology

We train the model by randomly selecting K sub-sequences of length $N+M$ from each sequence, where N, M are the number of input and output frames of the model respectively. The MSE loss between the predictions and targets is backpropagated and the weights are updated by using an Adam optimiser. A scheduling scheme adjusts the

learning rate (LR) by a scaling factor if there is no improvement in validation error for some epochs (patience).

To choose the best model for each architecture we did hyper-parameter tuning. For the initial LSTM model we did extensive grid search over the input length $N \in \{3,5,10\}$, training output length $M \in \{1,5,10,20\}$, samples per sequence $K \in \{1,3,5,10\}$, batch size $b \in \{16,32\}$, LR $\in \{10^{-2},10^{-3},10^{-4},10^{-5}\}$, LR scaling factor 0.1, and patience $p \in \{5,7\}$. Training intents to minimise the MSE of an output sequence of length M. We also noticed that batch size did not matter much in performance so for each model we chose the biggest batch size that would fit the GPU memory to reduce training time. For the rest of the architectures, we used a smaller grid search informed by our initial experiments on the baseline model. This means that we might have missed some better configurations but given the time constraints and the computational resources available for this project it was necessary. Because each model takes different time to conclude an epoch we fixed the training budget not in epochs but in time. We trained all models for both 16h and 24h hours because we noticed that for some models extra training time worked well while others showed signs of overfitting. The models were trained to minimise the MSE over their respective output length. This poses a comparison problem because models with fewer output frames tend to produce a lower error because it is easier to learn the dynamics over a shorter horizon. Hence, during model selection, we chose the hyper-parameters that gave the lowest validation error for the next 50 frames. The full set of hyperparameters and training times can be found in Tables B.1 and B.2 of the Appendix. Models were implemented in pytorch and trained on a GTX 1060 GPU with 6GB of memory. The code for the LSTM model was provided by authors and was significantly refactored. Publicly available code was adapted in developing the rest of the models [65, 66, 67].

At test time the models predict for a horizon of 80 time-steps after the last input frame. To test their extrapolation, MSE and SSIM values are plotted against future time step and three dummy baselines were also used to provide context: the last frame of the input, the previous frame and the flat surface i.e. an image where all the pixels correspond to height H from Equation 1.1.

4.4 LSTM Encoder-Forecaster

A natural choice for our baseline is the work of Sorterberg et al. [58] which was the first to address the wave propagation with a DL framework. It is based on the encoder-

forecaster (E-F) architecture using 3 LSTM cells.

Recurrent neural networks have been successfully applied to time-series and sequence forecasting problems like speech recognition or natural language processing [18]. These architectures are tailor-made for one-dimensional sequence and cannot be directly applied to spatio-temporal data. LSTMs, in particular, contain various fully connected layers in the form of gates. The problem with fully connected layers is that the number of weights grows exponentially with the number of input values. For example, a tensor of 5 video frames with dimensions $128x128$ pixels would require an 81.920-dimensional vector which would then require $O(n^2)$ parameters. Even if computation and memory was not an issue, there are spatial correlations between adjacent pixel values that are lost by simple concatenation of the input.

From a practical point of view using a convolutional encoder to reduce the input frames to a lower-dimensional latent space can help with both problems. In the encoder-forecaster framework, the encoder reduces the images to latent vector. This is passed as input to an LSTM that propagates it in time recursively producing latent representations of future frames. The deconvolutional part of the decoder reconstructs the predictions from the latent vectors. This approach has been used to predict natural videos [19] and physical system such as sliding balls [20].

The model we consider from [58] extends the E-F architecture to wave propagation forecasting and was the first work to address this problem. It employs a convolutional encoder/decoder scheme and 3 LSTMs for propagation (Figure 4.1).

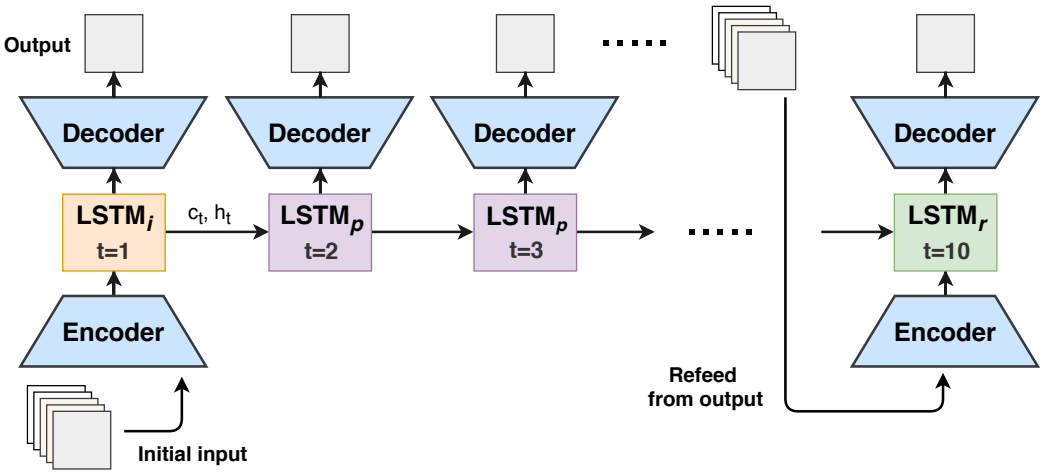


Figure 4.1: **LSTM Encoder-Forecaster Model.** Three LSTMs are used: one for capturing the input, one for propagation while the last is fed from the output of the model

4.4.1 Model description

The input to the model is N grayscale frames. The encoder consists of 4 convolutional layers with 60, 120, 240, 480 feature maps, kernel sizes 7, 3, 3, 3 and padding of 2, 1, 1, 1 pixels. Dimensionality reduction is achieved by using a kernel stride of size 2 in all layers. After each convolutional layer, there is a batch normalisation layer and a *tanh* non-linearity which is a commonly used choice. In the last convolutional layer, dropout is used on 25% of the units, that are chosen randomly in each pass. Note, that batch normalisation and dropout are only used in training and not for inference. The final part of the encoder is a fully connected layer of width $\mathcal{L} = 1000$ which is the latent vector representation of the input. For the forecasting part, the model uses three LSTMs and a decoder. The decoder is based on deconvolutions that double the spatial dimensions of the feature maps in each layer until the original 128×128 size is reached. It is a mirror of the encoder in terms of feature map size while the kernel is 3, the padding is 1 and the stride is 2 for all the layers. Figure 4.2 depicts the information flow through the architecture during one forward pass.

The three LSTMs are used different times of the prediction process. One of them is used to capture the input of the sequence and learn an internal representation, the second is used to recurrently propagate the representation for a number of steps without receiving any input while the third LSTM takes as input the last N prediction to update the internal representation and the process continues up to the requested number of output frames M. This process can be visualized in Figure 4.1 where orange, magenta and green colours denote the three LSTM. It can also be seen that while each LSTM has its own learned parameters, they do share the same inner hidden and cell state c_t, h_t during the process. During training, we optimize the MSE for output length M. At inference time we can get longer predictions by recursively following the same scheme.

The length of the latent vector $\mathcal{L}$, the number of input frames is N, the number of output frames M and the refeeding offset R are hyperparameters of the model. We kept the same $\mathcal{L}$ as the original model that has 3 LSTMs of length 1000 and the refeeding offset at $\frac{M}{2}$ frames. For comparison reasons the latent vector of [19] uses length of 2048 or 4096, while [20] use length 128 but a slightly different architecture.

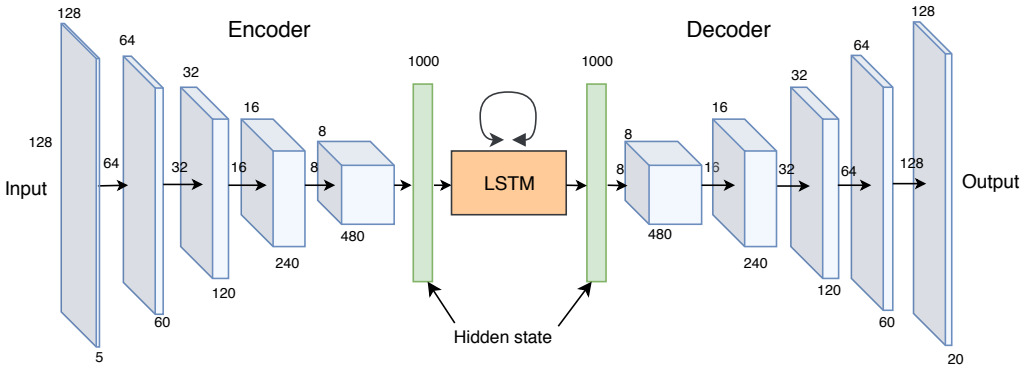


Figure 4.2: **Schematic of the encoder and decoder used in the LSTM model.** Dimensions or layers are left and up, number of channels at the bottom.

4.4.2 Evaluation

We assess the model performance both quantitatively and qualitatively. The image quality is measured as the MSE between the predictions and ground-truth. Figure 4.3 (left) shows a plot of the average MSE over the whole test set for the model and various baselines. The LSTM model (orange line) achieves an MSE of less than 0.04 for the first 10 frames when the first reinsertion happens. After that error accumulation is faster until $t = 20$ where MSE reaches 0.9 continues up to frame 80 (MSE 0.19). The fact that the model works reasonably well until $t = 20$ which was the training horizon is a sign of over-fitting to the training scheme. The model outperforms its last input frame(dashed blue line), an indication that it attains some knowledge about the dynamics of the system. By comparing to the flat line baseline (dotted green line), though, this seems to be mostly the case 20 first time-steps, after that the gains are not clear. Given that the motion is smooth, the previous frame baseline (red line) is very good approximation which, interestingly, the LSTM model outperforms up to the first 10 frames. A similar insight is drawn by the SSIM metric (Section B.3 in the Appendix).

To qualitatively asses the model predictions we plot them against the ground truth over various time intervals (4.3 - right). Despite the degradation in MSE, the predictions do maintain the relevant shape of wave propagation. Looking closer at the intensity profile of the line with the greatest variance (Figure 4.4) it seems that the prediction propagates slower than the ground truth and also includes visibly noise.

Overall, the model seems to be predicting best until internal re-feeding starts at $t = 10$ and shows signs of over-fitting to the training horizon. This requires further investigation on the what role the refeeding and the 3 distinct LSTMs play. While MSE

degrades considerably after that, the predictions do maintain the qualitative characteristics of the wave propagation.

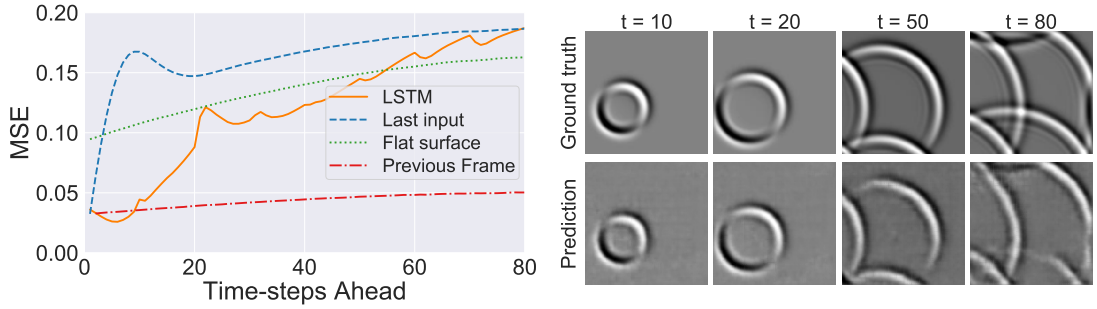


Figure 4.3: **LSTM Model Evaluation** - **Left:** Average MSE of predictions on the test set. The LSTM model has lower MSE than the last input and flat surface baselines for the most part. Given that the motion is smooth, the previous frame is a very good approximation that the model outperforms up until up to $t = 10$. **Right:** The predictions of the LSTM model preserve the qualitative characteristics of the ground truth.

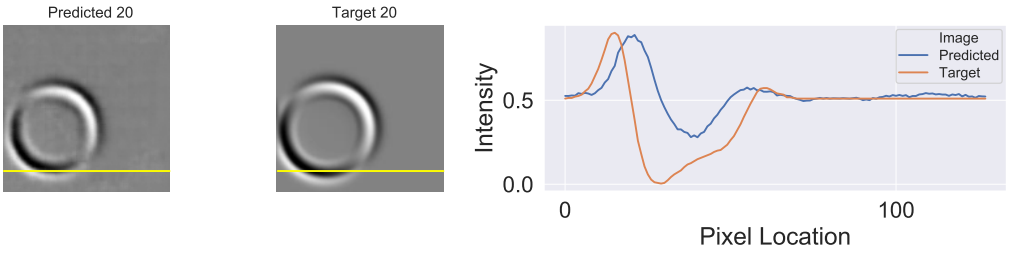


Figure 4.4: **Qualitative assessment of the LSTM model. Left:** Prediction of the LSTM model and ground truth at t=20. The yellow line corresponds to the horizontal dimension with the largest pixel variation in the ground truth image. **Right:** Intensity profile across the yellow line on the left reveals that the prediction of the LSTM model is slower than the actual wave and noisier.

4.5 CNN with Dilations

RNNs have been historically considered better suited for sequence modelling than feedforward neural networks due to their internal memory. Recent findings though, have challenged this idea and have shown that causal CNNs are at least as good in a variety of tasks [16]. Whether this holds for spatio-temporal data has not been studied in depth. The problem studied in this project, a deterministic dynamical system, offers

a good testbed to draw comparisons. This model is based on dilated convolution while in a later experiment we evaluate a CDNN architecture. Wave propagation is based on partial differential equations and the rationale for using an FCN is that if it is deep enough the convolutional filters will be able to capture the transformations needed to simulate dynamics of the system.

Our model is composed of convolutional layers without any dimensionality reduction. This means that the spatial dimensions of the input are preserved to the output. Dilations of the kernel are used to increase the receptive field and enable the aggregation of long-distance characteristics of the image. For a detailed explanation of dilations, we refer the reader to Section A.1.2.

4.5.1 Model Description

The implementation is designed to provide a straightforward baseline. Inspired by ResNet [64], it has 10 convolutional layers each followed by a ReLU layer. The overall architecture and the number of feature maps on each layer can be seen in Figure 4.5. All layers have kernel size 3 and padding of 1 to avoid dimensionality reduction. After the first convolutional layer, there is dimension-preserving max-pooling layer as in ResNet. To help the network capture features with larger spatial extent, we increase the receptive field of the kernels by using dilation of 2 and 4 in the last two blocks. Since the network is relatively shallow skip connections are not necessary and were omitted for simplicity.

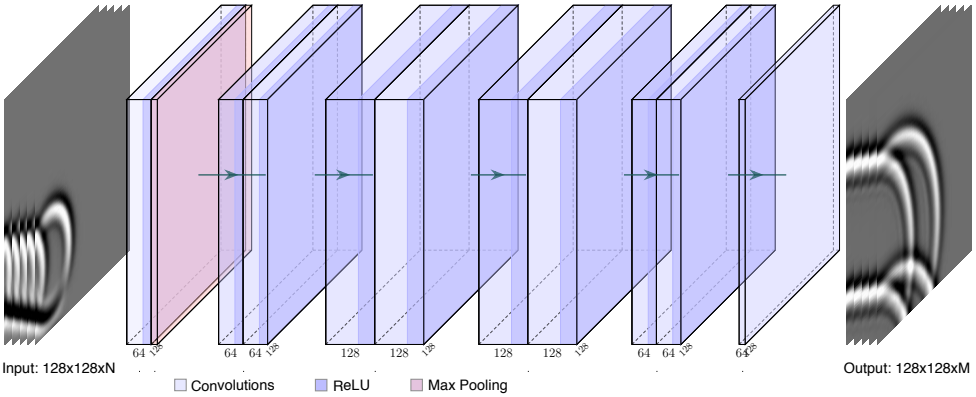


Figure 4.5: **CNN Model Architecture.** All layers preserve the original dimensions 128×128. The number of feature maps is on the bottom. Our model takes N=5 frames as input and outputs M=10 frames

4.5.2 Evaluation

In this experiment, we wanted to investigate if convolutions alone can learn the differential equations of wave propagation. Since feed-forward models can only predict a fixed amount of output frames, during inference we generate longer predictions by re-feeding the last N=5 frames for the output back to the model.

The MSE plot in Figure 4.6 (left) indicates that the CNN model has lower error than the LSTM baseline up to approximately time-step $t = 35$. After that point, both models have similar MSE. This indicates that the CNN model has enough capacity to maintain some memory. Furthermore, this results comes from a model that has less than 1% of the trainable parameters of the LSTM model: 0.8M for the CNN vs 88M for the LSTM. Additionally, the CNN curve is linear even extrapolates similarly even after the training length $t = 10$. One explanation for that could be that in LSTM, the memory over-fits to the short-term characteristic of the signal. The CNN model, on the other hand, learns a representation of the dynamics that are time-invariant, so recursively applying it give the roughly the same error accumulation over time.

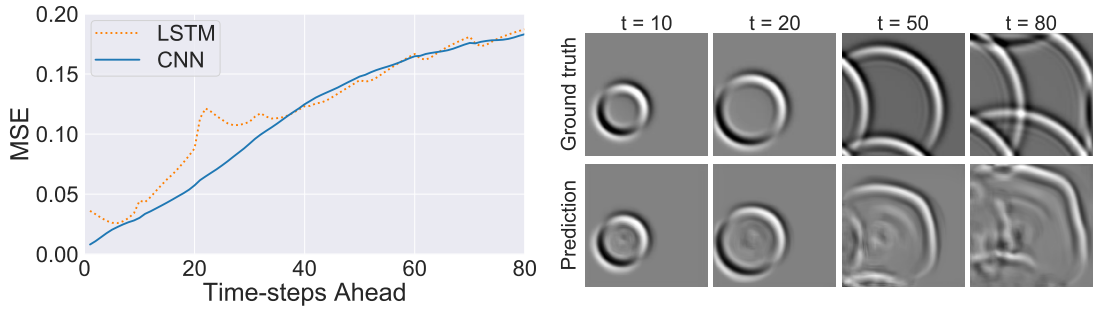


Figure 4.6: **CNN Model Evaluation** - **Left:** Average MSE of predictions on the test set. The CNN model has comparable or better performance to the baseline LSTM model. **Right:** CNN model predictions distort in a way that resembles real ripples.

Qualitatively the predictions seem quite different (Figure 4.6 - right). Here the wave pattern is not conserved that well in later time-steps. Ripple effects that start as early as $t = 10$ get accentuated and lead to distorted shapes which, perhaps, resemble closer how real waves propagate. In later time-steps, the prediction is noticeably lagging against the ground truth. Another difference is that the amount of noise is visibly less, which could an explanation for better performance in earlier parts of the MSE curve. Those differences illustrate the difficulty in assessing video prediction. While the MSE error is equal for $t = 80$ the models give perceptually different predictions. We also calculated the SSIM index to capture the perceptual similarity but, unfortu-

nately, for this problem, it does only provide the same insights as MSE (Figure 4.6). The SSIM plots can be found in the Appendix Section B.3.

4.6 Convolutional LSTM

The next model we evaluate uses a different idea to apply recurrent models to spatio-temporal data. Instead of using convolutions to reduce the input to a latent vector, Convolutional LSTMs (ConvLSTMs) make use of convolutions internally in the LSTM cell. The rationale behind this is to create a synergy between the temporal efficiency of the LSTMs with the spatial inductive bias of convolutions. ConvLSTMs are considered state-of-the-art on short-term video prediction. The motivation is that if a representation can provide good short-term results, it should be able to extrapolate beyond what has been observed, especially in a deterministic system as the one studied here. For a detailed explanation of ConvLSTM cells see Section A.2.2.

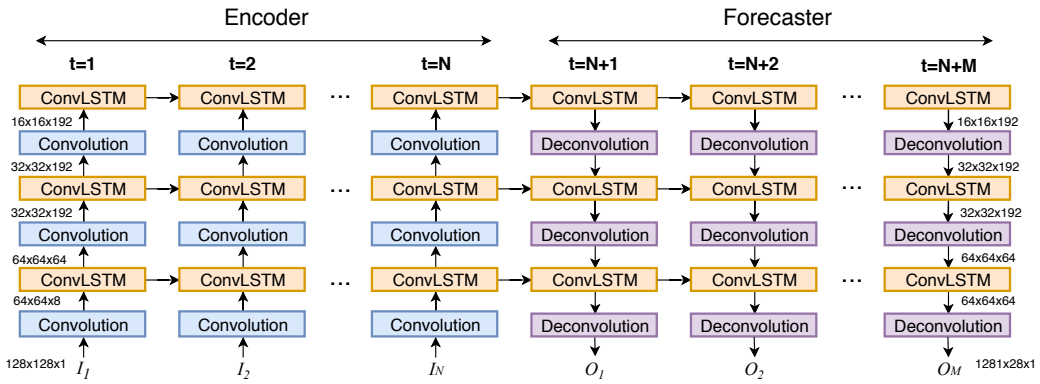


Figure 4.7: **ConvLSTM Model** The encoder processes the N=5 input frames one at a time to create an internal representation. The representation gets copied over to the forecaster that uses it to generate M=10 future frames. The feature map dimensions can be seen next to each layer.

4.6.1 Model description

Our design was inspired by the findings of the original paper ConvLSTM paper [23]. First of all, we followed the same encoder-decoder architecture that stacks multiple ConvLSTM layers to create a model with strong representational power, suitable for predicting complex dynamical systems like wave propagation. Furthermore, the authors found that ”deeper models can produce better results with fewer parameters” and

this guided the choice of using 3 layers.

Before each ConvLSTM layer, we use an extra convolutional layer with 8, 64, 192 feature maps respectively. These numbers are similar to the original paper that used two layers of 64 maps. Furthermore, according to the authors, using a kernel with size more than 1 is essential for capturing motion patterns. Driven by that, all the convolutional layers have kernel size 3 and zero paddings of width 1. The nonlinearities that are used are Leaky ReLUs with negative slope of 0.2 [68]. Stride 2 is used to reduce the dimensionality. Reducing the dimensionality serves two purposes, first it increases the receptive field of the kernel in the next layer, allowing it to capture long-distance relationships. At the same time, it reduces the number of parameters of the model and its computational complexity. At the final ConvLSTM layer the input is represented by a $16 \times 16 \times 192$ tensor.

Inside the ConvLSTMs the output feature maps must be equal to the input feature maps of the preceding convolutional layer so no dimensionality reduction happens. This is enforced by the need to recurrently feed the output when unfolding in time. All ConvLSTMs use kernels of size 3, again as per the guidance of the authors, and zero padding of 1 pixel to avoid the dimensionality reduction. The ConvLSTMs of the forecaster has the same dimensionality as the ones in the encoder because they need to share the same inner state which is copied over from the last time-step of the encoder. The forecaster uses deconvolutions with stride 2 to double up the pixel dimensions in each layer. The three layers have 192, 64, and 8 feature maps and a kernel size of 4. Since the prediction target has only one channel we feed the feature maps in the last layer into a 1×1 convolutional layer to generate the final output. The hyperparameters of the model can be seen in Table B.1

Overall, the encoding ConvLSTMs reduce the whole input sequence into a set of hidden state tensors and the forecaster ConvLSTMs unfold this hidden state into the future to generate the prediction. (Figure 4.7).

4.6.2 Evaluation

Figure 4.8 (left) depicts the MSE curve for the ConvLSTM model along with the LSTM model and flat surface baseline. The ConvLSTM model works best until roughly time-step 15. After that, the performance degrades rapidly until it plateaus around frame 35. The error accumulation starts shortly after the training output length $M = 10$ is reached which aligns with the previously reported nature of ConvLSTMs as pre-

dominantly short-term predictors. In that range though, ConvLSTM with MSE 0.22 at frame 10 and 0.36 at frame 15 outperforms the LSTM model that has 0.44 and 0.62 respectively. Furthermore, the ConvLSTM is about 7 times smaller with its 11M trainable parameters.

Qualitative results in Figure 4.8 (right) confirm the previous finding. For up to $t = 10$ the ConvLSTM provides an almost impeccable restoration of the ground-truth. Yet, for $t = 50$ the prediction losses a big part of the wave and at $t = 80$ it is almost a flat image. By comparing the MSE of the flat surface (green dashed line) with the ConvLSTM for $t > 40$ we see that they are almost equal which suggests that the ConvLSTM predictions degenerate to something close to a flat surface. This overfitting pattern is different from that before. In LSTM the long term predictions had the qualitatively wave-like appearance and the MSE error was coming from time-constant misalignment plus the noise. Here, the short term predictions are extremely accurate but extrapolating in the future is not reliable. One would expect that for a deterministic problem like this, a low short-term error would lead to better long-term predictions too. This is not the case and we revisit this problem by proposing a better way for inference in Section 4.8.

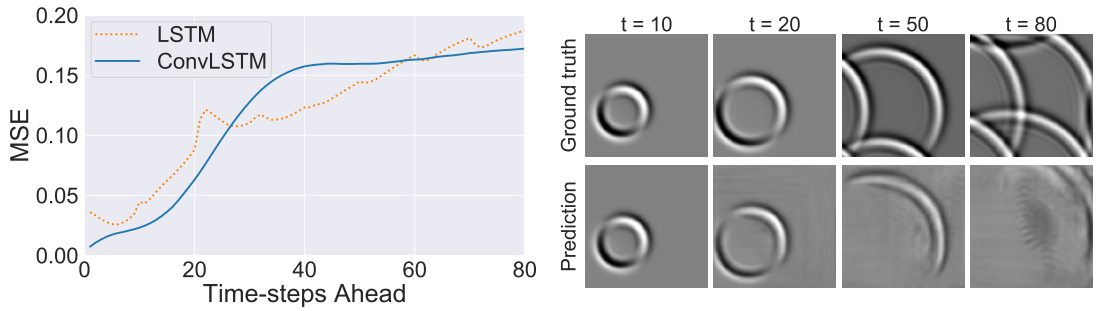


Figure 4.8: **ConvLSTM Model Evaluation** - **Left:** Average MSE of predictions on the test set. The ConvLSTM model clearly outperforms the baseline LSTM model in short-term prediction ($t < 20$). Prediction quality deteriorates fast after that. **Right:** While early predictions have good quality, for t=50 a big part of the wave is lost and by t=80 it degenerates to an almost flat image.

4.7 Causal LSTM

This is the second recurrent model we evaluate that uses convolutions internally in the LSTM cell. It is based on stacking various layers of Causal LSTMs that are described

in detail in Section 4.7. Causal LSTMs use an additional spatio-temporal memory and have been demonstrated to perform better than ConvLSTMs in forecasting tasks. [29]. Why include another model that primarily targets short-term prediction? Since the wave propagation system is deterministic, even small gains in short-term performance could potentially lead to significantly lower error accumulation in the long run.

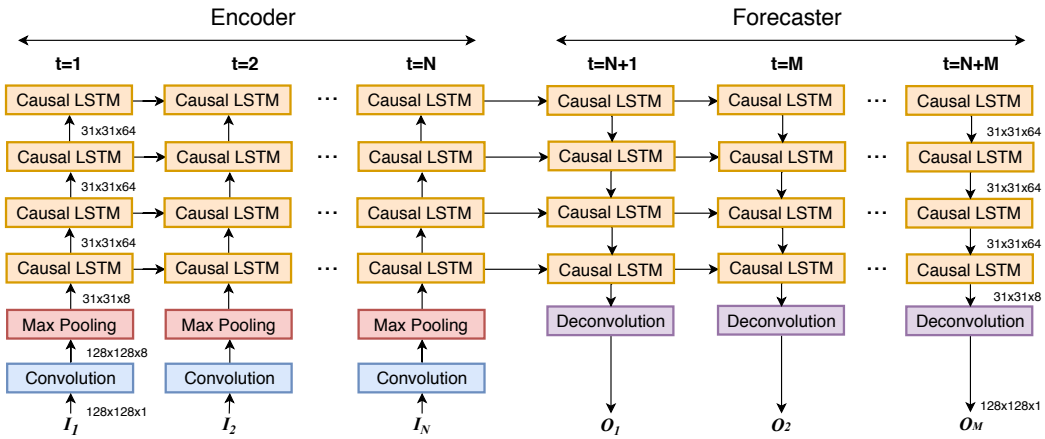


Figure 4.9: **Causal LSTM Model** The encoder processes the N=5 input frames one at a time and an internal representation is created in each Causal LSTM cell. These representations get copied over to the forecaster that uses them to generate M=20 future frames. The feature map dimensions can be seen next to each layer.

4.7.1 Model Description

As in all our recurrent architecture, the encoder-forecaster framework is used for training and inference. The unfolding of the model through time is presented in Figure A.4. The vertical stack is comprised of one convolutional, one max pooling and four Causal LSTM layers. The convolutional layer has a kernel of size 3, no padding and outputs 8 feature maps. In the original paper, they do not use any dimensionality reduction because their input dimensions are 64×64 per frame. Our input dimensions (128×128) are too big to fit in available GPU memory, so we used max-pooling with stride 4 to reduce the dimensions to 31×31 pixels. Following the original paper, we used 4 Causal LSTM layers but reduced the size of all of them to 64 channels each (they used 128 for the first layer), again to meet hardware memory constraints. The authors set the convolution kernel size inside the recurrent units to 5. Since our effective dimensions are smaller we set the kernel size to 3. The forecaster uses a deconvolutional layer with a kernel of size 7 and stride 4 to restore the internal state to the original dimensions.

4.7.2 Evaluation

The Causal LSTM model is considered state-of-the-art in short-term video prediction and this is also verified by our experiments. As illustrated in Figure 4.10 (left), the Causal LSTM based model significantly outperforms in MSE both the LSTM baseline and the ConvLSTM model in the time-steps up to $t = 40$. Causal LSTM not only takes advantage of a longer training horizon ($M = 20$) but also keeps the low prediction error further until $t = 30$. This is something that both previous models fail to with their error starting to increase much earlier. The additional spatio-temporal memory in the Causal LSTM model efficiently captures dependencies in short time scales. Long-term prediction, though, remains the problem with the model performing worse than the other two for approximately $t > 50$.

Comparing visually the actual predictions vs the ground-truth (Figure 4.10 - right) it seems that Causal LSTM preserves well the wave-like appearance. The problem that arises is that the predicted waves seem to propagate faster the real ones. That is the opposite to what was happening with LSTM where the predictions were slower and to ConvLSTM where the prediction collapsed to a flat surface. Under these observations, comparing the MSE of the three models for $t > 50$ reveals that qualitatively better predictions can potentially lead to results worse than the flat image. One explanation could be a catastrophic phase difference: the lows in the prediction coincide with the high of the ground-truth and vice-versa. This is a caveat for not using only quantitative metrics when comparing methods. Furthermore, it demonstrates that wave speed is a very important factor that seems to be impacted by error accumulation.

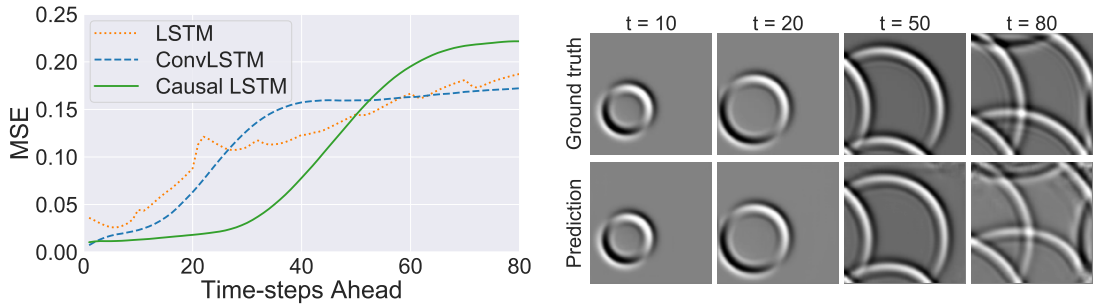


Figure 4.10: **Causal LSTM Model Evaluation** - **Left:** Causal LSTM further improves short-term prediction by a big marging, maintaining MSE below 0.10 for 43 frames, almost twice as long as ConvLSTM (23) but it has the same steep deterioration. **Right:** Causal LSTM prediction preserve the wave patterns but there is a phase mismatch. Opposite to the LSTM model (Fig. 4.4), the prediction is ahead of the ground truth.

As with ConvLSTM, the short-term efficacy of Causal LSTM does not extrapolate well to future predictions. In the next section, we try to exploit the deterministic nature of the problem by proposing a better way for inference.

4.8 Refeeding RNNs

The recurrent models that have been evaluated so far (LSTM, ConvLSTM, Causal LSTM) have noticeable differences but share one thing in common: they provide good predictions for a brief duration and then steeply deteriorate (Figure 4.10 - left). RNN based models for natural video prediction are known to be better suited for short term prediction [16]. Nevertheless, wave propagation has a strong element of determinism and we would expect that when a model learns an adequate representation of the dynamics then applying the same operation to produce the next frame could should always work irrespective of the input. After all, the underlying physics remain the same. It seem that the internal representation overfits the training horizon and is unable to produce stable predictions after a point. The CNN model on the other hand, while it is not the best performing network, displays a more linear degradation (Figure 4.6 - left).

Driven by these observations we propose a different inference mechanism for RNN models that exploits the combination of good short-term performance the deterministic nature of the problem. Normally in RNN encoder-forecaster the encoder learns a representation from the input which is copied over to the forecaster and gets unfolded in time to generate the predictions. This scheme has been followed in ConvLSTM and Causal LSTM (Figures 4.7 and 4.9) while the LSTM model is slightly different because the predictions are being fed back to the second LSTM (Fig. 4.1). Here we propose a different strategy. The forecaster only generates M frames, where M is the length of the output used during training. Then the last N predictions, where N the input length of the model, are being fed back to the encoder and a new iteration begins. This is repeated until the requested number of frames is reached (80 for evaluation). Why would that work? The motivation is that the models produce generally good predictions within the training horizon M. By re-feeding those predictions back to the model the encoder create a new internal representation is which can be then again propagated with low error for another M steps. The system dynamics, i.e. the underlying physics, do not change so a good model applied recursively should provide better results. Effectively, we trying to constrain the model from reaching the territory of disastrous error accumulation.

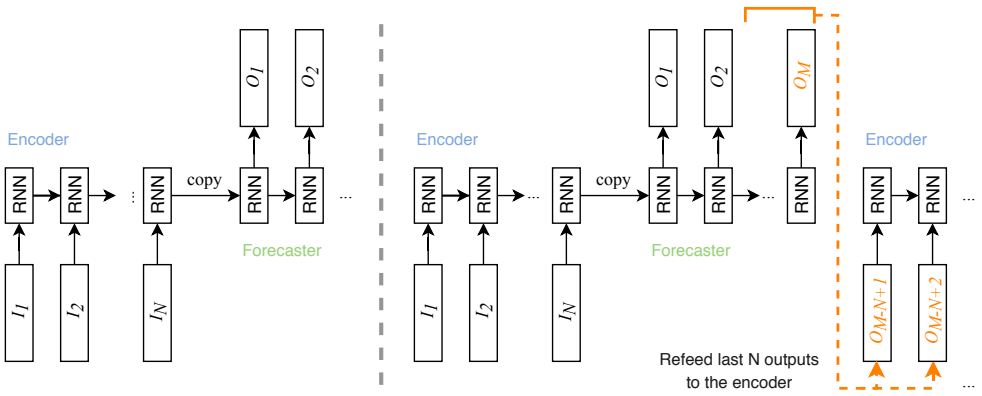


Figure 4.11: **Inference for RNN models - Left (standard method):** The encoder learns a internal representation from the input which gets copied over to the forecaster and used to predict future states of arbitrary length. **Right (proposed method):** The forecaster generates M frames, where M the output length during training. The last N (input length) frames are fed back to the encoder and the process starts over.

4.8.1 Evaluation

We evaluate the three RNN models, without retraining then, using the proposed inference scheme. Results in Figure 4.12 demonstrate that our inference method generates predictions with substantially lower MSE for the ConvLSTM and Causal LSTM models. In the case of the ConvLSTM model, the normal inference method reaches an MSE of 0.10 at time-step 25 while with refeeding this happens only at frame 50, twice as slow. The results for Causal LSTM are even more prominent. The previous method reached an error of 0.10 at time-step 43 and a maximum error of 0.22 at time-step 80 while the proposed method will only reach a maximum error of 0.09 at frame 80. In both cases, the new error curve is much close to being linear. This confirms our assumption that reusing the whole model can indeed reset the error accumulator of the internal state and exploit the determinism of the system. However, the outcome of the LSTM model was mixed. There are two possible explanations for that. First, contrary to the other two models the LSTM model was exhibiting already a steep increase in MSE just before it reached the training length $M = 20$, so refeeding was done with predictions of mediocre quality. Secondly, in the LSTM architecture, one of the 3 LSTMs had a refeeding mechanism already in place (Figure 4.1).

Figure 4.13 shows in more detail what changed for the Causal LSTM model. We compare the target (80 time-steps ahead) with predictions generated internal propagation and the proposed re-feeding method. From the actual frames, we can see that the

waves with normal inference are faster than the ground truth while re-feeding stays much closer to the correct speed. This is further verified by the intensity profile on the same figure. These and previous finding for LSTM suggest that models lose the time-constant and rectifying that can improve the results.

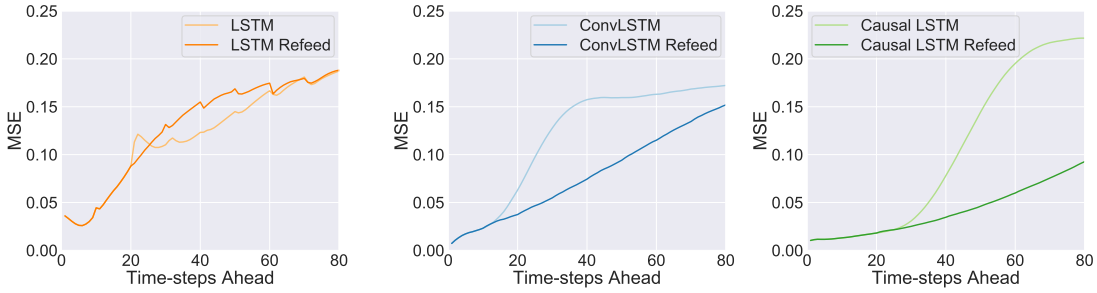


Figure 4.12: **Effect of proposed inference method (refeeding) to RNN performance. Left:** Refeeding does not improve the LSTM model probably because one of the LSTMs in the model was already refed. **Center, Right:** Refeeding significantly improves ConvLSTM and Causal LSTM predictions and makes the error accumulation more linear.

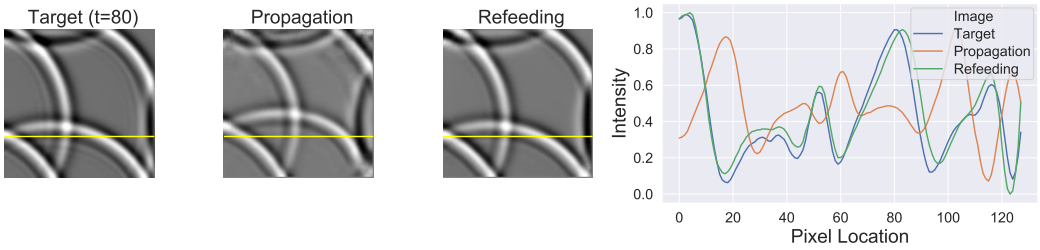


Figure 4.13: **Qualitative assessment of refeeding. Left:** Ground truth at t=80, predictions of Causal LSTM with internal propagation and prediction of the same model using refeeding. The yellow line is the most varying horizontal dimension of the ground truth. **Right:** Intensity profile across the yellow line on the left indicates that refeeding (green line) is much closer to the ground truth (blue line) than the previous method.

4.9 Convolutional-Deconvolutional Neural Network

The second purely convolutional models that we evaluate in this project is a Convolutional - Deconvolutional NN based on U-Net [69]. While initially used for image segmentation, U-Net variations have been employed to estimate optical flow [32] and to forecast physical processes like surface temperature [43] and fluid flow [1]. This makes CDNNs a promising candidate for the prediction of wave propagation. It would,

further, provide a stronger feed-forward competitor to the RNN based models than the Dilated CNN of Section 4.5.

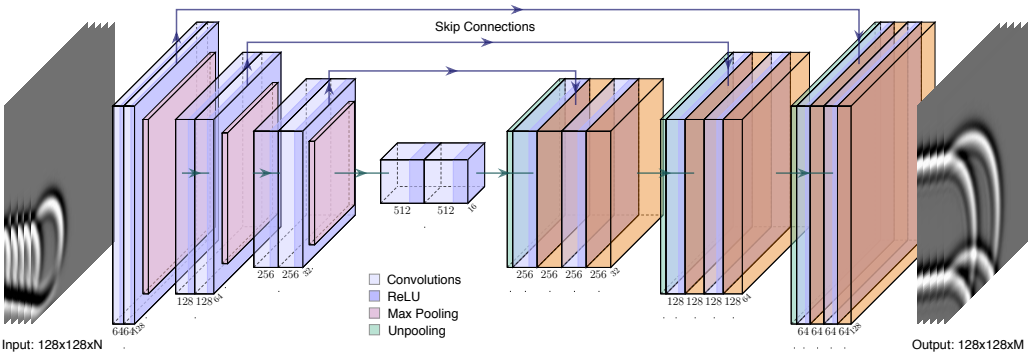


Figure 4.14: **Schematic of the CDNN model.** The number of channels is below each layer and its dimensions on the side. Our model has input N=5 and output M=20.

4.9.1 Model Description

CDNNs are composed of two parts. The contracting part reduces the frame dimensions while it increases the number of channels or feature maps. Reversely, the expanding part uses unpooling to restore the dimensions to the original size. In our implementation, the contracting part is composed of fours blocks each containing two convolutional layers with kernel size 3 and padding 1, followed by ReLU non-linearities. The first three blocks include a max-pooling layer of stride 2 that reduces the size in half. The number of feature maps doubles in each layer. We mostly follow the implementation of the original U-Net but instead of 5 blocks, we use 4 because our image dimensions are smaller. Other researchers have also effectively used fewer blocks for smaller image sizes [43]. For the expanding part, we use bilinear interpolation with scale factor 2 instead of deconvolutions to keep the number of parameters low. Skip connections are also employed to copy feature maps from earlier layers. This way, high-level, coarser feature maps are combined with fine-grained local information of lower layers. Using skip connections is important for propagating context information to higher resolution layers. The block sizes and dimensions can be seen in Figure 4.14.

One major difference in our implementation with U-Net is that we do not crop the feature maps before copying them. This is because for frame prediction information from the whole frame needs to be preserved because the location is important. A similar approach has been used in FlowNet [32]. Inspired by [43] we train the model in an end-to-end manner but we predict multiple frames at once instead of only one.

4.9.2 Evaluation

The question is whether a feed-forward model that has been previously used in physical problems can learn the dynamics of this problem and how its performance compares to the rest of the evaluated methods. As in previous experiments, we use the model to predict for 80 time-steps after the last input frame and compute the MSE on the test error. Results in Figure 4.15 demonstrate that the CDNN model we have derived improves the MSE error by a large margin compared to the original LSTM model. Furthermore, it performs on par with Causal LSTM-R up to $t = 40$ and after that point shows an even lower error. This is also evident by the qualitative results on Figure 4.15 where it is almost impossible for the human eye to detect any difference even at $t = 80$.

Judging by the test set MSE error, CDNN is the best model. First of all, why is this model better than the CNN we tried before? It seems that dimensionality reduction is more effective than dilations in capturing multi-scale image features. Also, skip-connections between feature channels allows information from different scales to be present at the same time. This helps with localisation during the upsampling and leads to a more precise output. It further indicates that recurrent memory is not so important for this problem. This might have to do with the determinism because the physics do not change. Going from one point in state-space to the next involves applying the same set of principles. Furthermore, the dynamics of the system are based on a set of PDEs and recent research has showed that CNNs with skip connections are linked to ODEs. For example a ResNet layer u can be written as the Euler discretization $u_{n+1} = u_n + \Delta t f(u_n)$. It might be that the CDNN is inherently capable of solving differential equations but further experiments are needed to justify such a link.

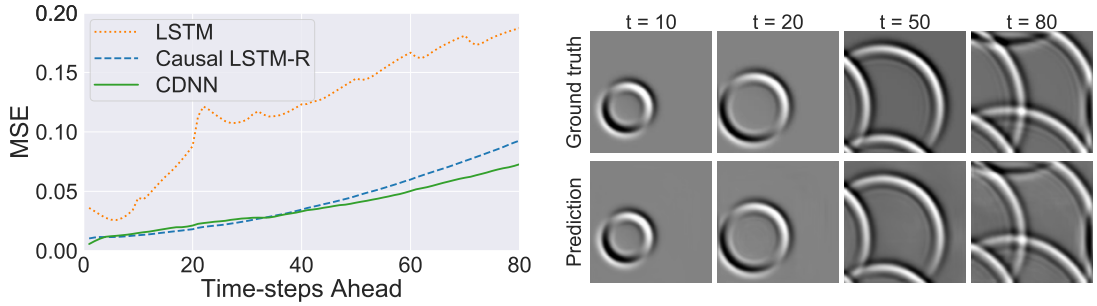


Figure 4.15: **CDNN LSTM Model Evaluation** - **Left:** CDDN is the best performing model on the test set. It performs on par with Causal LSTM-R for $t < 40$ and outpeforms it beyond that. MSE error stays below 0.05 for almost 60 time-steps ahead. **Right:** Long-term CDNN predictions are almost indistinguishable from the ground truth.

4.10 Generalization

Apart from asses the models on the left-out test set from the original dataset we also test the generalisation capabilities in seven different datasets: lines, double drop, opposite illumination, random illumination, shallow depth, smaller tank, bigger tank (see Section 4.1 for details). For conciseness, we focus on the best performing models, Causal LSTM-R and CDNN, that also represent the recurrent and feed-forward categories. A comprehensive list of figures can be found in Section B.4 of the Appendix.

Results in Figure 4.16 show that both models exhibit similar generalisation capabilities. The networks seem to be quite robust to changes in illumination. The MSE for opposite illumination azimuth (135°) is indistinguishable to the original test set. This indicates that the network can deal with a perpendicular phase change in lighting conditions. Random illumination does make things harder for the models but still has the lowest MSE among the generalisation datasets. Lines and double drop are the two datasets that have different initial conditions. Of the two, lines appear to be quite challenging with a much higher MSE than the original test set. Qualitative results in Figure 4.18 show that while both networks capture the propagation of the wavefront they also introduce a lot of circular artefacts, reminiscent of the data they were trained upon. The double drop seems to be less challenging than linear waves since it is more similar to what the model has observed. Two datasets with different tank sizes have been included to study the effect of wave speed. It seems that a smaller tank size, or equivalently faster waves, confuses the network more than any other dataset. The bigger tank also exhibits high MSE, only behind the smaller tank and the lines. Looking at the actual predictions in Figure 4.18 (e) and (f), we observe that both networks indeed miss the time constant: predictions fall behind ground truth in the smaller tank and move faster in the bigger tank. These findings coincide with our previous qualitative observations that the networks can not keep in phase with the actual waves due to error accumulation. Similar conclusions are drawn from the SSIM charts in Section B.4 of the Appendix.

To better understand why such big differences exist we focus on the datasets. Figure 4.17 shows the MSE of the flat image (left) and previous frame (right) when used as predictions. We observe that lines and smaller tank have the highest MSE for the flat image while the big tank has the least. This is expected because in the smaller tank waves propagate fast and more pixels deviate from the reference height and in linear waves the wavefront is large from the first frame. The previous frame baseline indi-

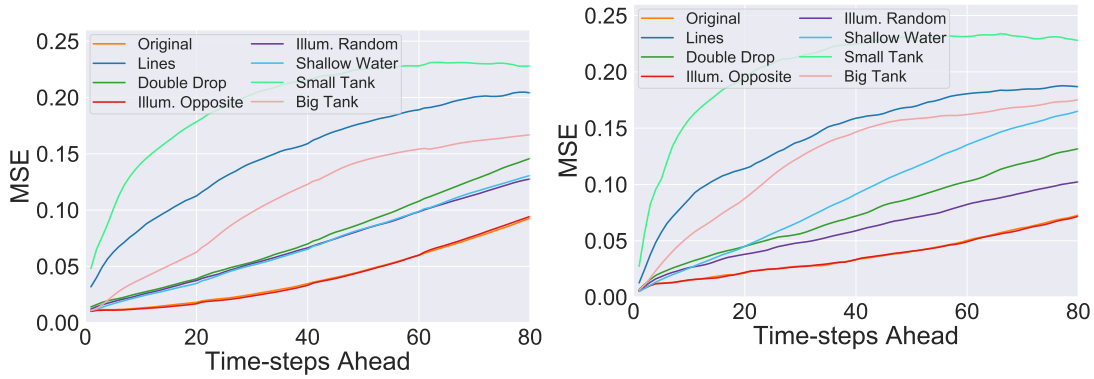


Figure 4.16: Generalization of **Causal LSTM-R (left)** and **CDNN (right)** across different datasets. The faster waves in the smaller tank are hard for models to predict (turqoise line). Linear waves, also, have high MSE due to the different initial conditions (blue line). Interestingly, opposite illumination (red line) seems to have no effect at all.

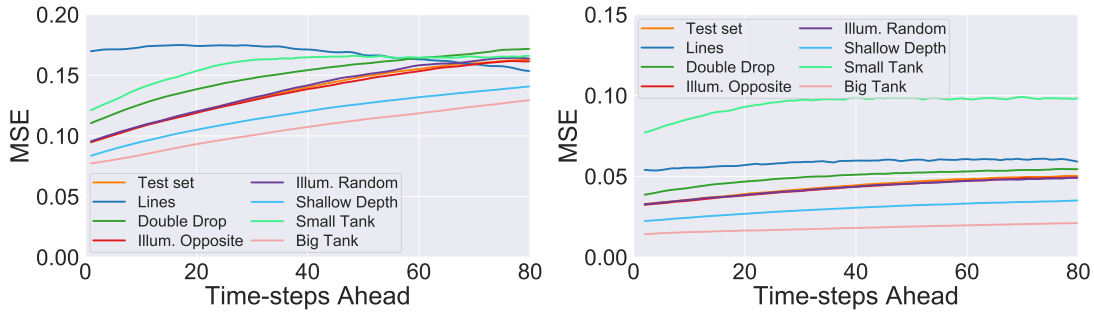
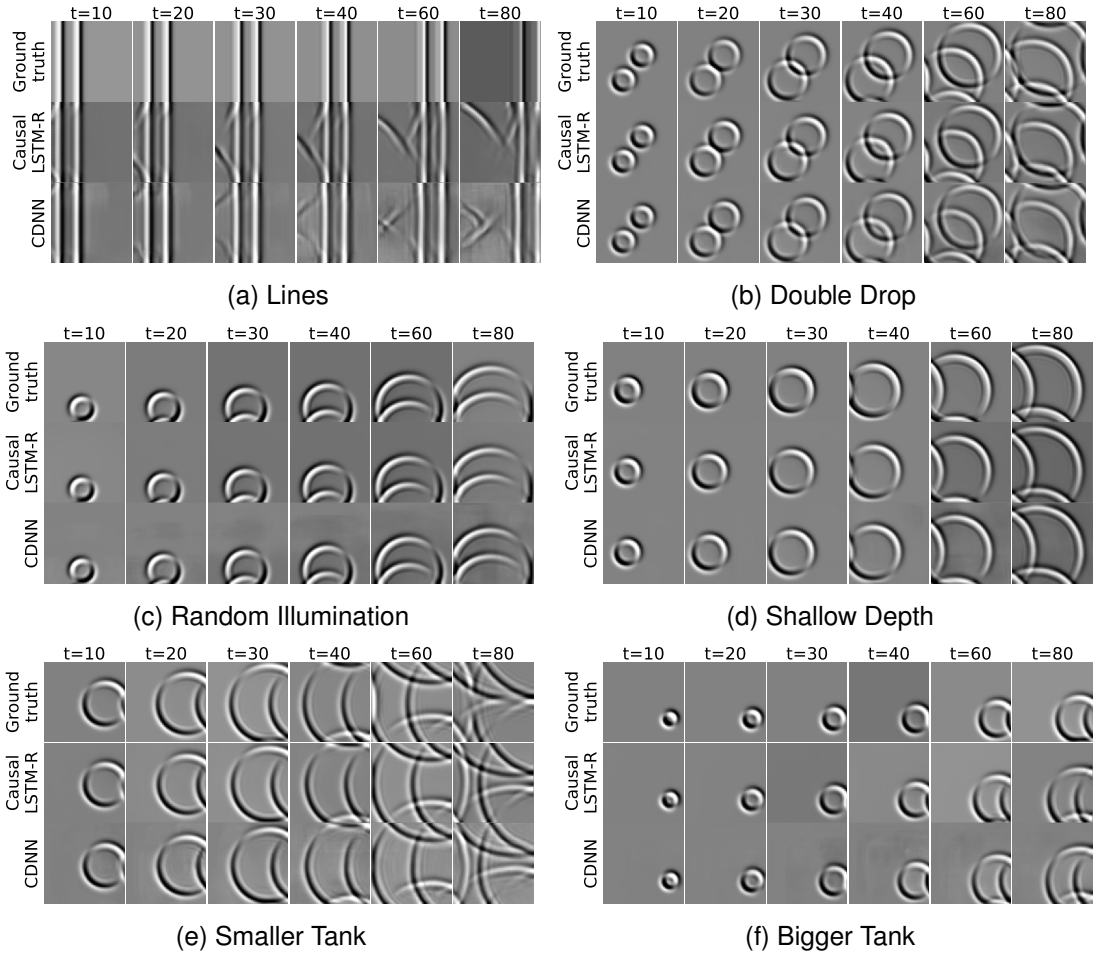


Figure 4.17: **Flat Image (left)** and **Previous Frame (right)** baselines compared against ground truth across different datasets. The flat image baseline measures how much the frame is away from the reference height. The Previous Frame indicates how much consecutive frames differ. Datasets are not equally hard to predict and this should be taken into account when assessing the generalisation capacity of a model.

cates how much consecutive frames vary. Again, as expected, the small tank has the highest MSE due to highest speed but the movement of lines seems much smoother.

Overall we note that the difference in smoothness between datasets or, in physical terms, the rate of entropy increase is a confounding factor for MSE and SSIM differences. One should be aware of this fact when using such metrics to assess the generalisation capacity of a model. Using baselines to benchmark, as done here, helps but a more systematic comparison approach should be explored.



(a) Lines (b) Double Drop

(c) Random Illumination (d) Shallow Depth

(e) Smaller Tank (f) Bigger Tank

Figure 4.18: Qualitative assessment of the Causal LSTM-R and CDNN models over 6 generalization datasets. Predictions of the line dataset contain circular artefacts. The time difference between ground truth and predictions is most obvious in the smaller and bigger tanks.

4.11 Training with constant speed waves

To understand better how the wave speed affects performance we created a dataset that has a fixed tank size. This means that all the sequences in the dataset have waves that propagate at the same speed. What we want to discover is whether the mixed tank sizes in the original dataset confuse the network or make it generalise better. Results show that the latter is true. On the left part of Figure 4.19 we compare two variations of CDNN, one trained with the original dataset and one with fixed tank data. Using a fixed tank for training harms generalisation in every case except in the test set. But this is only because the test set in this case has the same wave speed as the training set which the network manages to learn well.

4.12 Making the network reversible

Since the system is deterministic, a network that forecasts the future from the past should, in principle, be able to decipher the past from the future. We train the CDNN with the original dataset but half of the time we reverse the order and ask it to predict the past from the future (CDNN Reversible). Results in Figure 4.19 (right) demonstrate that this helps the generalisation in other datasets, especially in lines, at the cost of losing some accuracy in the original test set. A closer look at a prediction in the lines dataset in Figure 4.20 indicates that the CDNN Reversible model seems to be producing straighter lines that keep closer to the ground truth. While this is by no means a thorough analysis, it could be an indication that reversibility is a viable way to improve deterministic system forecasting.

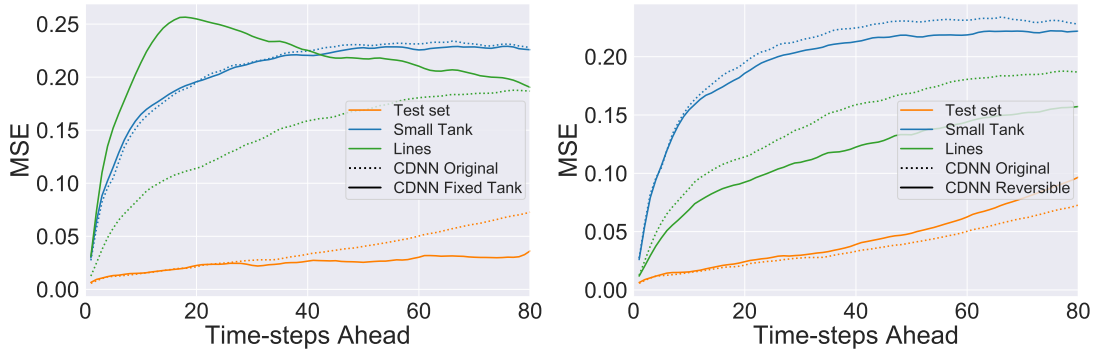


Figure 4.19: **Left:** Training the CDNN using a dataset with fixed tank size (solid lines) harms generalisations. **Right:** A deterministic induction bias was applied to CDNN by making it reversible, i.e. able to predict the past from the future (solid lines). This improves the generalisation on the lines dataset but slightly harms test set error.

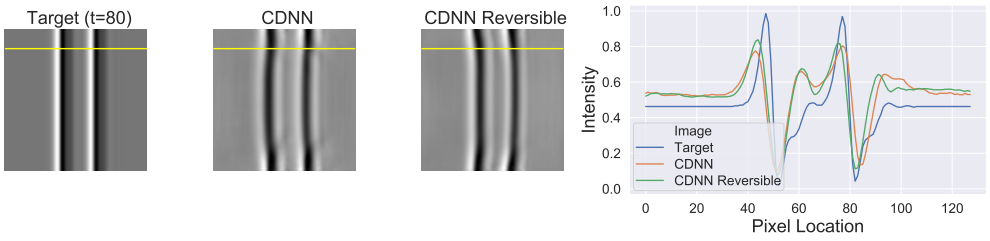


Figure 4.20: Ground truth at t=80, CDNN prediction and CDNN reversible prediction. The intensity profile on the right shows that the *reversible* CDNN (green line) is closer to the ground truth (blue line) than the original CDNN.

Chapter 5

Conclusions

This report evaluated various neural network architectures for forecasting wave propagation. Since the problem is a physical system an open question was how to exploit the inherent determinism. In total 5 different models were evaluated across 8 datasets. We first established a baseline LSTM model and showed that its predictions carry the qualitative characteristics of the wave even long-term. Then a simple dilation-based CNN model was evaluated. The assumption was that convolutions alone should be able to learn the underlying differential equations of the model since it is Markovian. It was shown that the CNN performs on par with or better than the baseline but the predictions were visibly distorted in the long run. Nevertheless, this provided a first indication that memory-based networks are not necessarily the only solution. Next, ConvLSTM and Causal LSTM were assessed, two recurrent architectures known for producing state-of-the-art short-term spatio-temporal predictions. ConvLSTM improved short-term predictions over the baseline while Causal LSTM provided significant gains up to mid-range, extrapolating beyond the horizon it was trained upon. Both those models, though, displayed a tipping point after which error accumulated very fast. We proposed a new inference method that exploits the combination of good short-term performance and the determinism of the problem. Effectively, we refed the recurrent network before the internal propagation reached the territory of rapid error accumulation. The inductive bias is that if a model learns a good representation of the dynamics then applying the same operation should work irrespective of the input. This was shown to substantially improve the long-term performance of both models and turned their error curves close to linear. Finally, we evaluated CDNN, a more powerful feed-forward model than the CNN, and demonstrated that it yields similar or better performance than the Causal LSTM, even when the latter is used with refeeding. This is another in-

dication that feed-forward models are capable of capturing spatio-temporal sequence dynamics at least as good as RNNs. This corroborates findings of other researchers in one-dimensional data so more work is needed to figure out if feed-forward models are only good when used in deterministic settings or if they can capture the complex dynamics in general.

The generalisation capabilities of the models were evaluated against very different datasets. The networks had difficulties to extrapolate to unseen wave speeds. Illumination seems to have the least effect, with most models being invariant to opposite illumination. The models were even able to predict a linear propagating wavefront. Furthermore, we used baselines to show that some datasets have more inherent variance than others and using metrics like mean square error should always be done with cautiousness. To increase robustness in different wave speeds, we proposed training with two different datasets. In the first all the samples had the same wave speed, to find out if the different speeds in the dataset was confusing the network. This setup harmed generalization. We also used the original data to make the model reversible in time i.e. trained it to predict the future from the past and the past from the future. This reversible architecture showed some improvement in generalisation and further analysis is required to show if there is merit to this direction.

For future work, there are few directions to explore. Even though the system is deterministic, generalisation in different conditions will inevitably introduce some uncertainty. One way to cope with that uncertainty is to use models like Generative Adversarial Networks or Variational Autoencoders. Graph networks could be used to model a particle-based simulation of the flow. Domain knowledge could also be exploited, for example, add a loss on the latent representation inspired by the wave function or convert the data to the frequency domain. More datasets, with different boundary conditions and real-world data, are also interesting to explore.

Bibliography

[1] Nils Thuerey, Konstantin Weissenow, et al. Well, how accurate is it? a study of deep learning methods for reynolds-averaged navier-stokes simulations. *arXiv preprint arXiv:1810.08217*, 2018.

[2] James A Carlson, Arthur Jaffe, and Andrew Wiles. *The millennium prize problems*.

[3] Joel H Ferziger and Milovan Peric. *Computational methods for fluid dynamics*. Springer Science & Business Media, 2012.

[4] E Stupak, R Kačianauskas, A Kačeniauskas, V Starikovičius, A Maknickas, R Pacevič, M STAŠKŪIENĖ, G Davidavičius, and A Aidietis. The geometric model-based patient-specific simulations of turbulent aortic valve flows. *Archives of Mechanics*, 69, 2017.

[5] Cheng Yang, Xubo Yang, and Xiangyun Xiao. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424, 2016.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[7] Clint Dawson and Christopher M Mirabito. The shallow water equations. 2008.

[8] Roger Moussa and Claude Bocquillon. Approximation zones of the saint-venant equations f flood routing with overbank flow. *Hydrology and Earth System Sciences Discussions*, 4(2):251–260, 2000.

[9] Ilya Starodumov, Anna Derevianko, and Dmitri Alexandrov. Application of the saint-venant model and the modified stefan model for modeling the formation of the ice cover at the thermal growth stage. In *IOP Conference Series: Materials Science and Engineering*, volume 192, page 012033. IOP Publishing, 2017.

[10] Serdar Göktepe, Jonathan Wong, and Ellen Kuhl. Atrial and ventricular fibrillation: computational simulation of spiral waves in cardiac tissue. *Archive of Applied Mechanics*, 80(5):569–580, 2010.

[11] David H Wolpert, William G Macready, et al. No free lunch theorems for optimization.

[12] 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.

[13] Ross Goroshin, Michael F Mathieu, and Yann LeCun. Learning to linearize under uncertainty. In *Advances in Neural Information Processing Systems*, pages 1234–1242, 2015.

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

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

[16] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

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

[18] Xingjian Shi and Dit-Yan Yeung. Machine learning for spatiotemporal sequence forecasting: A survey. *arXiv preprint arXiv:1808.06865*, 2018.

[19] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

[20] Sebastien Ehrhardt, Aron Monszpart, Niloy J Mitra, and Andrea Vedaldi. Learning a physical long-term predictor. *arXiv preprint arXiv:1703.00247*, 2017.

[21] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015.

[22] Francesco Cricri, Xingyang Ni, Mikko Honkala, Emre Aksu, and Moncef Gabbouj. Video ladder networks. *arXiv preprint arXiv:1612.01756*, 2016.

[23] Shi Xingjian et al. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NIPS*, pages 802–810, 2015.

[24] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[25] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, pages 667–675, 2016.

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

[27] Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Deep learning for precipitation nowcasting: A benchmark and a new model. In *Advances in neural information processing systems*, pages 5617–5627, 2017.

[28] Yunbo Wang, Mingsheng Long, Jianmin Wang, Zhifeng Gao, and Philip S. Yu. Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 879–888, USA, 2017. Curran Associates Inc.

[29] Yunbo Wang, Zhifeng Gao, Mingsheng Long, Jianmin Wang, and Philip S Yu. Predrnn++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning. *arXiv preprint arXiv:1804.06300*, 2018.

[30] Nal Kalchbrenner, Aäron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. In *Pro-*

ceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1771–1779. JMLR. org, 2017.

[31] Ziru Xu, Yunbo Wang, Mingsheng Long, Jianmin Wang, and MOE KLiss. Predcnn: Predictive learning with cascade convolutions.

[32] Philipp Fischer et al. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[33] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. In *European conference on computer vision*, pages 286–301. Springer, 2016.

[34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

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

[36] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*, 2017.

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

[38] Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. *arXiv preprint arXiv:1802.07687*, 2018.

[39] Noel Cressie and Christopher K Wikle. *Statistics for spatio-temporal data*. John Wiley & Sons, 2015.

[40] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Numerical gaussian processes for time-dependent and non-linear partial differential equations. *arXiv preprint arXiv:1703.10230*, 2017.

[41] Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[42] Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian scene understanding: Unfolding the dynamics of objects in static images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3521–3529, 2016.

[43] Emmanuel de Bezenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. *arXiv preprint arXiv:1711.07970*, 2017.

[44] Jiajun Wu, Joseph J Lim, Hongyi Zhang, and Joshua B Tenenbaum. Physics 101: Learning physical object properties from unlabeled videos.

[45] Raban Iten, Tony Metger, Henrik Wilming, Lídia Del Rio, and Renato Renner. Discovering physical concepts with neural networks. *arXiv preprint arXiv:1807.10300*, 2018.

[46] Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

[47] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. *arXiv preprint arXiv:1710.09668*, 2017.

[48] Nicholas Watters et al. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547, 2017.

[49] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

[50] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. In *Advances in Neural Information Processing Systems*, pages 8799–8810, 2018.

[51] Roozbeh Mottaghi, Mohammad Rastegari, Abhinav Gupta, and Ali Farhadi. what happens if... learning to predict the effect of forces in images. In *European Conference on Computer Vision*, pages 269–285. Springer, 2016.

[52] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. *arXiv preprint arXiv:1603.01312*, 2016.

[53] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*, 2015.

[54] Misha Denil, Pulkit Agrawal, Tejas D Kulkarni, Tom Erez, Peter Battaglia, and Nando de Freitas. Learning to perform physics experiments via deep reinforcement learning. *arXiv preprint arXiv:1611.01843*, 2016.

[55] Z Zhang and Q Chen. Comparison of the eulerian and lagrangian methods for predicting particle transport in enclosed spaces. *Atmospheric environment*, 41(25):5236–5248, 2007.

[56] So Hyeon Jeong, Barbara Solenthaler, Marc Pollefeys, Markus Gross, et al. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)*, 34(6):199, 2015.

[57] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *ICML*, pages 3424–3433. JMLR. org, 2017.

[58] Wilhelm E Sorteberg, Stef Garasto, Chris C Cantwell, and Anil A Bharath. Approximating the solution of surface wave propagation using deep neural networks. In *INNS Big Data and Deep Learning conference*, pages 246–256. Springer, 2019.

[59] Amir Barati Farimani, Joseph Gomes, and Vijay S Pande. Deep learning the physics of transport phenomena. *arXiv preprint arXiv:1709.02432*, 2017.

[60] Sungyong Seo and Yan Liu. Differentiable physics-informed graph networks. *arXiv preprint arXiv:1902.02950*, 2019.

[61] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

[62] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

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

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[65] Hzzone. ConvLSTM implementation. https://github.com/Hzzone/Precipitation-Nowcasting, 2019. [Online; accessed 11-8-2019].

[66] nanton96. Causal LSTM implementation. https://github.com/nanton96/Mobile-Data-Forecasting-With-Spatio-Temporal-Networks, 2019. [Online; accessed 11-8-2019].

[67] usuyama. U-Net implementation. https://github.com/usuyama/pytorch-unet, 2019. [Online; accessed 11-8-2019].

[68] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[69] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[70] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[71] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[72] Christopher Olah. Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs, 2015. [Online; accessed 11-8-2019].

Appendix A

Addendum to backround

A.1 Convolutional Neural Networks

Here we present in detail some of the building blocks for CNNs that are used in this project i.e. convolutions, deconvolutions, pooling and stride for dimensionality reduction and skip connections.

A.1.1 Convolutions

Traditional Multi-Layer Perceptrons use fully connected layers where all the neurons of adjacent layers are connected. MLPs are powerful universal approximators but as layers become wider the number of connections grows exponentially. We can take advantage of the spatial correlations in the pixel values of images by using convolutions. Exploiting locality is achieved by keeping the convolutional kernel sizes relatively small in comparison to the initial image dimensions. Using shared weight for each kernel further reduces the number of parameters and makes training faster. During convolution a kernel is passed over the input channels to produce a feature map as shown in Figure A.1 (left). Formally a convolution C of an input X with a kernel W of size $m \times m$ is defined as:

$$C_{i,j} = \sum_{k=1}^{m} \sum_{l=1}^{m} W_{k,l} X_{i-k,j-l} \tag{A.1}$$

Multiple filters are used in each layer to produce a stack of feature maps. A CNN is built by stacking convolutional layers followed by some non-linearity like sigmoid, tanh or Rectified Linear Units (ReLU) [70]. This way each layer learns a higher-level representation of the features from the previous layer.

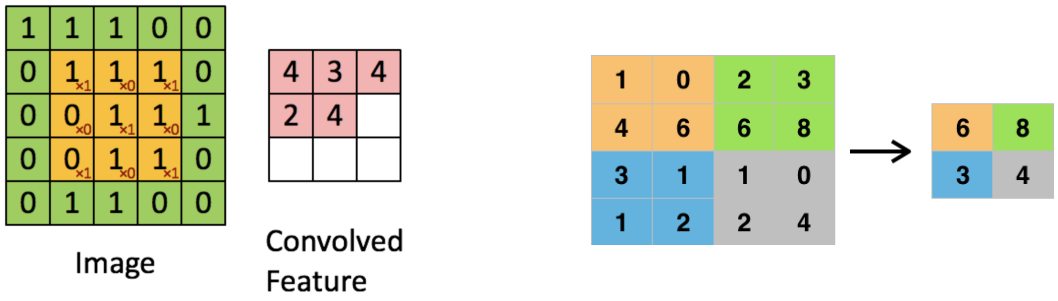


Figure A.1: **Left:** The convolutional kernel (yellow) is applied on a grid (green) to produce a feature map (pink). **Right:** A max pooling operation with a 2×2 kernel and stride 2 reduces the number of pixels to 25% of the original

A.1.2 Receptive field size

Despite the aforementioned advantages, keeping the convolutional kernel size small means that it is hard to capture the interactions between image regions that are further apart. One way to remedy this is by reducing the dimensionality by using a kernel stride bigger than 1 or by using subsampling layers like max-pooling (Figure A.1). This way in deeper layers the original pixel distances become smaller, effectively increasing the receptive field size, and enable the kernels to learn higher-order abstractions of the input. Lower layers extract simple visual features like edges and corners while higher layers learn more complicated patterns. When dimensionality reduction is not desirable, increasing the receptive field size can be attained by using dilations, i.e. the exponential expansion of the kernel size in deeper layers, while keeping the number of kernel weights constant [71]. This approach allows the aggregations of multi-scale contextual information without losing resolution but can also be used in conjunction with dimensionality reduction. **Our experiments show that using dilations improved the performance of our convolutional approach.**

A.1.3 Deconvolutions

Dimensionality reduction induces a bottleneck that is sometimes desired and beneficial. However, in tasks like in video prediction where dimensions must be preserved upsampling methods are used. Deconvolutions also referred to as transposed convolutions, is a common method used for upsampling in CNNs. Deconvolutions act as the opposite of convolutions in the sense that by using a stride larger than one they can increase the dimensions of the feature maps. Other methods without learnable

parameters like bi-linear sampling can also be used.

Deconvolutions have been used in CDNNs (Figures 2.2 and 4.14) and have been proven effective in sea surface temperature forecasting [43], and fluid flow simulation [1]. In this report, we demonstrate that they also perform well in predicting wave propagation.

A.1.4 Skip Connections

Skip connections copy features from earlier layers into higher layers. This helps information traverse faster in deep neural networks and alleviates the issue of vanishing gradients. Another benefit is that they prevent the higher layer from learning the identity transform which means they will perform at least as good as the lower layer. Skip connections were popularized by Residual Networks (ResNet) and significantly help with training very deep neural networks [64].

A.2 Recurrent Neural Networks

A characteristic of intelligence is keeping track and using information that was acquired in the past. This is especially important in sequence modelling because the assumption that the data are independent and identically distributed (i.i.d) does not hold anymore. CNNs are good at capturing spatial appearances they do not explicitly model temporal dynamics. Recurrent neural networks (RNNs) on the other hand, contain a memory of what it has seen so far and uses it to connect past states to future ones. We can formulate that as a transformation ϕ that takes the current input example $\mathbf{x_t}$ and the internal state in the previous time step $\mathbf{h_{t-1}}$ to produce the next $\mathbf{h_t}$:

$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right) \tag{A.2}$$

Information held in the internal state can span many time steps as it cascades forward and influences how new input examples are processed. By sharing weights across time the network can capture long-term dependencies that would otherwise be impossible to do with a feed-forward net. For time-series prediction this is very important because an event in time depends on the events that preceded it.

A.2.1 LSTM

While vanilla RNNs can theoretically learn long-term dependencies, it has been shown that due to vanishing gradients, much of the temporal information is lost after several steps [17]. The authors proposed a variation called Long-Short Term Memory (LSTM) network that apart from the hidden state h_t also poses a cell state c_t which corresponds to longer-term memory. The cell state acts as an accumulator of information and allows effective backpropagation through time (BPTT).

With each new input x_t the hidden and cell state are updated using three gating mechanisms. The forget gate f_t controls how much information we are going to discard from the cell state, the input gate i_t decides what information from the new example is stored in the cell state and the output gate o_t filters how much information from the cell state the network outputs. The information flow within an LSTM cell can be seen in Figure A.2 and is described by the following equations:

$$\begin{aligned} i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i\right) \\ f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f\right) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\ o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \\ h_t &= o_t \circ \tanh\left(c_t\right) \end{aligned} \tag{A.3}$$

where $\circ$ is the Hadamard product. LSTMs have been extensively used in sequence modelling tasks like video prediction [19, 20, 18].

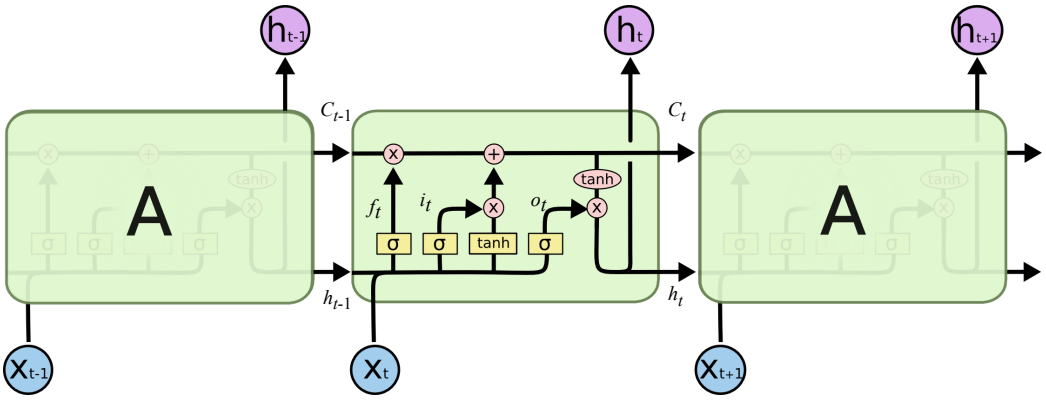


Figure A.2: At each time-step the LSTM cell takes an input X_t and output the internal state h_t using three gating mechanisms: the forget gate f_t, the input gate i_t and the output gate o_t. The internal and cell states h_t, c_t get passed between states. Image adapted from [72]

A.2.2 Convolutional LSTM

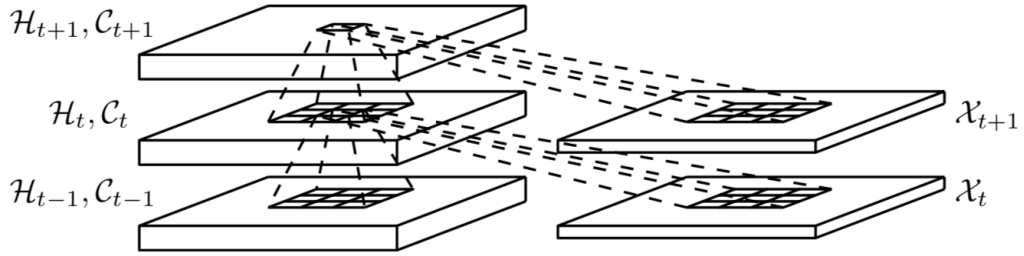


Figure A.3: Convolutionals inside a ConvLSTM model acros time-steps

A.2.3 Causal LSTM

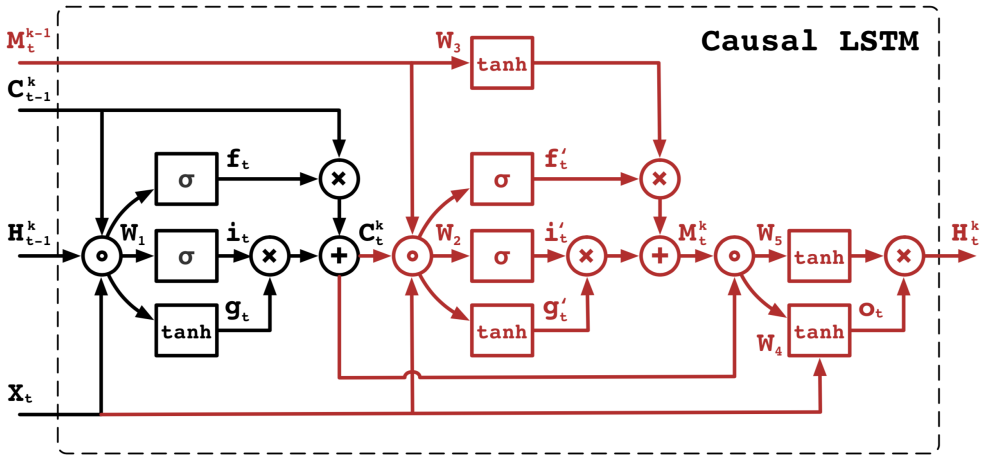


Figure A.4: Causal LSTM

A.3 Multi-step estimation

In spatio-temporal data, the spatial and temporal domains are intertwined in non-trivial ways and multi-step forecasting poses several challenges. The two main categories of multi-step prediction strategies are iterative multi-step (IMS) and direct multi-step (DMS) estimation [18].

In IMS a model is trained to predict the next frame in a sequence. Multi-step predictions are achieved by iteratively feeding the output back to the model. The key

inductive bias here is that the physics of the system remain the same so applying the same operation is enough to propagate information to the next step irrespective of input. IMS has advantages in training because only the next-step error is needed and also in inference, where it is easy to produce forecasts of arbitrary length. The main drawback is that due to the iterative process error accumulation occurs. DMS tries to avoid the error drifting problem by minimizing a long-term prediction error. The drawback is that for each forecast horizon a different model has to be trained. To disentangle the forecast length from the model one can use a single-step predictor recursively, like in IMS, but in this case minimising the multi-step error [19, 21]. DMS is harder to train but more accurate is short-term prediction tasks while IMS is preferred in long-term forecasting. The Markovian assumption, that everything we need to know is included in the current state, is essential to keep the input finite in both approaches. **Our experiments verify that DMS achieves better short-term performance than IMS.**

Appendix B

More charts

B.1 Hyperparameters for best models

The models were trained to minimise the error for the output length. For mode selection though we chose the hyper-parameters that gave the lowest validation error for the next 50 frames. The reason is to make comparisons fair and to avoid overfiting to a small horizon.

Model	**Input Length**	**Output Length**	**Samples per Sequence**	**Batch Size**	**Learning Rate**	**Patience**
LSTM	5	20	10	16	10^{-4}	5
CNN	5	10	5	16	10^{-4}	7
ConvLSTM	5	10	5	8	10^{-3}	7
Causal LSTM	5	20	5	4	10^{-4}	3
CDNN	5	20	10	16	10^{-4}	7

Table B.1: Hyper-parameters of the best performing model for each architecture

B.2 Model size and training times

All models were trained for both 16h and 24h because some models exhibited over-fiting while other did not. As with other hyper-parameters we chose the training time that gave the lowest validation error for the next 50 frames. Total training time includes evaluation overhead.

Model	Trainable Parameters	Epoch Time	Num Epochs	Best Epoch	Total Training Time
LSTM	88.2M	12m	75	71	16h
CNN	0.8M	8m	185	181	24h
ConvLSTM	12.3M	36m	24	18	16h
Causal LSTM	2.5M	33m	43	36	24h
CDNN	7.8M	8m	171	166	24h

Table B.2: Model size and training times

B.3 Test set performance

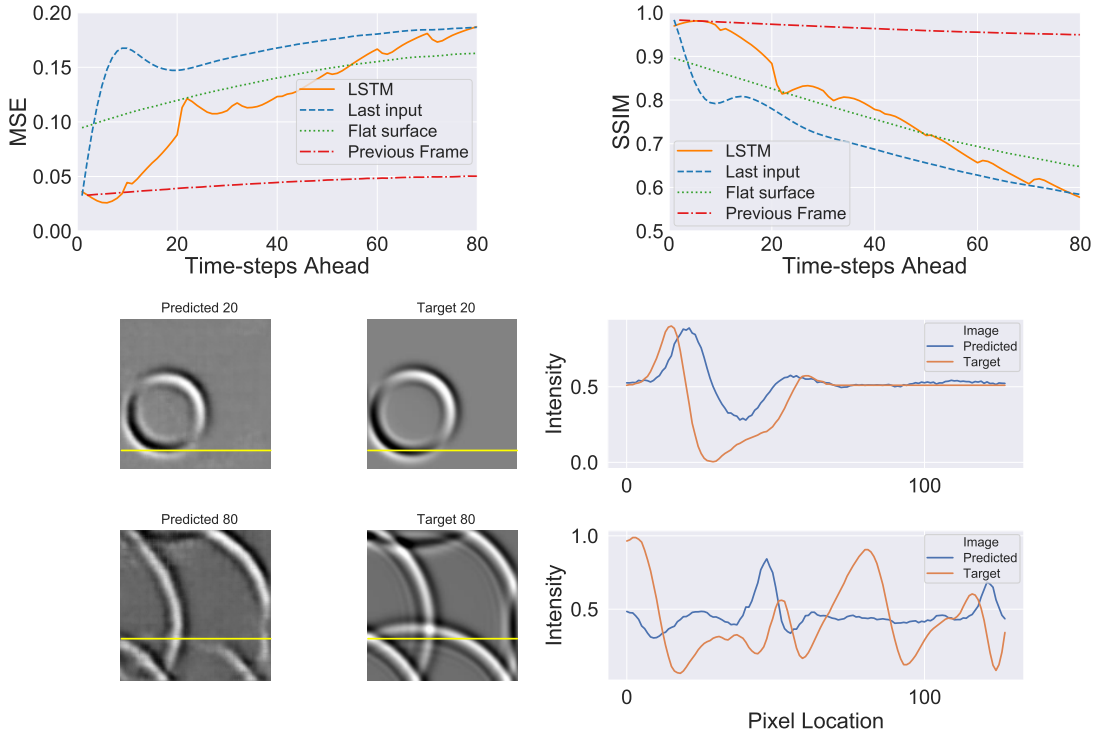


Figure B.1: **LSTM model performance on test set** - **Upper Left:** MSE **Upper Right:** SSIM **Center & Bottom:** Intensity profiles for predictions t=20 and t=80

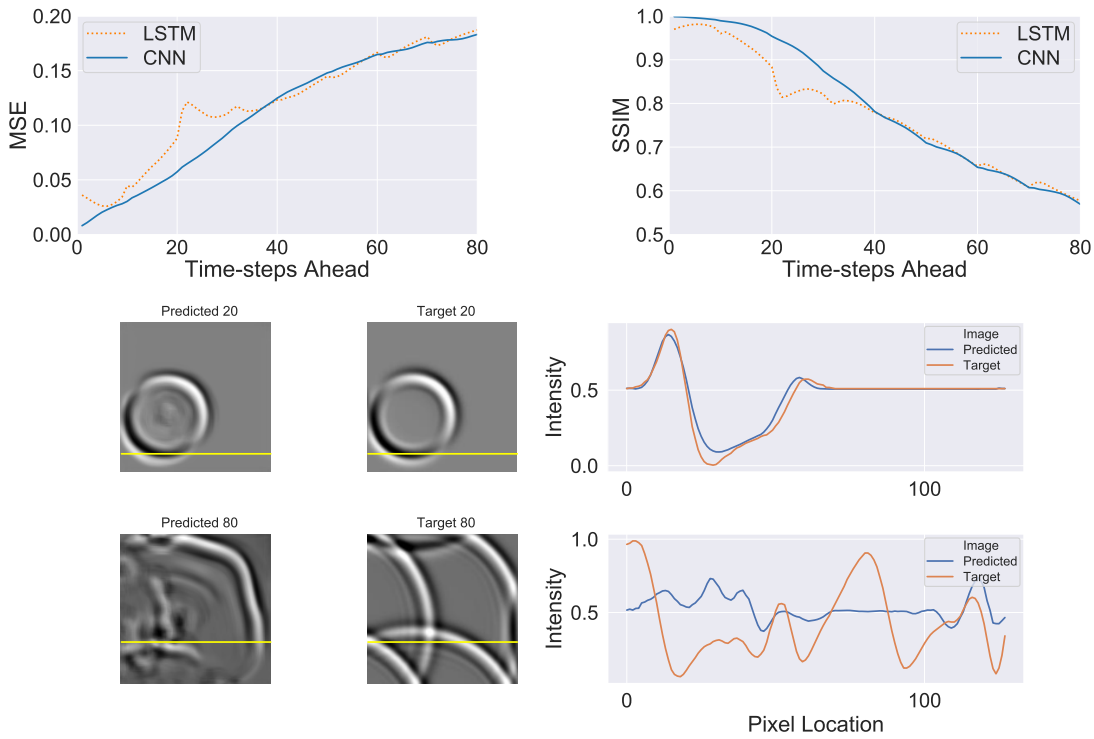


Figure B.2: **CNN model performance on test set** - **Upper Left:** MSE **Upper Right:** SSIM **Center & Bottom:** Intensity profiles for predictions t=20 and t=8

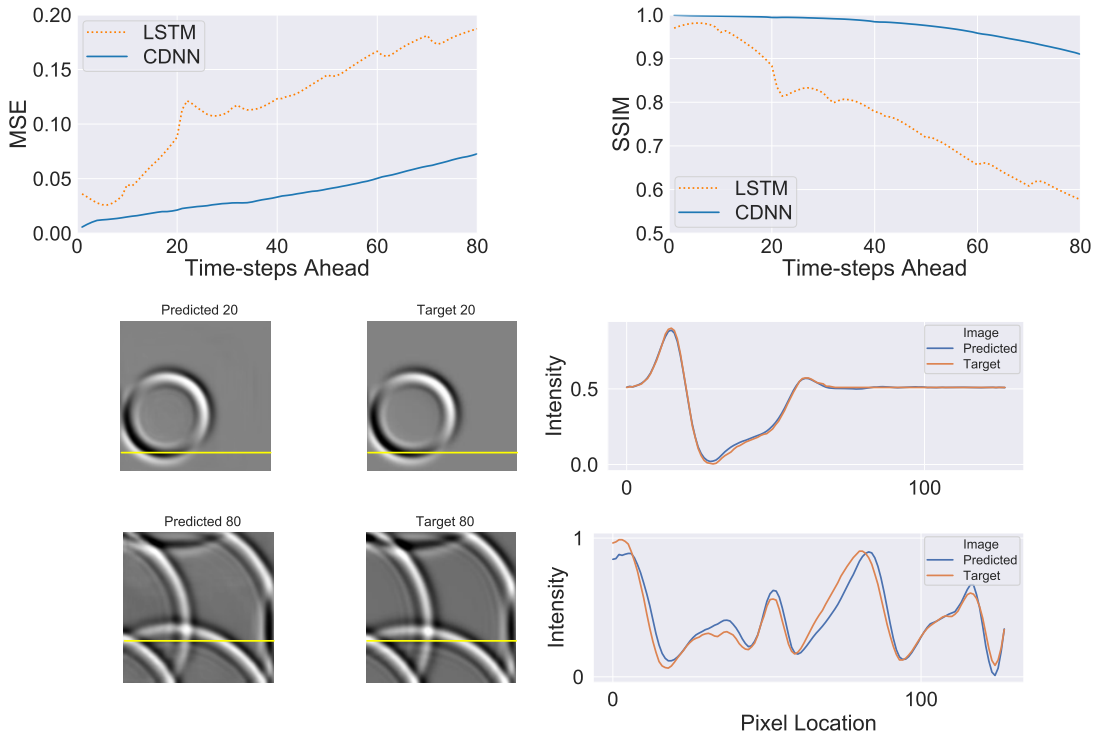


Figure B.3: **CDNN model performance on test set** - **Upper Left:** MSE **Upper Right:** SSIM **Center & Bottom:** Intensity profiles for predictions t=20 and t=8

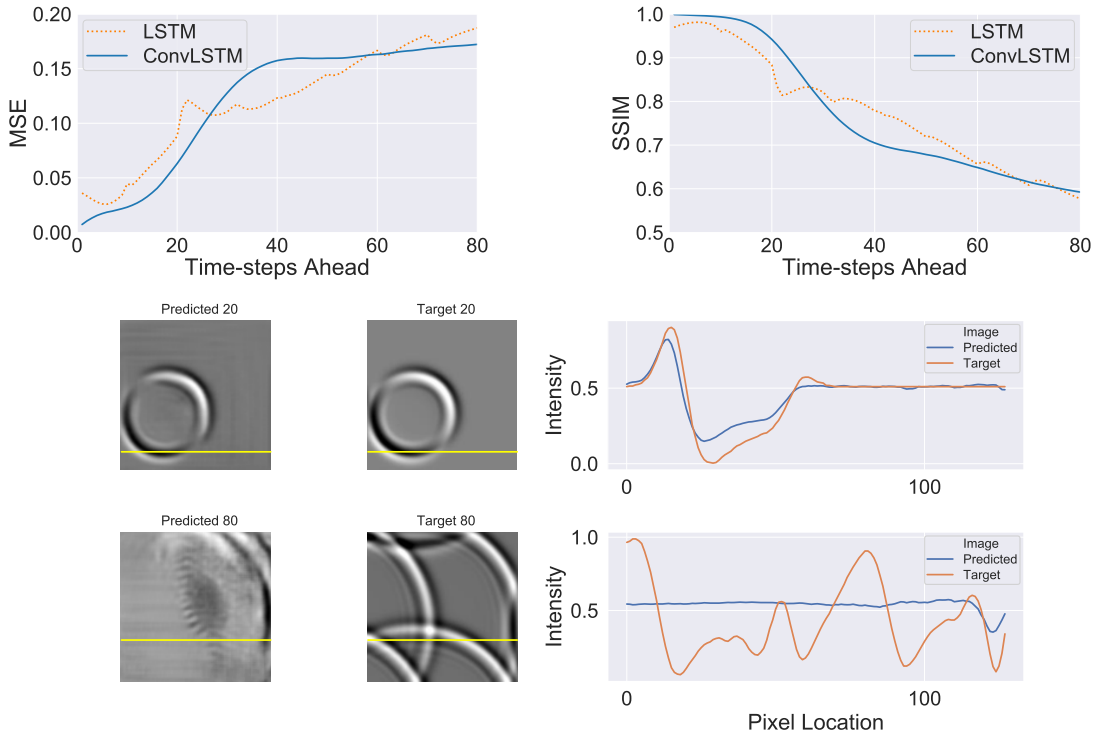


Figure B.4: **ConvLSTM model performance on test set** - **Upper Left:** MSE **Upper Right:** SSIM **Center & Bottom:** Intensity profiles for predictions t=20 and t=8

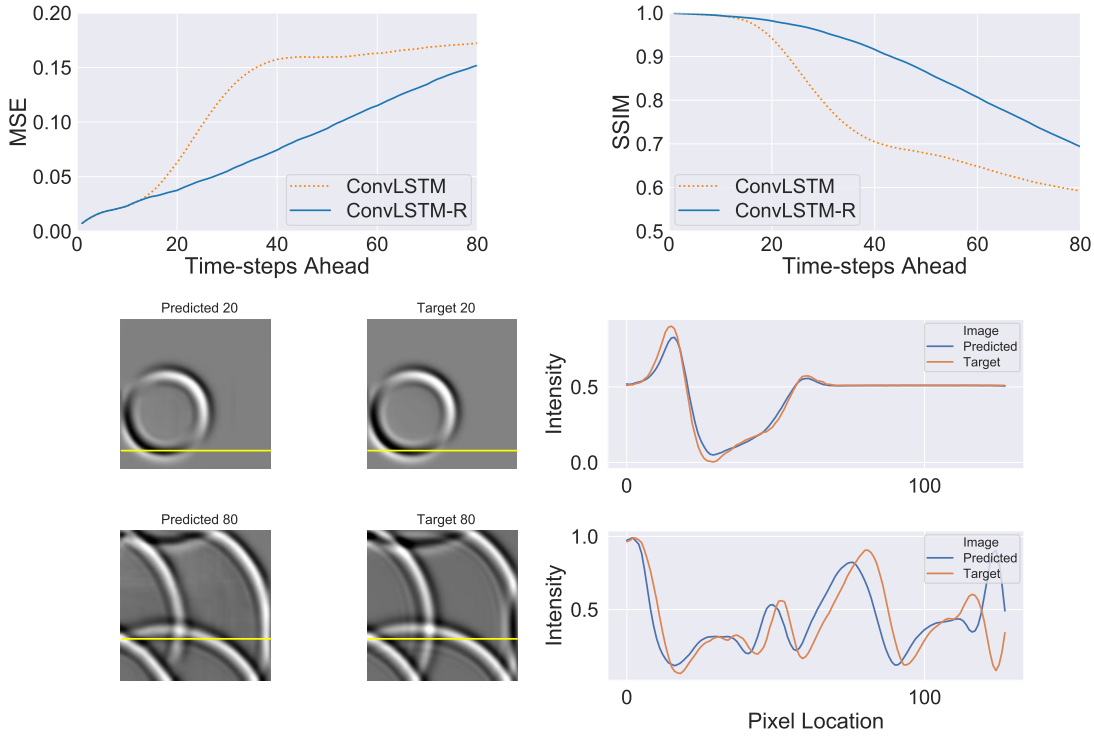


Figure B.5: **ConvLSTM-R model performance on test set** - **Upper Left:** MSE **Upper Right:** SSIM **Center & Bottom:** Intensity profiles for predictions t=20 and t=8

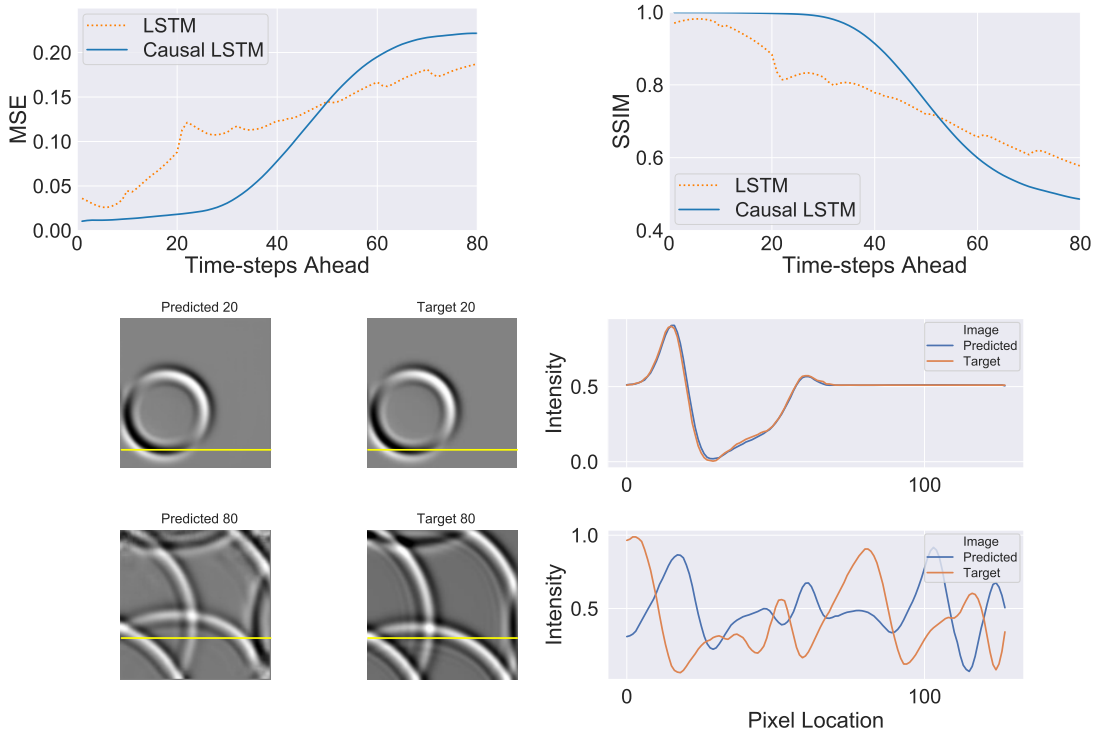


Figure B.6: **Causal LSTM model performance on test set** - **Upper Left:** MSE **Upper Right:** SSIM **Center & Bottom:** Intensity profiles for predictions t=20 and t=8

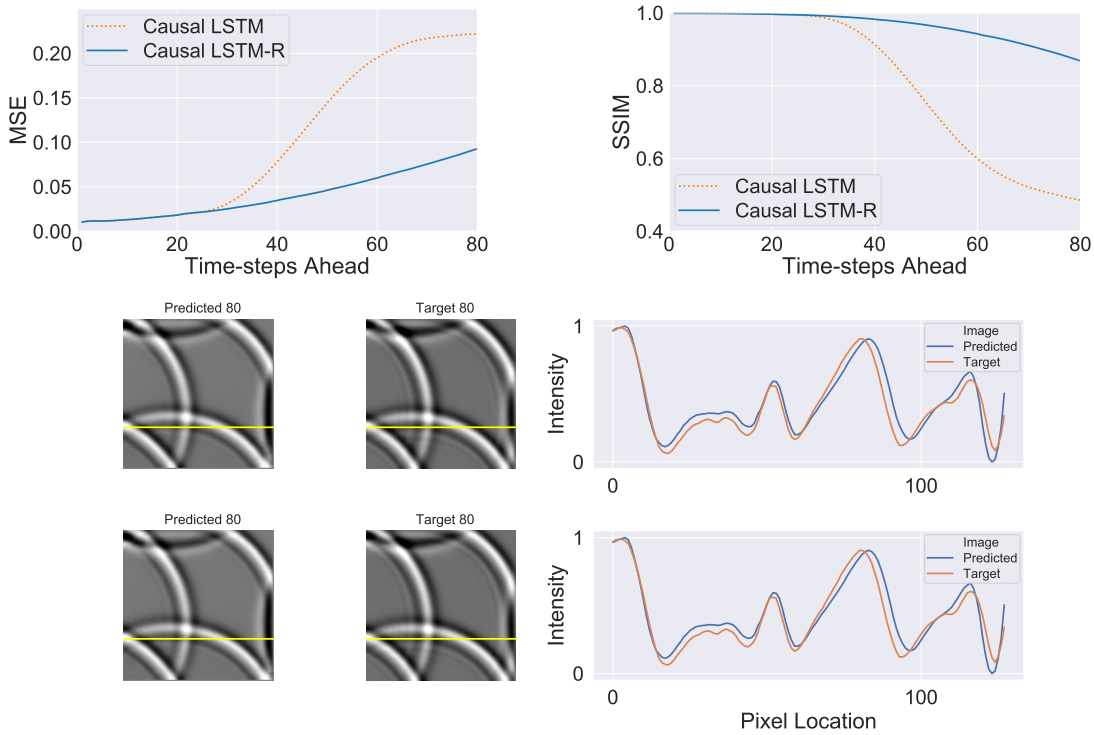
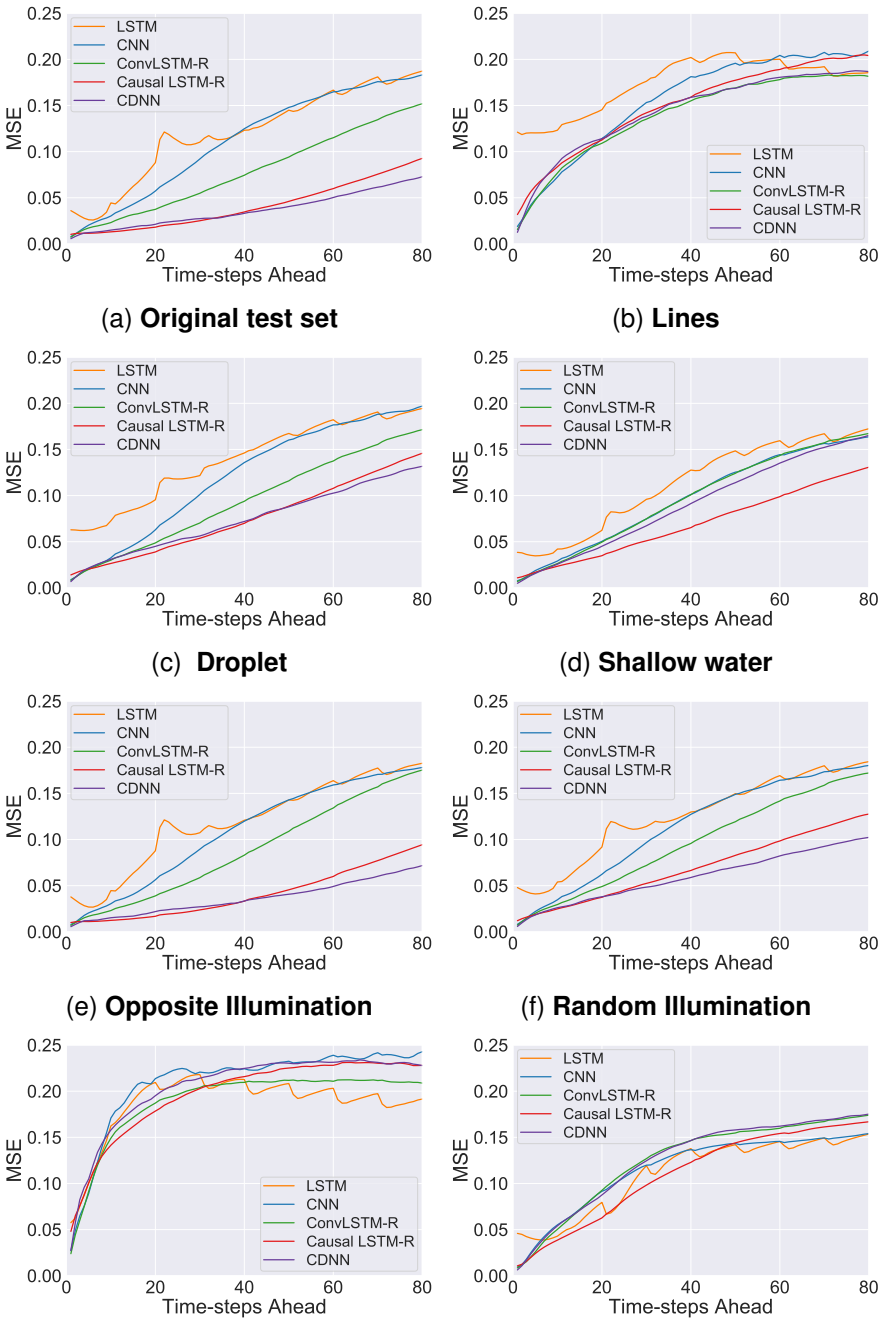


Figure B.7: **Causal LSTM-R model performance on test set** - **Upper Left:** MSE **Upper Right:** SSIM **Center & Bottom:** Intensity profiles for predictions t=20 and t=8

B.4 Generalization performance



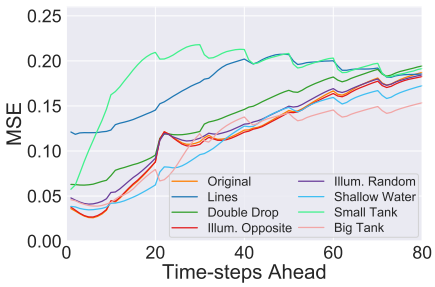
(a) **Original test set** (b) **Lines**

(c) **Droplet** (d) **Shallow water**

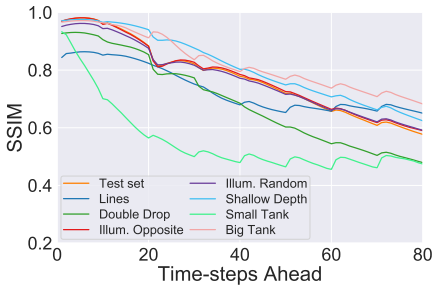
(e) **Opposite Illumination** (f) **Random Illumination**

(g) **Smaller Tank** (h) **Bigger Tank**

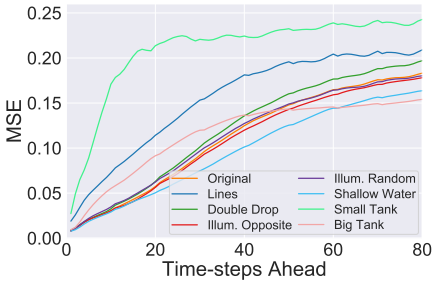
Figure B.8: Each charts shows how the networks perform in one datasets



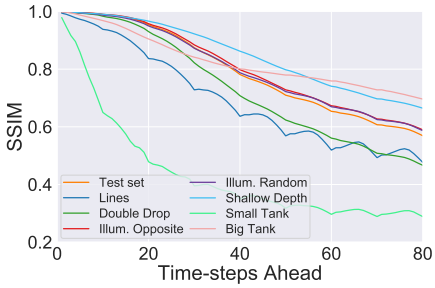
(a) **LSTM** MSE for all datasets



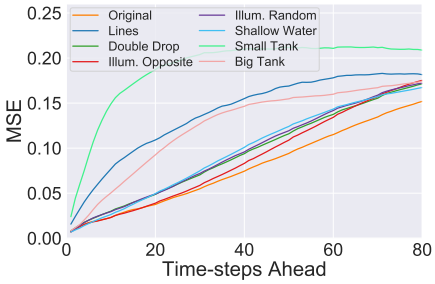
(b) **LSTM** SSIM for all datasets



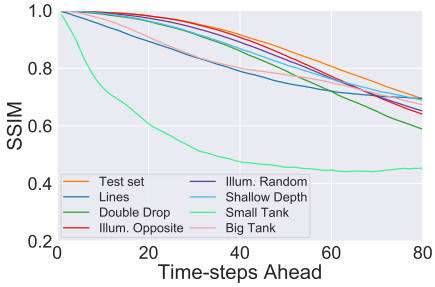
(c) **CNN** MSE for all datasets



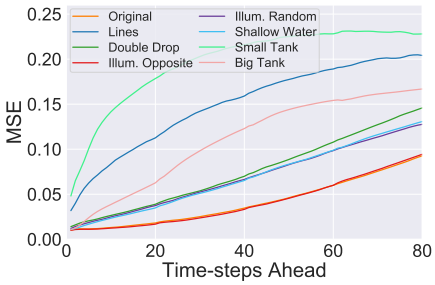
(d) **CNN** SSIM for all datasets



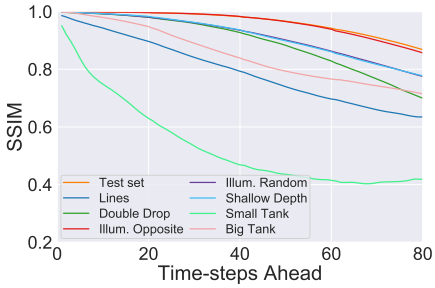
(e) **ConvLSTM-R** MSE for all datasets



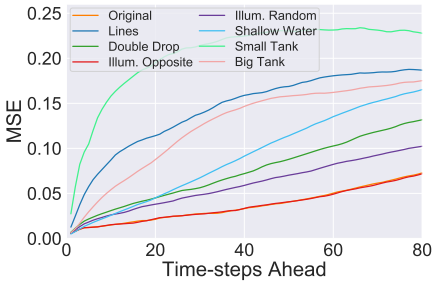
(f) **ConvLSTM-R SSIM** for all datasets



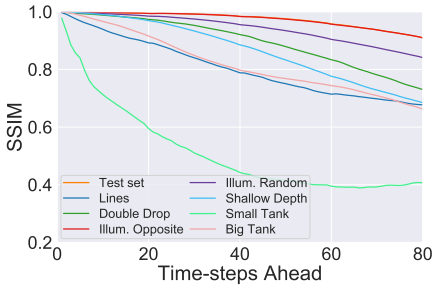
(g) **Causal LSTM-R** MSE for all datasets



(h) **Causal LSTM-R** SSIM for all datasets



(i) **Causal LSTM-R** SSIM for all datasets



(j) **Causal LSTM-R** SSIM for all datasets

Figure B.9: Each charts shows how one networks performs across all datasets

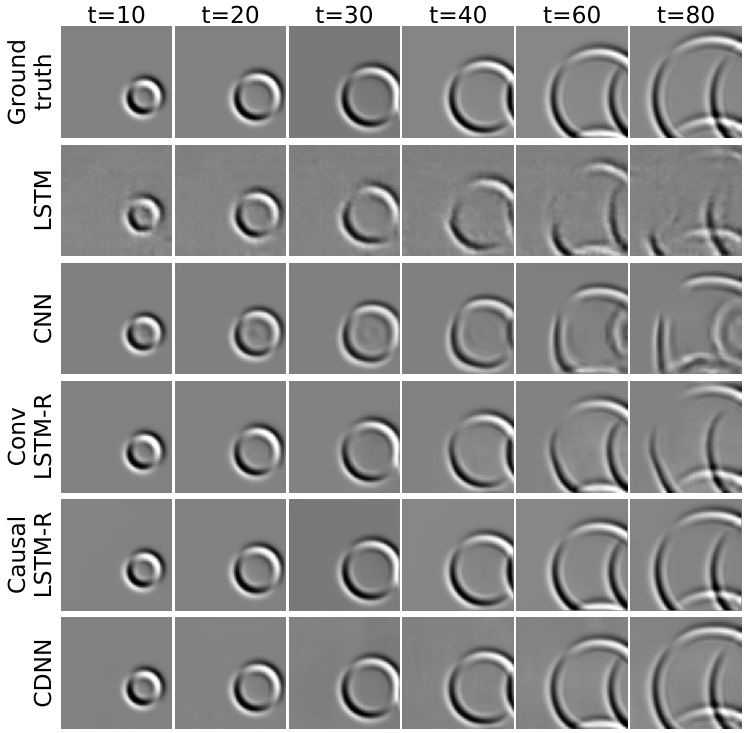


Figure B.10: **Original Test Set**

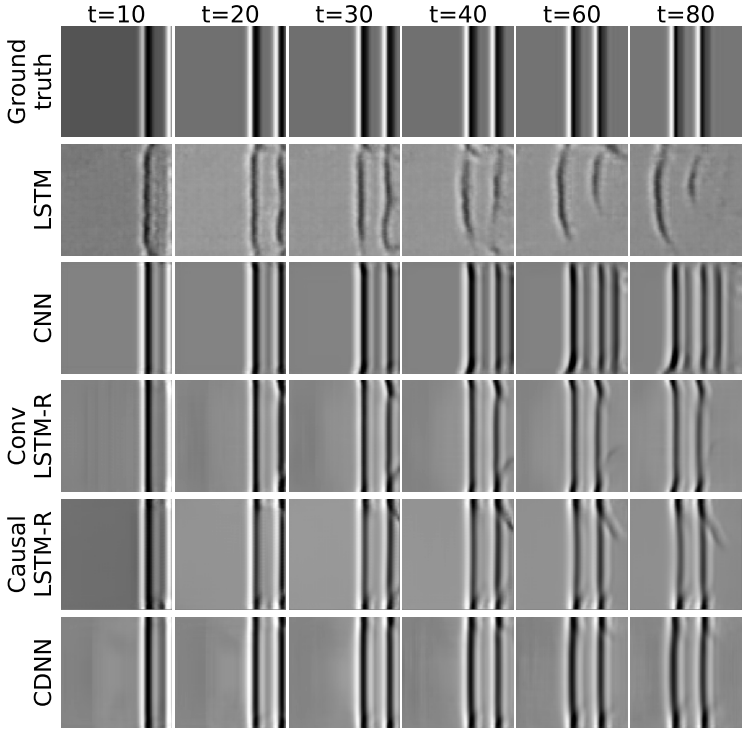


Figure B.11: **Lines**

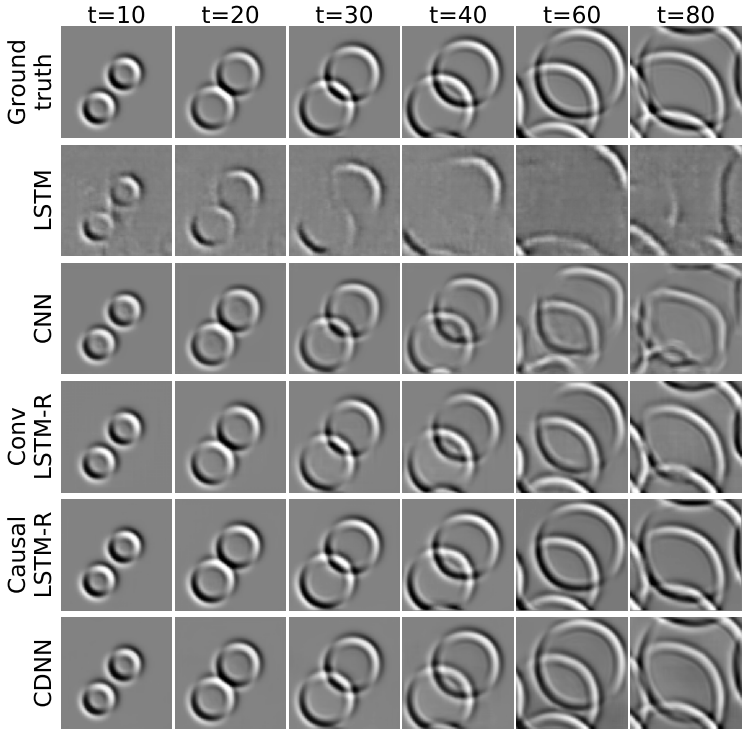


Figure B.12: **Double Drop**

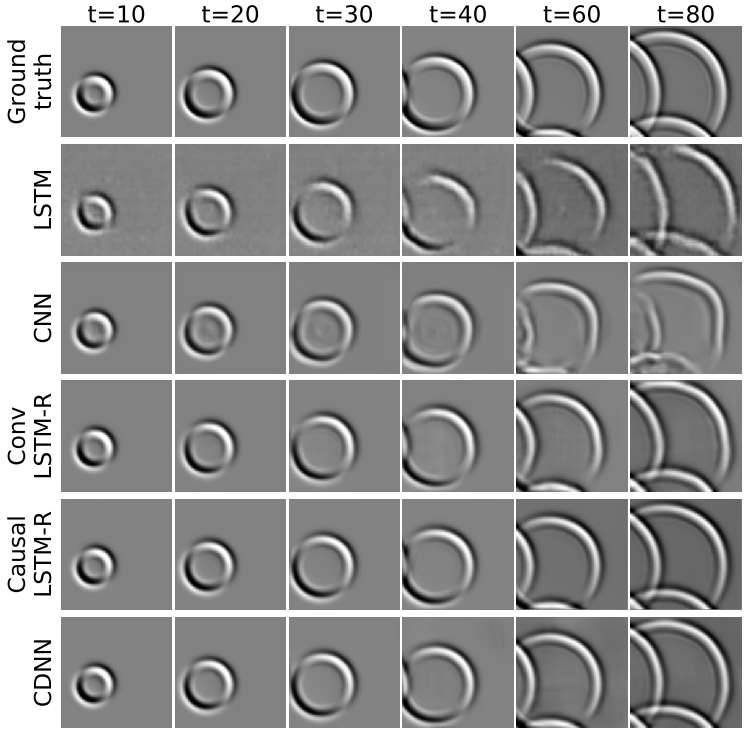


Figure B.13: **Shallow Depth**

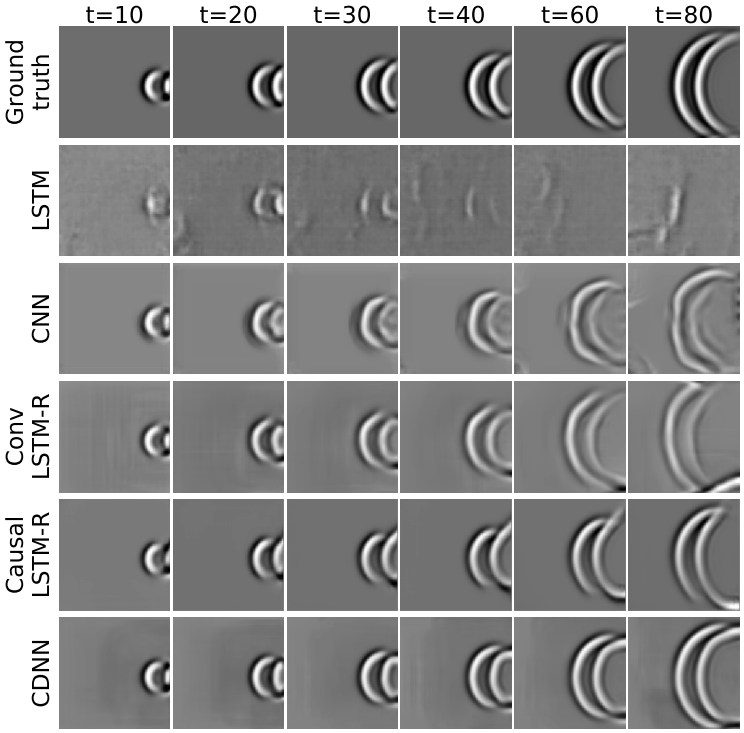


Figure B.14: **Random Illumination**

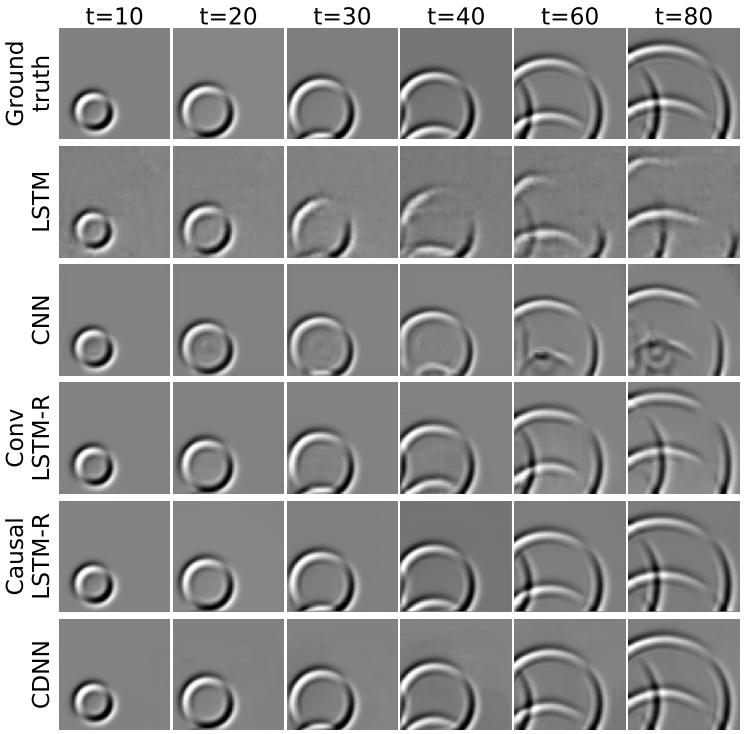


Figure B.15: **Opposite Illumination**

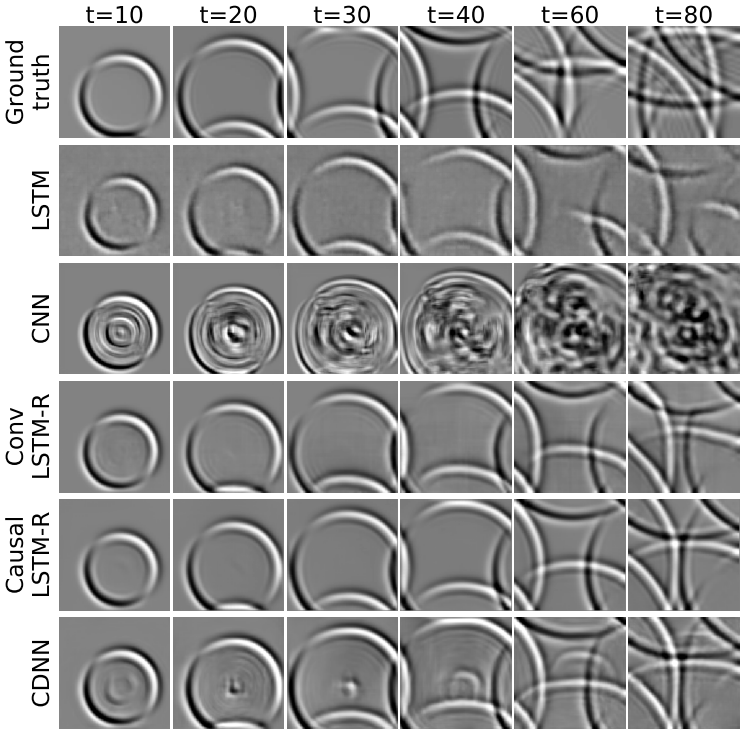


Figure B.16: **Smaller water tank**

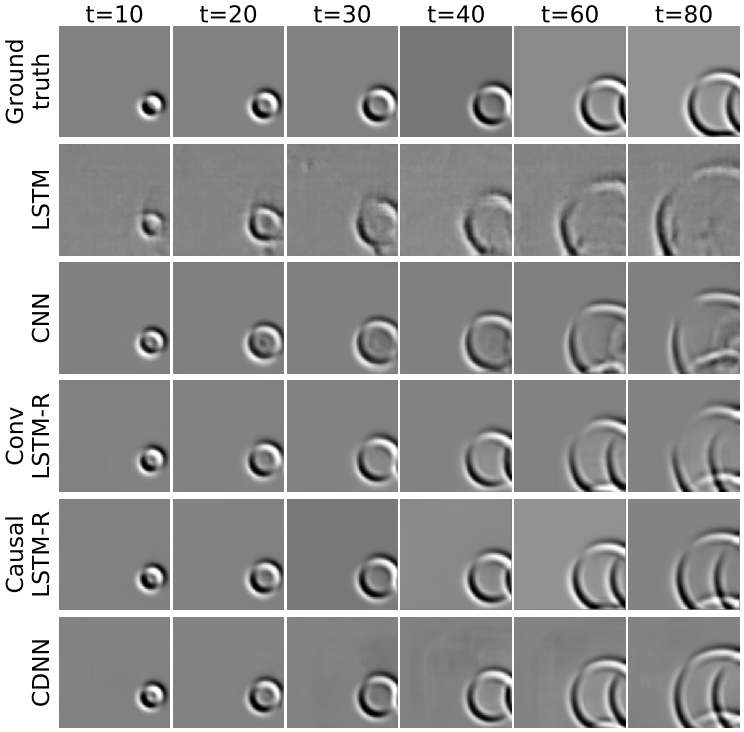


Figure B.17: **Bigger water tank**