FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Improving short-term rain prediction by using deep neural networks with advanced architecture
Student:	Filip Miškařík
Supervisor:	Mgr. Petr Šimánek
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

Improving short-term prediction of rain is a very important and challenging task. We currently use a method called PhyDNet [1] which provides a very good solution, but we want to improve it further. The PhyDNet architecture currently uses a very basic ConvLSTM model as a background. The task is to understand if we can improve PhyDNet by improving the background model.

1) Survey and understand methods for Spatio-temporal prediction using artificial neural networks
(e.g. ConvLSTm, predNet, predRNN, predRNN++, phyDNet).
2) Explore and describe the weather radar dataset (provided by Meteopress).
3) Implement one chosen method (e.g. predRNN++) into phyDNet architecture, in Pytorch.
4) Train the new model on radar data.
5) Compare the model performance with original phyDNet and analyze the results.

Literatura:
[1] Disentangling Physical Dynamics from Unknown Factors for Unsupervised Video Prediction, Le Guen et al.
[2] PredRNN: A Recurrent Neural Network for Spatiotemporal Predictive Learning, Y. Wang, et al.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 8 November 2022 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Improving short-term rain prediction by using deep neural networks with advanced architecture

Filip Miškařík

Department of Applied Mathematics
Supervisor: Mgr. Petr Šimánek

May 10, 2023

Acknowledgements

I would like to thank Mgr. Petr Šimánek for his guidance, advice and suggestions. I would also like to thank the company Meteopress and Datalab of FIT CTU for providing the dataset and computational power, which made this thesis possible.

Declaration

In Prague on May 10, 2023 . . . .

Citation of this thesis

Miškařík, Filip. *Improving short-term rain prediction by using deep neural networks with advanced architecture.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstrakt

Nowcasting srážek si klade za cíl poskytovat přesné krátkodobé předpovědi srážek pro určitou oblast. V posledních letech se k takovým předpovědím čím dál častěji využívají hluboké neuronové sítě. Zaměřením této práce je vylepšení jedné z těchto sítí, PhyDNet, sestávající se z dvouvětvé architektury, která rozplétá fyzikální dynamiku bouřek od ostatních informací.

V jedné z větví PhyDNetu se v současnosti využívá jednoduchý model pro obecné předpovídání snímků videa, ConvLSTM. V této práci provádíme několik úprav PhyDNetu, od malých změn současné architektury až po experimentování s využitím jiných modelů místo ConvLSTM, specificky SA-ConvLSTM a PredRNN.

Tato práce přináší dvě perspektivní modifikace – nahrazení ConvLSTM za komplexnější model PredRNN a přidání zkratky (tzv. skip connection) do větve zodpovědné za modelování fyzikální dynamiky. Naše experimenty ukazují, že tyto změny přináší lepší výsledky oproti původnímu modelu, obzvláště pro dlouhodobější předpovědi.

Klíčová slova časoprostorová predikce, předpovídání srážek, hluboké učení, konvoluční neuronová síť, rekurentní neuronová síť, PyTorch

Abstract

The goal of precipitation nowcasting is to give a precise short-term prediction of rainfall intensity in a local region. In recent years, this problem has seen wide adoption by many deep learning models for spatiotemporal prediction. This thesis focuses on improving one such model, PhyDNet, which disentangles physical dynamics from other, unknown, information in a two-branch architecture.

The branch for capturing the unknown phenomena uses a basic ConvLSTM model for general video prediction. In this thesis, we apply several modifications to the PhyDNet, ranging from small adjustments to the current architecture to experiments with swapping the ConvLSTM for different models, namely SA-ConvLSTM and PredRNN.

We present two promising alterations – replacing ConvLSTM with a more complex PredRNN model and adding a skip connection to the branch that models physical dynamics. The results of our experiments show that these modifications can outperform the original network, especially for more long-term predictions.

Keywords spatiotemporal prediction, precipitation nowcasting, deep learning, convolutional neural network, recurrent neural network, PyTorch

Contents

List of Figures

Introduction

In recent years, deep learning has emerged as a cutting-edge solution to many different types of problems. One of the problems where artificial neural networks have received growing interest is spatiotemporal prediction – the task of predicting sequences of data based on their variations in both time and space. An important application of spatiotemporal prediction is short-term rain prediction, also known as precipitation nowcasting, which is the main focus of this thesis.

Precipitation nowcasting is a significant problem in the field of meteorology, which aims to give precise short-range (usually 0-6 hours) forecast of rainfall intensity in a local region based on radar echo maps or using numerical weather prediction. It has a substantial impact on our daily lives as it provides airports with weather guidance, helps predict road conditions, and aids with issuing emergency rainfall alerts, potentially saving many lives. However, with the very complex nature of atmospheric dynamics and increasing demand for real-time, accurate, and timely prediction, precipitation nowcasting presents a significant challenge. [1, 2]

Many machine learning models have been specifically designed or successfully used for precipitation nowcasting. They often consist of convolutional and recurrent neural networks that predict rainfall from sequences of radar images. The PhyDNet model takes this a step further and combines the power of these neural networks with knowledge of physical laws. The resulting architecture contains two major subnetworks, one for modeling the physics and the other for capturing the unknown factors through a recurrent convolutional network.

Although the results of PhyDNet for precipitation nowcasting are promising, there is room for improvement. The recurrent convolutional subnetwork uses ConvLSTM, a relatively simple model for general spatiotemporal video prediction. This thesis aims to improve this part of PhyDNet, either through various modifications of the current ConvLSTM model or by replacing it with a more novel and complex model.

This thesis is divided into five chapters. The first chapter provides theoretical deep learning background and introduces the reader to concepts important for the thesis. Chapter 2 then describes various models for spatiotemporal prediction and explains their architecture. Chapter 3 defines precipitation nowcasting and outlines some methods used for the task. Chapter 4 describes the radar echo dataset and outlines the details of the training procedure. In chapter 5, all of the conducted experiments are described and evaluated and their results are summarized, compared, and discussed.

Chapter 1

Machine learning background

This thesis deals with the application of neural networks for predicting the weather. It is assumed that the reader is familiar with basic principles of neural networks and how they are trained. This chapter describes select concepts in neural networks and deep learning important for this thesis. It also explains more advanced topics in deep learning, like convolutional and recurrent neural networks, or attention.

1.1 Deep learning

The core building blocks of neural networks are layers consisting of individual neurons, each computing some function of its inputs. Neural networks can model very complex functions by stacking multiple such layers on top of each other. The values of the input to network are referred to as the *input layer*, while the last layer, which represents a final score or decision, is called the *output layer*. All intermediate layers are collectively known as the *hidden layers*. Deep learning refers to the practice of using neural networks with many hidden layers.

1.1.1 Fully-connected layer

A *fully-connected layer* (also called a *dense* layer) is a basic type of layer used in neural networks. It is made up of a vector of neurons with every neuron connected to all p neurons in the previous layer. Each neuron calculates

$$z = \sum_{i=1}^{p} w_i x_i + b = w^T x + b, \tag{1.1}$$

where $x = (x_1, \ldots, x_p)^T$ are the inputs, $w = (w_1, \ldots, w_p)^T$ are the weights and b is the bias. The output of the entire layer with n neurons can be written in matrix form as

$$\mathbf{z} = W^T x + b \tag{1.2}$$

where i-th column of $W \in \mathbb{R}^{p \times n}$ and i-th element of $b \in \mathbb{R}^n$ are the weights and bias of i-th neuron in the layer.

z or $\mathbf{z}$ are subsequently passed to an activation function f, such as the sigmoid, hyperbolic tangent, ReLU or Leaky ReLU, to form the output $\hat{Y} = f(z)$.

Nowadays, many neural networks use the fully-connected layers in tandem with other types of layers, like convolutional and pooling layers (discussed in sections 1.2.1 and 1.2.2), to form more complex architectures.

1.1.2 Adam

Neural networks are trained to minimize a function $J(w)$, which represents the average prediction error measured with loss function $\mathcal{L}$, given the network parameters w. The function is minimized using gradient descent, which calculates the gradient $\nabla_w J$ and then updates the parameters as $w \leftarrow w - \eta \nabla_w J$, moving in the opposite direction of the gradient. [3]

The hyperparameter η (called the *learning rate*) determines how much the parameters are updated. Instead of traditional gradient descent, which maintains a single unchanging learning rate, modern networks use optimization algorithms with multiple adaptive learning rates, such as RMSprop, AdaGrad, or Adam. The network implemented in this thesis uses the Adam algorithm, which computes individual adaptive learning rates for each parameter from estimates of the first (mean) and second (uncentered variance) moments of the gradients. [4]

1.1.3 Vanishing gradient problem

Training deep neural networks presents a significant challenge because of the *vanishing gradient problem.* When the weights are updated, the gradient moves back through the network, often getting smaller with each step. This prevents the weights from updating their values and causes the neurons in earlier layers to learn at a much slower pace than neurons in later layers.

Because the gradient is calculated using the chain rule, the gradient in the early layers is the product of terms from all the later layers. When combined with the fact that some activation functions like the sigmoid have derivatives with values in a range between 0 and 1, the multiplication of n such small numbers causes the gradient to vanish. However, the opposite situation, where the gradient gets progressively bigger, can also occur and is called the *exploding gradient problem.* [3]

1.1.4 Normalization

Another issue, which makes training deep networks difficult, is the fact that the inputs of each layer depend on the parameters of previous layers. This means that small changes to the network's parameters amplify as the network becomes deeper. This phenomenon, known as *internal covariate shift*, slows down training by requiring lower learning rates and careful parameter initialization. [5] A solution to this problem is to normalize the inputs through one of the following methods:

Batch Normalization normalizes the means and variances of the inputs to a layer for each training mini-batch. It significantly accelerates the training and also helps the gradient to flow through the network by reducing the dependence of gradients on the scale of the parameters or their initial values. [5] However, the effect of batch normalization is highly dependent on the mini-batch size with the error rapidly increasing with decreasing batch size. [6, 7]

Layer Normalization computes the means and variances from the sum of all inputs to a layer on a single training case. It is also highly effective at stabilizing the hidden state dynamics in recurrent neural networks. [6]

Group Normalization divides the channels of the input into groups and computes the mean and variance for normalization within each group. This makes the normalization independent of the mini-batch size. [7]

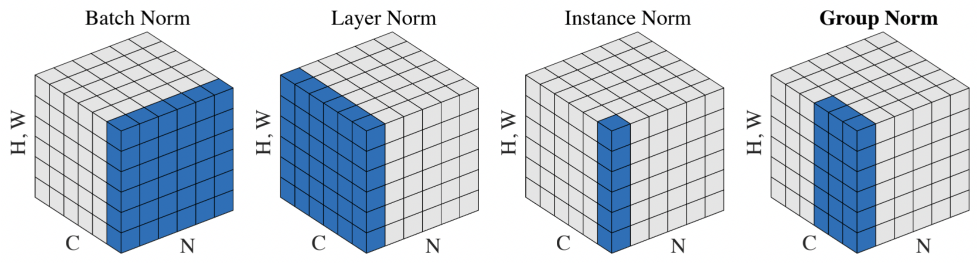

Figure 1.1: Comparison of normalization techniques on an input tensor. N is the batch size, C the number of channels and (H, W) the height and width. The blue area is normalized by the same and variance, computed by aggregating the values of these pixels. [7]

1.2 Convolutional neural networks

Convolutional neural networks (CNNs) are specialized kinds of neural networks designed mainly for working with images (although they have numerous other applications, such as audio, text, or time series analysis). [8] Compared to traditional, fully-connected networks, they present two significant advantages. Firstly, CNNs can take into account the spatial structure of the image and extract spatial features without explicit engineering. Secondly, they scale much better with the size of the images. For example, in a fully-connected network with a color input image of size 600×600, each fully-connected neuron in the first hidden layer would have over a million parameters. In convolutional networks, each point of the output is connected to only a small area of the input, which vastly reduces the number of parameters in the network and makes the computations much more efficient. [9, 3]

A typical convolutional network is made up of an alternating pattern of convolutional and pooling layers. The convolutional layers are usually immediately followed by an activation function like ReLU. Fully-connected layers are commonly found at the end of the network, with the last fully-connected layer computing the output, i. e. the classification score. [9]

1.2.1 Convolutional layers

Convolutional layers are the main building blocks of CNNs. They consist of a set of learnable 3D filters of shape $C \times k \times k$, which process 3D input tensors of shape $C \times H \times W$, where C is the number of image channels, H and W are the height and width of the image and k is the size of the filter. These filters slide across the input image and calculate cross-correlation at each position in the image (shown in Figure 1.2). This produces a set of 2D tensors called *activation maps*. These activation maps are then stacked along the channel dimension to obtain a 3D output. The network will usually learn filters that activate upon detecting a particular visual feature, for example, an edge or a specific color. [10, 9]

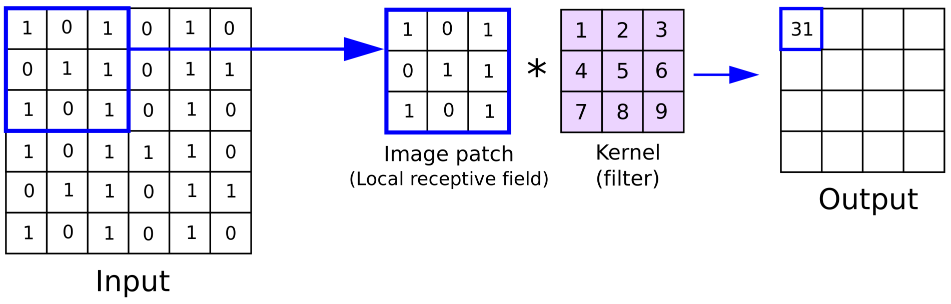

Figure 1.2: Example of a convolutional layer with a 3×3 kernel [11]

1.2.2 Pooling layers

Pooling layers are used to progressively reduce the spatial size (also known as *downsampling*) of the images, thereby diminishing the number of parameters, which leads to faster computations and helps control overfitting. This is done by replacing the outputs at a specific location with a summary statistic of the nearby pixels. A commonly used technique is *max pooling*, which outputs the maximum value within a rectangular neighborhood. Other pooling functions include average pooling and L^2 norm pooling, both of which have seen less use in recent years compared to max pooling. The pooling function is applied to a $k \times k$ filter, which slides across the image and downsamples each slice independently to a single value (see Figure 1.3). [10, 9]

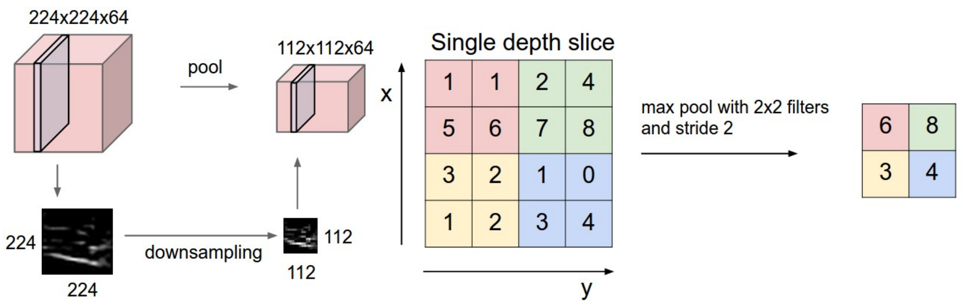

Figure 1.3: Example of max pooling using a 2×2 filter with stride 2. The maximum value of the 2×2 square is outputted, resulting in an image with half the height and width of the original and only 25% of the values. [9]

An opposite operation to pooling, *upsampling*, is used for increasing the resolution of the image. It is often found in convolutional encoder-decoder networks (section 1.3.3), where upsampling is used to scale outputs of the downsampling layers back to the original size of the input images. [12]

1.3 Recurrent neural networks

In many learning tasks, e.g. text prediction, music generation, or video analysis, predictions depend not only on the current input but also on the previous inputs. However, feed-forward or convolutional neural networks can only process one input at a time, which makes them unsuitable for such tasks. Recurrent neural networks (RNNs) address this issue by processing sequences of data via recurrent connections, which can be thought of as loops in the network that allow information to persist. [13] As shown in Figure 1.4, these loops can be unfolded into a sequence of networks that share some of the parameters across time steps. To achieve this, RNNs introduce a *hidden state* containing a summary of the information from previous inputs that transfers the memorized features to the next time step. [8, 10]

1.3.1 LSTM

While RNNs can access information from previous inputs, this ability diminishes when going further into the past because of the vanishing gradient problem. A widely-used solution to this problem are Long Short-Term Memory

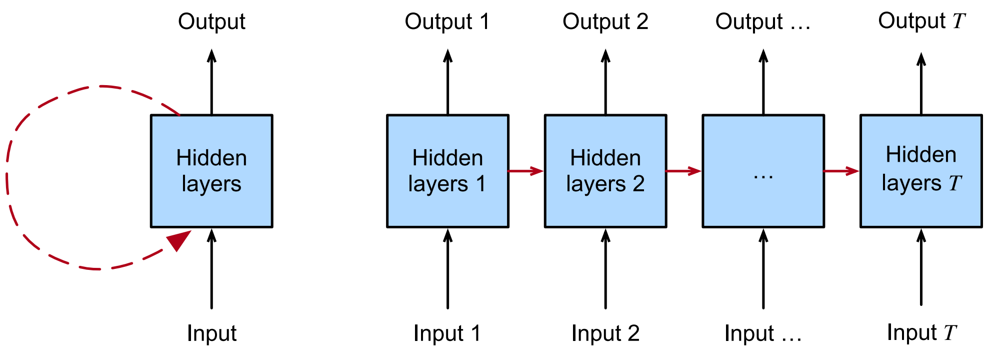

Figure 1.4: Recurrent connections can be represented by cycles in the network (left) or as an unfolded series of networks with recurrent edges connecting the network across adjacent time steps (right). [8]

(LSTM) networks, which are capable of learning long-term dependencies. In addition to the hidden state, they contain a memory cell $\mathcal{C}$ for passing long-term information and possess a more intricate inner structure that allows to add or remove information from the memory cell state. [13, 8]

Inside the LSTM, there are three multiplicative *gates*, which determine whether and how the inputs should affect the memory cell state and the output of the unit. The *forget gate* (f_t) controls which information is kept and which is discarded, the *input gate* (i_t) decides which parts of the input will be added to the memory cell, and the *output gate* (o_t) determines which information will be outputted at the current time step. [8]

The network takes both the input at the current time step X_t and the hidden state at the previous time step $\mathcal{H}_{t-1}$ as its input and passes it to the gates, which are comprised of fully-connected layers with activation functions and element-wise operations. Collectively, these can be described by the following set of equations, where W and b are the weight and bias parameters and $\odot$ is the element-wise (Hadamard) product:

$$\begin{aligned}
f_t &= \sigma\left(W_{xf}X_t + W_{hf}\mathcal{H}_{t-1} + b_f\right)\\
i_t &= \sigma\left(W_{xi}X_t + W_{hi}\mathcal{H}_{t-1} + b_i\right)\\
o_t &= \sigma\left(W_{xo}X_t + W_{ho}\mathcal{H}_{t-1} + b_o\right)\\
\tilde{\mathcal{C}}_t &= \tanh\left(W_{xc}X_t + W_{hc}\mathcal{H}_{t-1} + b_c\right)\\
\mathcal{C}_t &= f_t \odot \mathcal{C}_{t-1} + i_t \odot \tilde{\mathcal{C}}_t\\
\mathcal{H}_t &= o_t \odot \tanh\left(\mathcal{C}_t\right)
\end{aligned} \tag{1.3}$$

Figure 1.5 shows a visual representation of these calculations. [8]

There are many modifications of the LSTM model, a notable example being the Gated Recurrent Unit (GRU). Another variation suited for spatiotemporal prediction, ConvLSTM, is described later in section 2.2.

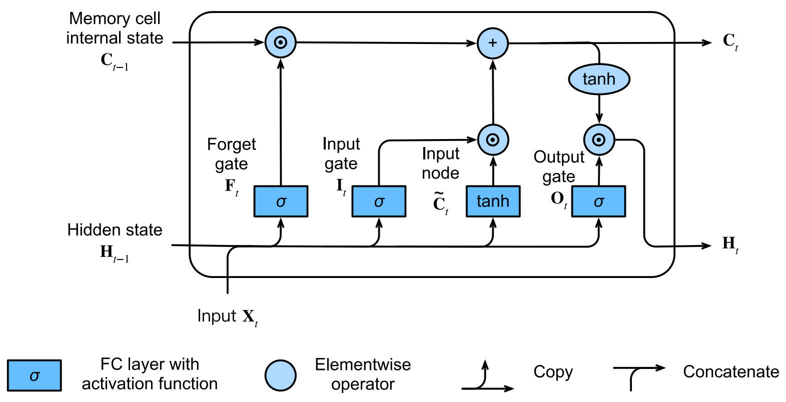

Figure 1.5: Inner structure of the LSTM unit. [8]

1.3.2 GRU

GRU is a variation on LSTM, which simplifies the model by merging the hidden state and memory cell state. It also replaces the three gates with just two gates. The *reset gate* (r_t) decides how much the previous state will be remembered. When the reset gate is close to 0, the hidden state ignores the previous hidden state $\mathcal{H}_{t-1}$ and resets itself with the current input. This allows the network to discard information that is not relevant. The *update gate* (u_t) controls how much of the previous hidden state will be copied into the current hidden state. This serves a similar function to the memory cell of LSTMs and helps capture long-term dependencies. The resulting model performs similarly to LSTM, while being simpler and faster to compute. [8, 14]

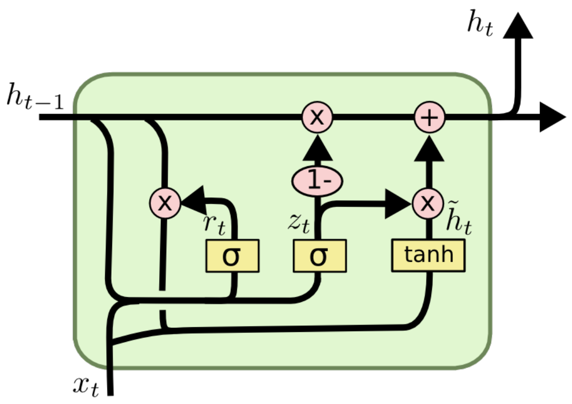

Figure 1.6: Inner structure of the GRU unit. [13]

1.3.3 Encoder-decoder and seq2seq networks

Some problems require the recurrent network to take one sequence of data and output a different sequence. For example in machine translation, the model takes a sentence in one language and tries to predict a corresponding sentence in a different language. But due to different grammatical rules, the two sentences might have a different ordering of the words or even different lengths. Generally, problems that involve mapping a variable-length sequence to a different variable-length sequence are known as *sequence-to-sequence* (seq2seq) problems. [8]

A common approach for seq2seq is to use an *encoder-decoder* network. The encoder RNN processes the input sequence $X_1, \ldots, X_t$ into a fixed-size hidden state often called the *context* and the decoder RNN then uses the context to generate the output sequence $Y_1, \ldots, Y_{t'}$. The last state $\mathcal{H}_t$ of the encoder is often provided as an input to the decoder RNN (demonstrated in Figure 1.7). Both RNNs are then jointly trained to maximize the average of $\log \mathrm{P}(Y_1, \ldots, Y_{t'} \mid X_1, \ldots, X_t)$ across all pairs of X and Y in the training set. [10, 15]

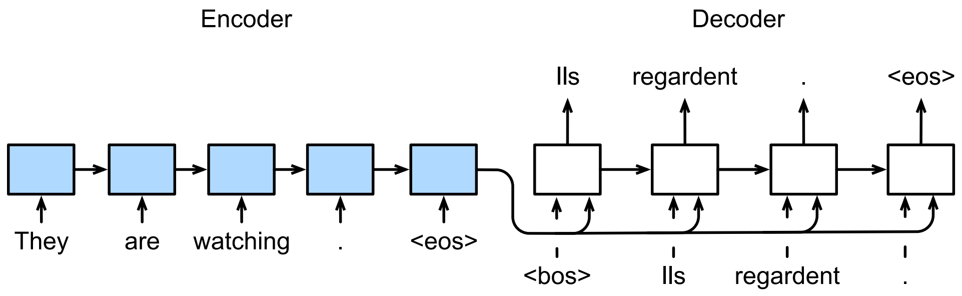

Figure 1.7: Example of seq2seq network for machine translation with encoder and decoder RNNs. The last state of the encoder is fed into the decoder as an input along with the previously generated word. [8]

Networks for processing sequences of images combine the encoder-decoder architecture with convolutional layers. The encoder utilizes convolutional and pooling layers to reduce the input images into a smaller representation and the decoder uses convolutional and upsampling layers, which increase the spatial size of the output. [12] A notable feature in these networks are *skip connections*, which combine information from early layers into later layers, usually through addition or concatenation (shown in Figure 1.8). This provides an additional path for the gradient to flow through and is one of the ways of tackling the vanishing gradient problem. Skip connections also allow to pass features from the encoder directly to the decoder in order to recover information lost during downsampling and to help with feature reusability. Many modern network architectures, such as U-Net, ResNet, or DenseNet, heavily utilize skip connections. [16, 12]

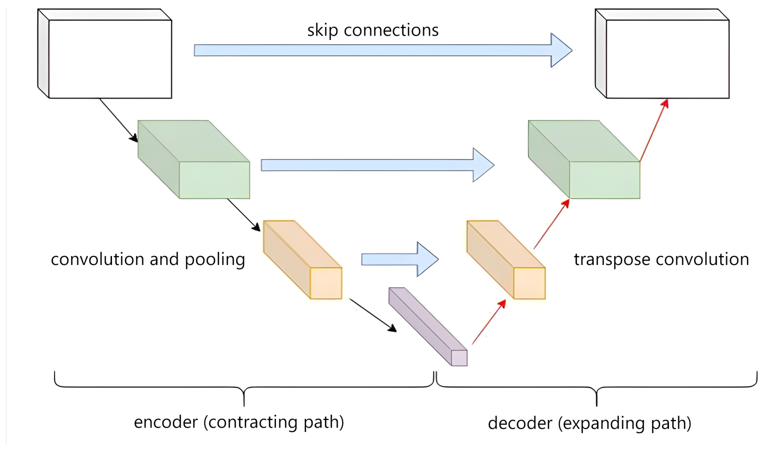

Figure 1.8: Convolutional encoder-decoder network architecture with skip connections. [16]

1.4 Attention mechanism

One flaw of encoder-decoder networks is that they need to compress all the information from the input sequence into a fixed-length state, which presents a challenge with increasing length of the sequences. A solution to this problem is a mechanism known as *attention*. It is the core idea behind Transformers, found in many state-of-the-art models, including OpenAI's GPT-4. The intuition behind attention is that it allows the network to selectively focus on specific parts of the input at each decoding step by enabling the decoder to periodically revisit the input sequence. [8, 17]

The idea is for the encoder to produce a representation of the same length as the input sequence. The decoder then receives a weighted vector of these representations with the weights deciding how much focus should be given to a particular input token. When predicting, the model only attends to parts of the input that are deemed relevant to the current prediction. [8]

Vaswani et al. describe attention in [18] as "mapping of a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key."

Given a database $\mathcal{D} = (k_1, v_1), \dots, (k_n, v_n)$ of n pairs of keys and values and a query q we can define attention over $\mathcal{D}$ as

$$\text{Attention}(q, \mathcal{D}) = \sum_{i=1}^{n} \alpha(q, k_i) v_i, \tag{1.4}$$

where $\alpha(q, k_i) \in \mathbb{R}, i = 1, \dots, n$ are scalar attention weights of the values.

To ensure that the weights are not negative and add up to 1, they are usually normalized by the softmax function as

$$\alpha(q, k_i) = \text{softmax}(a(q, k_i)) = \frac{\exp(a(q, k_i))}{\sum_j \exp(a(q, k_j))}, \tag{1.5}$$

where $a(q, k_j)$ is an *attention scoring* or *alignment* function, most common being dot-product and additive attention.

In practice, attention is often calculated on multiple queries simultaneously. Suppose we have n queries and m key-value pairs with queries and keys of length d_k and values of length d_v. The queries, keys and values are stacked into matrices $Q \in \mathbb{R}^{n \times d_k}$, $K \in \mathbb{R}^{m \times d_k}$ and $V \in \mathbb{R}^{m \times d_v}$. The attention can then be formulated using matrix multiplication as:

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(a\big(QK^T\big)\Big)V. \tag{1.6}$$

Figure 1.9 shows the entire attention pooling process. [18, 17]

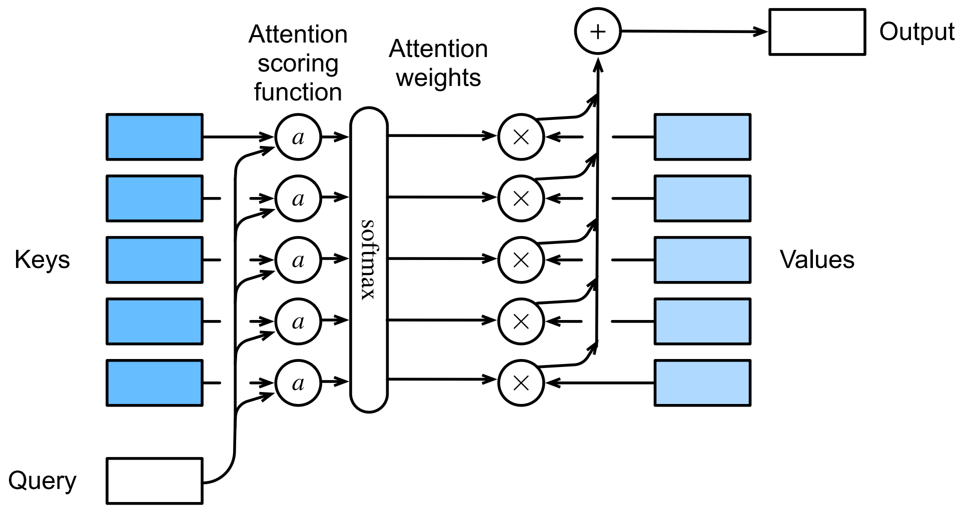

Figure 1.9: Computing the output of attention pooling as a weighted average of values, where weights are computed with the attention scoring function and the softmax operation. [8]

A special case of the attention mechanism, when all of the keys, values, and queries come from the same source, is known as *self-attention*. It relates different positions of a single sequence in order to compute a representation of the sequence. It is a key concept behind SA-ConvLSTM, a model for spatiotemporal prediction described in section 2.3. [18]

Chapter 2

Models for spatiotemporal prediction

Spatiotemporal predictive learning is an important problem and a hot research topic in the fields of artificial intelligence and computer vision. The task is to make predictions from input sequences that consider both the variations over time and the changes in the spatial structure. It has applications in numerous fields – video prediction, surveillance, traffic flow prediction, simulation of physical interactions, or, most importantly for this thesis, weather forecasting.

To define this problem formally, suppose that we are observing a dynamical system, such as a video clip, of P measurements (e.g. RGB image channels) over time. These measurements are recorded across all points in a spatial region represented by $M \times N$ grid with M rows and N columns. Observation at any time can be represented by a tensor $X \in \mathbb{R}^{P \times M \times N}$. Observations over T time steps form a sequence of tensors $X_1, X_2, \ldots, X_T$. The spatiotemporal prediction problem is to predict the most likely length-K sequence in the future

$$\tilde{X}_{t+1}, \ldots, \tilde{X}_{t+K} = \underset{X_{t+1},\ldots,X_{t+K}}{\arg\max}\, p\left(X_{t+1}, \ldots, X_{t+K} \mid \hat{X}_{t-J+1}, \ldots, \hat{X}_t\right) \tag{2.1}$$

given the previous J observations, including the current one. [1, 19]

Neural networks that solve this problem usually use the recurrent encoder-decoder architecture described in the previous chapter, along with convolutional layers for processing the images (although recently, other machine learning models, such as the Transformer, have also been successfully adopted for this task [20]). This chapter presents some of the models used for spatiotemporal prediction.

2.1 PhyDNet

PhyDNet, introduced by Vincent Le Guen and Nicolas Thome in [21], is the primary focus of this thesis. It is a seq2seq deep model dedicated to video prediction, which leverages physical dynamics described by partial differential equations (PDEs) that are disentangled from unknown factors (i.e. appearance, details, or texture).

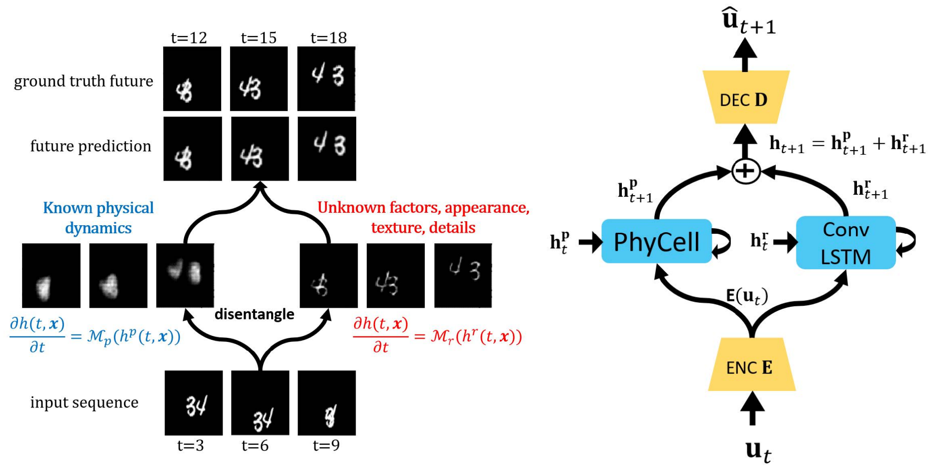
t=12
t=15
t=18
ground truth future
future prediction
Known physical dynamics
Unknown factors, appearance, texture, details
disentangle
$\frac{\partial h(t,\mathbf{x})}{\partial t} = \mathcal{M}_p(h^p(t,\mathbf{x}))$
$\frac{\partial h(t,\mathbf{x})}{\partial t} = \mathcal{M}_r(h^r(t,\mathbf{x}))$
input sequence
t=3
t=6
t=9
$\hat{\mathbf{u}}_{t+1}$
DEC **D**
$\mathbf{h}_{t+1} = \mathbf{h}^{\mathbf{p}}_{t+1} + \mathbf{h}^{\mathbf{r}}_{t+1}$
$\mathbf{h}^{\mathbf{p}}_{t+1}$
$\mathbf{h}^{\mathbf{r}}_{t+1}$
$\mathbf{h}^{\mathbf{p}}_{t}$
PhyCell
$\mathbf{h}^{\mathbf{r}}_{t}$
Conv LSTM
$\mathbf{E}(\mathbf{u}_t)$
ENC **E**
$\mathbf{u}_t$

Figure 2.1: Left: Example of the disentanglement of physical dynamics from unknown factors on the Moving MNIST dataset. Right: The structure of the recurrent PhyDNet unit. [21]

Because physical laws might not apply at pixel level of the input video, PhyDNet maps the input into latent space $\mathcal{H}$. The authors assume that in $\mathcal{H}$, the mechanics in the video can be described by physical laws. The model learns $\mathcal{H}$ in an unsupervised manner using a deep convolutional encoder E and decoder D so that the physical dynamics are disentangled from other information.

The inner architecture consists of two parallel branches (see Figure 2.1). The left branch captures the physical dynamics using a recurrent physical cell, PhyCell, which models a broad class of PDEs approximated by convolutional layers. The right branch contains a deep recurrent ConvLSTM network that captures all the residual dynamics.

Both branches complement each other – the PhyCell regularizes training and improves generalization, while the ConvLSTM captures unknown phenomena which do not correspond to any prior model and are learned entirely from data. The physical constraints of the PhyCell also help reduce the number of parameters.

Given spatial coordinates $\mathrm{x} = (x, y)$, the latent representation of the video up to time t, $\mathrm{h}(t, \mathrm{x}) \in \mathcal{H}$, can be decomposed as $\mathrm{h} = \mathrm{h}^{\mathrm{p}} + \mathrm{h}^{\mathrm{r}}$, where h^{p} and h^{r}

represent the physical and residual dynamics, respectively. The evolution of the video in $\mathcal{H}$ is then governed by the following PDE:

$$\frac{\partial \mathrm{h}(t, \mathrm{x})}{\partial t} = \frac{\partial \mathrm{h}^{\mathrm{p}}}{\partial t} + \frac{\partial \mathrm{h}^{\mathrm{r}}}{\partial t} := \mathcal{M}_p(\mathrm{h}^{\mathrm{p}}, \mathrm{u}) + \mathcal{M}_r(\mathrm{h}^{\mathrm{r}}, \mathrm{u}), \tag{2.2}$$

where $\mathrm{u} = \mathrm{u}(t, \mathrm{x})$ is the frame of a video at time t and $\mathcal{M}_p(\mathrm{h}^{\mathrm{p}}, \mathrm{u})$ and $\mathcal{M}_r(\mathrm{h}^{\mathrm{r}}, \mathrm{u})$ are the physical and residual dynamics in the latent space $\mathcal{H}$.

The encoded mapping of the input frame $\mathrm{E}(\mathrm{u}_t)$ is passed to the two branches that fulfill the respective parts of the PDE in 2.2. They subsequently compute $\mathrm{h}^{\mathrm{p}}_{t+1}$ (resp. $\mathrm{h}^{\mathrm{r}}_{t+1}$) from $\mathrm{E}(\mathrm{u}_t)$ and h^{p} (resp. h^{r}). The combined representation $\mathrm{h}_{t+1} = \mathrm{h}^{\mathrm{p}}_{t+1} + \mathrm{h}^{\mathrm{r}}_{t+1}$ is passed to the decoder D to predict the image $\hat{\mathrm{u}}_{t+1}$. [21]

2.2 ConvLSTM

Convolutional LSTM, or ConvLSTM, is an extension of the LSTM model for spatiotemporal prediction. Although LSTM handles temporal correlation well, it struggles with capturing the spatial correlation, because the fully-connected layers in input-to-state and state-to-state transitions in the cell cannot encode spatial information. To overcome this, ConvLSTM replaces these layers with convolutional layers. [1]

The inputs X_t to the ConvLSTM cell are 3D tensors with the first dimension being the number of channels and the other two being the spatial dimensions (height and width). The inner workings of the cell can be described by following equations with '$*$' representing the convolution operator:

$$\begin{aligned}
i_t &= \sigma\left(W_{xi} * X_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \odot \mathcal{C}_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{xf} * \mathcal{C}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \odot \mathcal{C}_{t-1} + b_f\right) \\
o_t &= \sigma\left(W_{xo} * \mathcal{C}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \odot \mathcal{C}_t + b_o\right) \\
g_t &= \tanh\left(W_{xc} * \mathcal{C}_t + W_{hc} * \mathcal{H}_{t-1} + b_c\right) \\
\mathcal{C}_t &= f_t \odot \mathcal{C}_{t-1} + i_t \odot g_t \\
\mathcal{H}_t &= o_t \odot \tanh\left(\mathcal{C}_t\right)
\end{aligned} \tag{2.3}$$

Like LSTM, ConvLSTM can also be used as a building block for more complex structures, often stacks of multiple ConvLSTM units. An example given by the authors of the original paper uses an encoding and forecasting network with both using two ConvLSTM units. Stacking multiple ConvLSTMs brings strong representational power to the network, making it suitable for predictions in complex dynamical systems, like precipitation nowcasting. [1] A similar stacked structure is also used in the residual cell of PhyDNet.

2.3 SA-ConvLSTM

SA-ConvLSTM adds a novel self-attention memory (SAM) module to Conv-LSTM. SAM is aimed at memorizing features with long-term dependencies in both the spatial and temporal domains. Based on self-attention, SAM can capture information by aggregating features across all positions of both the input itself and the network's memory with pair-wise similarity scores. Additionally, an extra memory cell $\mathcal{M}$ is used to memorize global spatial features. $\mathcal{M}$ can also capture long-range temporal dependencies through a gating mechanism, similar to that in LSTM. [22]

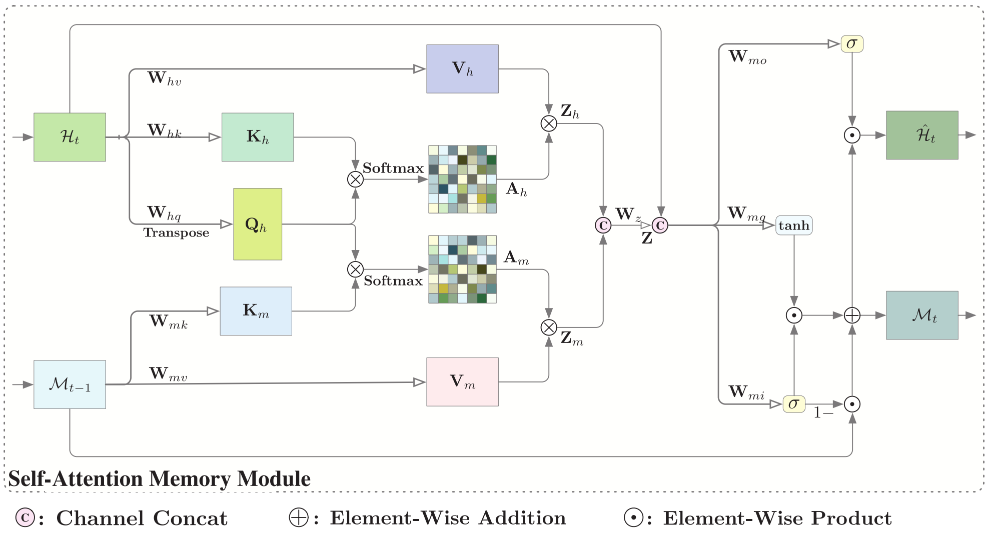

Figure 2.2: Structure of the SAM module. Q, K and V represent the query, key and value used in the attention mechanism. [22]

Figure 2.2 shows the pipeline of the self-attention memory module. SAM applies self-attention to both the hidden state $\mathcal{H}_t$ and the previous state of the memory cell $\mathcal{M}_{t-1}$. The hidden state $\mathcal{H}_t$ is mapped into different feature spaces as the query $Q_h = W_{hq} * \mathcal{H}_t \in \mathbb{R}^{\hat{C}\times N}$, key $K_h = W_{hk} * \mathcal{H}_t \in \mathbb{R}^{\hat{C}\times N}$ and value $V_h = W_{hv} * \mathcal{H}_t \in \mathbb{R}^{C\times N}$ and the memory cell state $\mathcal{M}_{t-1}$ is mapped into the key $K_m = W_{mk} * \mathcal{M}_{t-1} \in \mathbb{R}^{\hat{C}\times N}$ and value $V_m = W_{mv} * \mathcal{M}_{t-1} \in \mathbb{R}^{C\times N}$, where W are sets of weights for 1×1 convolutions, C and $\hat{C}$ are number of channels and $N = H \times W$.

The similarity scores between the queries and keys are calculated using matrix multiplication as:

$$e_h = Q_h^T K_h \in \mathbb{R}^{N\times N} \tag{2.4}$$

These are then normalized along columns by a softmax function:

$$\alpha_{h;i,j} = \frac{\exp e_{h;i,j}}{\sum_{k=1}^{N} \exp e_{h;i,k}}, i, j \in 1, 2, \ldots, N \tag{2.5}$$

The similarity scores e_m and α_m of the memory cell $\mathcal{M}$ are calculated the same way.

The features Z_h and Z_m are obtained as a weighted sum across all locations in the value. The i-th location in the feature is calculated as:

$$\begin{aligned} Z_{h;i} &= \sum_{j=1}^{N} \alpha_{h;i,j} V_{h;j} = \sum_{j=1}^{N} \alpha_{h;i,j} (W_{hv} \mathcal{H}_{t;j}) \\ Z_{m;i} &= \sum_{j=1}^{N} \alpha_{m;i,j} V_{m;j} = \sum_{j=1}^{N} \alpha_{m;i,j} (W_{mv} \mathcal{M}_{t-1;j}), \end{aligned} \tag{2.6}$$

where $V_{h;j}$ and $V_{m;j}$ are the j-th columns of the values. Z_m and Z_h are then concatenated into the aggregated feature $Z = W_z[Z_h, Z_m]$.

The memory $\mathcal{M}$ is updated by a gating mechanism, which combines the aggregated feature Z, hidden state $\mathcal{H}_t$ and memory $\mathcal{M}_{t-1}$. This allows the SAM to capture global past spatiotemporal information. The output feature of SAM is a dot product between an output gate and the updated memory state $\mathcal{M}_t$. The memory updating and module output can be formulated as:

$$\begin{aligned} i'_t &= \sigma\left(W_{m;zi} * Z + W_{m;hi} * \mathcal{H}_t + b_{m;i}\right) \\ g'_t &= \tanh\left(W_{m;zg} * Z + W_{m;hg} * \mathcal{H}_t + b_{m;g}\right) \\ \mathcal{M}_t &= (1 - i'_t) \odot \mathcal{M}_{t-1} + i'_t \odot g'_t \\ o'_t &= \sigma\left(W_{m;zo} * Z + W_{m;ho} * \mathcal{H}_t + b_{m;o}\right) \\ \hat{\mathcal{H}}_t &= o'_t \odot \mathcal{M}_t \end{aligned} \tag{2.7}$$

The SAM is then embedded into the ConvLSTM unit as shown in Figure 2.3. Without the SAM module, the SA-ConvLSTM would degenerate into standard ConvLSTM. [22]

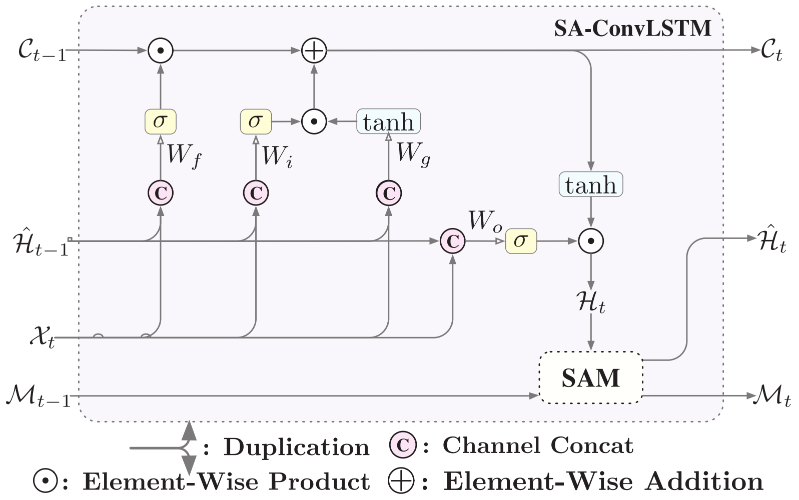

Figure 2.3: The SA-ConvLSTM unit. [22]

2.4 PredRNN

Another variation on ConvLSTM is the predictive RNN (PredRNN). It is based on the idea that both spatial and temporal variations should be memorized in a unified memory cell $\mathcal{M}$. This memory then travels through the network in two directions – across stacked RNN layers vertically and through all RNN states horizontally, allowing different LSTMs to interact with each other. On the whole, the standard temporal memory $\mathcal{C}$ flows in the horizontal direction and the new spatiotemporal memory $\mathcal{M}$ zigzags through the network in both directions (see Figure 2.4). This accomplishes an efficient flow of spatial information while being relatively prone to the vanishing gradient problem. [19]

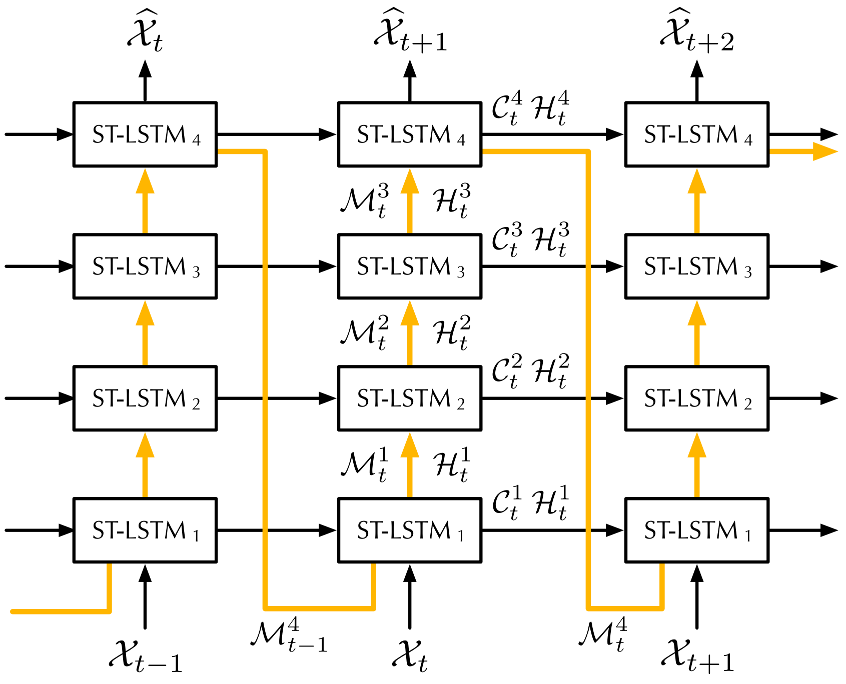
$\widehat{\mathcal{X}}_t$
$\widehat{\mathcal{X}}_{t+1}$
$\widehat{\mathcal{X}}_{t+2}$
ST-LSTM 4
ST-LSTM 3
ST-LSTM 2
ST-LSTM 1
$\mathcal{C}_t^4\ \mathcal{H}_t^4$
$\mathcal{M}_t^3\ \mathcal{H}_t^3$
$\mathcal{C}_t^3\ \mathcal{H}_t^3$
$\mathcal{M}_t^2\ \mathcal{H}_t^2$
$\mathcal{C}_t^2\ \mathcal{H}_t^2$
$\mathcal{M}_t^1\ \mathcal{H}_t^1$
$\mathcal{C}_t^1\ \mathcal{H}_t^1$
$\mathcal{X}_{t-1}$
$\mathcal{M}_{t-1}^4$
$\mathcal{X}_t$
$\mathcal{M}_t^4$
$\mathcal{X}_{t+1}$

Figure 2.4: The architecture of PredRNN. The orange arrows show the flow of the memory state $\mathcal{M}$. © 2021 IEEE [23]

This design is supported by a new unit, the spatiotemporal LSTM (ST-LSTM), shown in Figure 2.5. It maintains the cells and gates from LSTM but adds a new gating mechanism for updating the spatiotemporal memory $\mathcal{M}$. ST-LSTM also uses a shared output gate for both memory types to enable seamless memory fusion, which can effectively model the shape deformations and motion trajectories in the spatiotemporal sequences. [19]

The entire model is described by the following set of equations, where $\mathcal{H}_{t-1}^l$ represents the hidden state in layer l at a time step $t-1$:

$$
\begin{aligned}
i_t &= \sigma\left(W_{xi} * X_t + W_{hi} * \mathcal{H}_{t-1}^{l} + b_i\right)\\
f_t &= \sigma\left(W_{xf} * X_t + W_{hf} * \mathcal{H}_{t-1}^{l} + b_f\right)\\
g_t &= \tanh\left(W_{xg} * X_t + W_{hg} * \mathcal{H}_{t-1}^{l} + b_g\right)\\
i_t' &= \sigma\left(W_{xi}' * X_t + W_{mi} * \mathcal{M}_{t}^{l-1} + b_i'\right)\\
f_t' &= \sigma\left(W_{xf}' * X_t + W_{mf} * \mathcal{M}_{t}^{l-1} + b_f'\right)\\
g_t' &= \tanh\left(W_{xg}' * X_t + W_{mg} * \mathcal{M}_{t}^{l-1} + b_g'\right)\\
o_t &= \sigma\left(W_{xo} * X_t + W_{ho} * \mathcal{H}_{t-1}^{l} + W_{co} * \mathcal{C}_{t}^{l} + W_{mo} * \mathcal{M}_{t}^{l} + b_o\right)\\
\mathcal{C}_t^l &= f_t \odot \mathcal{C}_{t-1}^{l} + i_t \odot g_t\\
\mathcal{M}_t^l &= f_t' \odot \mathcal{M}_{t}^{l-1} + i_t' \odot g_t'\\
\mathcal{H}_t^l &= o_t \odot \tanh\left(W_{1\times 1} * \left[\mathcal{C}_t^l, \mathcal{M}_t^l\right]\right)
\end{aligned}
\tag{2.8}
$$

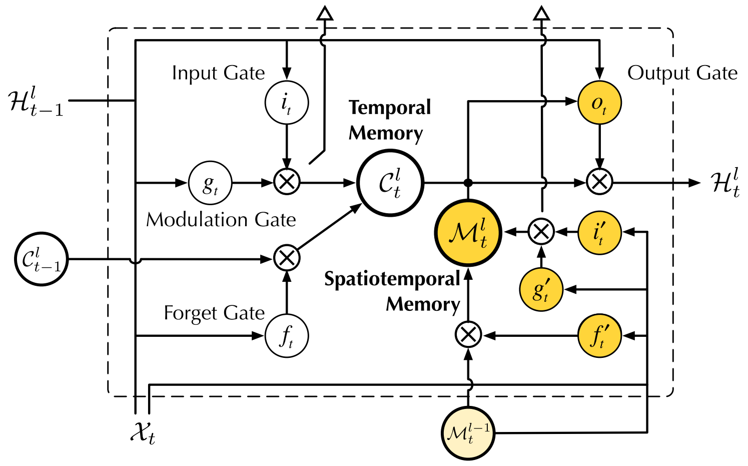

Figure 2.5: Inner structure of the ST-LSTM unit. Yellow circles show additional features compared to ConvLSTM. © 2021 IEEE [23]

2.4.1 PredRNN-v2

The authors of PredRNN later improved upon the concept in [23] with a new decoupling loss function that maximizes the distance of memory states between the two memory cells $\mathcal{C}$ and $\mathcal{M}$. This forces the cells to focus on different aspects of spatiotemporal variations. They also presented a new learning procedure, Reverse Scheduled Sampling, used at the encoding time steps, that makes the model learn more about long-term dynamics by randomly hiding real observations with decreasing probabilities as the training phase progresses.

2.4.2 PredRNN++

A different improvement on PredRNN presented by the same authors is PredRNN++. It replaces the ST-LSTM unit with a new unit, Casual LSTM, which updates the memory through a cascaded mechanism shown in Figure 2.6. This allows for a larger recurrence depth from one step to the next and thereby obtains more modeling power for short-term video dynamics and sudden changes. A second improvement presented is a highway layer that provides an uninterrupted alternative way for the gradient to flow, further helping to combat gradient backpropagation issues. [24]

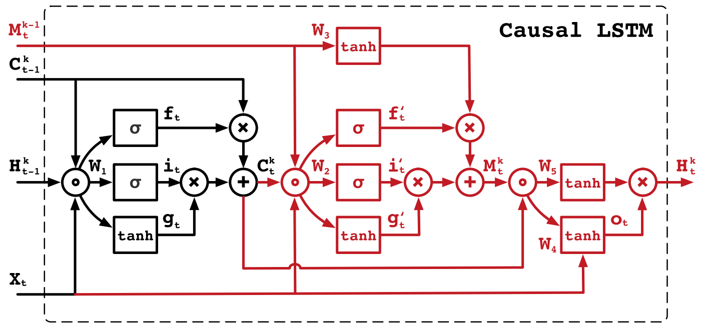

Figure 2.6: Casual LSTM unit. Colored parts represent additional operations compared to ConvLSTM. [24]

2.5 TrajGRU

One problematic aspect of convolutional RNNs is that the convolutions apply a location-invariant filter. In other words, the neighborhood used for the calculation stays the same for all locations. However, most motion patterns require different neighborhoods for different locations. TrajGRU attempts to combat this by using the current input and the previous state to generate a neighborhood set for each location at each time step through a convolutional subnetwork (see Figure 2.7). The subnetwork can learn this connection topology simply by learning its parameters. [2]

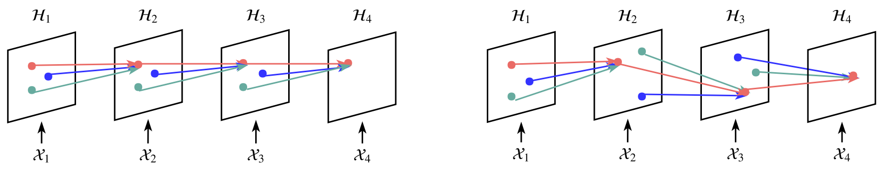

Figure 2.7: The connections in a convolutional RNN (left) are fixed, whereas connections in TrajGRU (right) are dynamically determined. [2]

Chapter 3

Precipitation nowcasting

Weather is an essential aspect of our lives. It influences both our day-to-day decisions, like the choice of clothing or transportation, as well as large-scale, critical activities, such as agriculture, transportation or disaster management. In many areas of the world, weather presents a significant threat to human lives and property, with its impact possibly increasing in the near future due to climate change. Weather forecasting allows us to understand the current and future weather conditions and is critical to making informed decisions and taking appropriate actions.

Nowcasting is a specific type of weather forecasting defined by the World Meteorological Organization as “forecasting with local detail, by any method, over a period from the present to 6 hours ahead, including a detailed description of the present weather”. [25] Precipitation nowcasting applies this concept to predictions of rainfall intensity. It is important for enhancing the safety of people by providing weather guidance for air traffic control and helping to issue emergency rainfall alerts among many others.

However, precipitation nowcasting is a difficult problem. Besides the already very complex nature of the atmosphere and its dynamic processes, the forecasting resolution and time accuracy required to give precise predictions are much higher than in other forecasting tasks, like predicting the average temperature. [1]

Methods for precipitation nowcasting roughly fall into three categories – methods based on numerical weather prediction (NWP), radar echo extrapolation methods, and deep learning methods. NWP-based methods build on very sophisticated simulations of physical equations in atmospheric models. Even though they can extend the nowcasting range up to 6 hours, they require long computation times. Most models, therefore, favor faster and more accurate radar echo map extrapolation techniques or deep learning models. [1, 25]. This chapter gives an overview of some of these methods.

3.1 Weather radars

Weather radar systems are the most important instruments for measuring precipitation data. Similar to how sound reflects off surfaces and comes back in the form of an echo, radars measure rainfall based on signal reflections. More specifically, radars send out electromagnetic (radio) waves traveling at approximately the speed of light, which are then reflected by the precipitation back to the radar. From this information, the radar can calculate the position and intensity of the precipitation. Radars can usually measure two types of data – reflectivity, which measures the amount of precipitation in an area, and velocity, which uses the Doppler effect to measure the speed and direction of the rainfall. [26]

Radars have numerous advantages over other observing systems. They can capture precipitation particles over a very large area in three dimensions with an update rate of a few minutes. They can also operate unmanned in all weathers and during the day or night. However, they are also very sophisticated and expensive to build and maintain. [25]

Methods for precipitation nowcasting take radar echo images as their input data. These are maps of reflected particles over an area surrounding the radar at a constant altitude. The maps display a color image of the precipitation intensity measured in dBZ units, which measure the relative reflectivity factor (Z) of an object.

3.2 Radar echo extrapolation

Conventional methods for precipitation nowcasting often use optical flow for extrapolation from radar echo maps. Generally, these algorithms identify storms as objects in the radar maps and then track the motion of the storm by identifying the same object in successive radar echo images. [25]

3.2.1 COTREC

COTREC is a radar extrapolation method used by the nowcasting system of the Czech Hydrometeorological Institute built on the TREC method. [27] TREC predicts the motion of the storms by taking two consecutive radar scans and computing translation vectors of patterns observed in the two images.

The input images are divided into boxes of the same size. Each box from the first image is then compared to the relevant boxes in the second image, usually those found in a circular area around the initial box. The pair with the highest correlation coefficient is then used to determine the translation vector that indicates the motion of the initial box.

The results of TREC are often noisy and inconsistent. COTREC attempts to solve this with two improvements. Firstly, vectors that significantly deviate from the mean direction of their neighbors are replaced by the average of the neighboring vectors. Secondly, COTREC forces continuity on the derived motion vectors with specific requirements through variational analysis. [28]

3.2.2 STEPS

Extrapolation-based methods assume that the precipitation field does not grow or shrink but simply shifts to a different location. To address this, the nowcasting system STEPS (Short-Term Ensemble Prediction System) presents an ensemble model that merges extrapolation nowcasting with a scaled-down NWP forecast.

The system is able to decompose the precipitation field into a cascade of features using the Fourier transform. Each level of the cascade represents a different spatial scale and is treated separately from the other levels. By combining the predictions, the model can overcome the limitations of both methods at different scales. [29]

3.3 Deep learning

In recent years, deep learning models have emerged as a promising alternative to optical flow methods. These models use the seq2seq encoder-decoder architecture that takes a sequence of radar images as input and produces a sequence of output images with the predictions.

The spatiotemporal prediction models presented in the previous chapter have been used for spatiotemporal prediction with promising results. The experiments conducted on ConvLSTM and TrajGRU have shown that these models can outperform state-of-the-art optical flow models. [1, 2]

Chapter 4

Dataset and implementation

The aim of this thesis is to try to improve the PhyDNet model when used for precipitation nowcasting. As the original PhyDNet uses only a basic Conv-LSTM model in the residual branch, our improvements focused on this part of the model. This chapter describes details of the implementation, the radar echo dataset, and the procedure used for training the networks.

4.1 Radar echo dataset

The dataset used in this thesis was provided by the company Meteopress. It consists of 249,196 radar images taken from 2015-10-23 19:30 UTC up until 2020-07-21 23:50 UTC, acquired through the OPERA program of the European Meteorological Network (EUMETNET). The data was recorded every 10 minutes, meaning there are 144 frames per day. The aerial domain of the images is a rectangular area covering the entirety of the Czech Republic with small parts of the neighboring countries. The dataset contains a total of 108,646 rainy images, amounting to about 18,096 hours of precipitation. [30]

The images are stored as gray-scale PNG files with resolution of 544×352 pixels. The pixel values represent the intensity of rainfall in dBZ units. The radar measurements, which range from 0 dBZ (no precipitation) to 60 dBZ (heavy rainfall), are linearly mapped into the 8-bit range [0, 255] of the PNG images. These input values are then scaled to floating point numbers in the range [0, 1] to help stabilize the training process.

The consecutive images are sliced with a 12-frame-wide sliding window to generate the sequences, which are then split into a training dataset with 17,361 sequences, a validation dataset with 4,960 sequences, and a test dataset with 2,481 sequences.

4.2 Used libraries and tools

The project was implemented in Python 3.10 using machine learning frameworks PyTorch 1.13, PyTorch Lightning 1.9, and Tensorboard, used for logging the metrics. The visualizations of the predictions were made using the popular Python library matplotlib.

The implementation of PhyDNet was adapted from the source code[1] published with the original paper [21] and from later modifications[2] introduced by Matej Choma in his thesis [30]. The code for PredRNN was derived from the official implementation[3] of [23] and the SA-ConvLSTM implementation was based on an open source implementation [31]. The rest of the code is our own work.

The networks were trained on a specialized DGX computing station, belonging to Datalab of FIT CTU, equipped with AMD EPYC 7742 64-core processors and four Nvidia A100 graphics processing units (GPUs) with 40 GBs of memory.

4.3 Common settings and hyperparameters

Most of the settings and hyperparameters were the same for all trained models. Cases, where the experiments deviated from these common settings, are noted in the next chapter.

4.3.1 Training

All models were trained using the Adam optimizer (section 1.1.2) with default parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and learning rate of 10^{-3}. [4] The loss function minimized during training was $\mathcal{L}_{1,2}$. The training process was set to a maximum of 50 epochs with early stopping deciding when to end the training to avoid overfitting.

The networks were trained with 6-frame input sequence and 6-frame output sequence. When evaluating some experiments, we increased the length of the output sequence, because the differences between the networks become more noticeable with higher lead times and also to examine the long-term capabilities of the networks. The mini-batch size was set to 8 for all models.

4.3.2 PhyCell and ConvLSTM hyperparameters

The PhyCell consists of a series of convolutions approximating various PDEs. These PDEs were computed up to an order of $k = 7$ with 49 linear terms in the PDE.

[1] `https://github.com/vincent-leguen/PhyDNet`

[2] `https://gitlab.fit.cvut.cz/chomamat/dp-text`

[3] `https://github.com/thuml/predrnn-pytorch`

The ConvLSTM model used 3 individual ConvLSTM layers with dimensions of hidden states set to 128, 128, and 64, respectively. All convolutional layers used 3×3 kernels.

The convolutional encoders and decoders both had 3 convolutional layers with 3×3 kernels, followed by a GroupNorm with 16 groups and a Leaky ReLU with a negative slope of 0.2. The encoder used standard convolutions to downsample the images, while the decoder used transposed convolution operators to upsample the images.

Chapter 5

Experiments

In this chapter, all of the conducted experiments are presented and described. These range from smaller adjustments to the baseline PhyDNet network to experiments with replacing the ConvLSTM network with different models. The chapter contains both quantitative and qualitative comparisons of these networks and discusses the results.

5.1 Branch connection

Our first experiment tried connecting both branches of the network by adding the output from the PhyCell as a second input to the ConvLSTM. The idea was that instead of each branch learning independently, the residual branch could know the features learned in the physical branch and utilize them in its own prediction. The inputs were combined using two approaches, addition and concatenation along the channel axis. For concatenation, we tried mapping the two 64-channel inputs into one 64-channel input with convolutional layers.

In both cases, the modified networks had slightly worse MAE but better SSIM, as shown in Table 5.1. The additive approach also achieved a minor improvement in MSE. Surprisingly, the concatenating network yielded predictions with more areas with intense precipitation (shown in Fig 5.1). However, it also tended to predict no precipitation in some areas with less intense rainfall (as evident by some of the holes in the bottom right picture in Fig 5.1), possibly causing the increase in MSE and MAE.

5.2 Separate decoders

In the original model, the decoder D receives the summed outputs from both branches and then maps them into the original input pixel space. As both branches serve distinct purposes, their outputs differ quite significantly. Therefore, we tried passing the output of each branch to its own decoder and

Table 5.1: Relative change in performance of PhyDNet with connected branches compared to the baseline model. Red denotes loss of performance. This convention is used in the rest of the tables in this chapter.

Model	MSE	MAE	SSIM
connected branches (addition)	−0.60%	+1.63%	+0.10%
connected branches (concat)	+1.73%	+0.40%	+0.06%

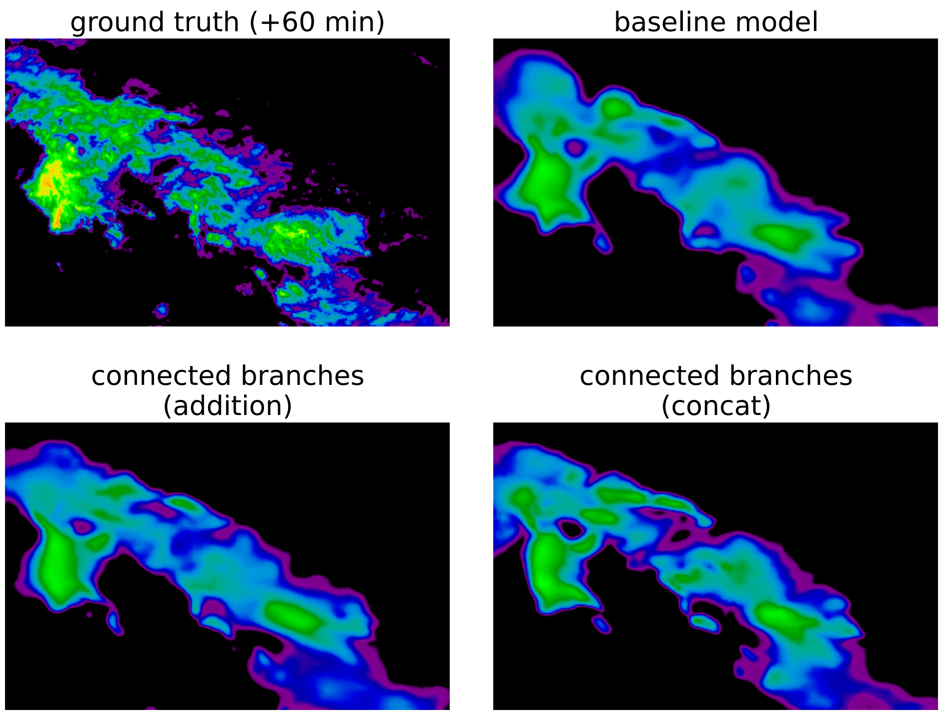

Figure 5.1: Comparison of predictions of PhyDNet with connected branches.

summing the outputs after decoding. Both decoders used the same architecture with 3 convolutional layers with 3×3 kernels and respective strides of 2, 1, and 2. This setup was also trained with the ConvLSTM taking the concatenation of PhyCell and encoder outputs as an input in the same fashion as the previous experiment.

Table 5.2 shows that both networks did not achieve any significant improvement. Although MSE and SSIM did improve slightly for the network with independent branches, it did not warrant the additional parameters and memory usage. The network with connected branches had worse results across all metrics, likely due to the fact that sharing the features between both branches forced the decoders to operate less independently. Examples in Figure 5.2 show that the network with connected branches produced slightly less

Table 5.2: Performance of PhyDNet with separate decoders.

Model	MSE	MAE	SSIM
separate decoders	−1.81%	+0.54%	+0.06%
separate decoders (connected branches)	+2.42%	+1.27%	−0.006%

accurate shapes of the predicted clouds. Conversely, the other network (shown on the bottom left) struggled somewhat with generating the high-intensity areas.

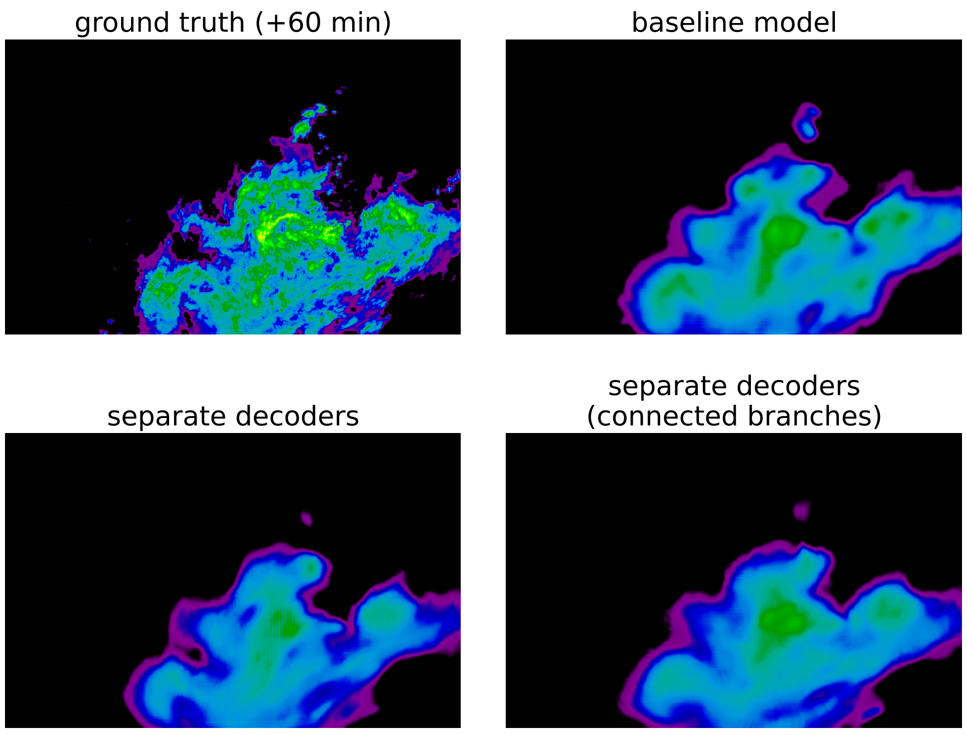

Figure 5.2: Comparison of predictions of PhyDNet with separate decoders.

5.3 Skip connections

As mentioned in 1.3.3, many networks benefit from the usage of skip (or residual) connections, which act as a shortcut for both the information and gradient flow in the network. These were the inspiration for the next set of experiments.

In the first experiment, we added a skip connection to the physical branch of the network, multiplying the output from the PhyCell with the decoder output. The second experiment introduced a global skip connection across the

Table 5.3: Performance of PhyDNet with skip connections.

Model	MSE	MAE	SSIM
PhyCell connection	−0.24%	−0.27%	+0.11%
global connection	+1.81%	+9.17%	−1.11%
global & PhyCell connection	−3.03%	+2.17%	−0.08%

entire network. The generated final prediction is added to the original input, causing the network to only predict updates to the previous input instead of predicting the entire image. The third experiment with skip connections combined the approaches from the first and second experiments.

Although the first experiment brought only a very slight decrease in MAE and MSE and increase SSIM (Table 5.3), the predictions shown in Figure 5.4 have a more refined shape and can more accurately capture the core areas of the storm with heavy rainfall. Figure 5.3 demonstrates that when inputting longer sequences, the network with the PhyCell connection gives much more detailed predictions compared to the baseline model, suggesting that the connection might help capture long-term dependencies. Also, as shown in Figure 5.8, when comparing the metrics for longer output sequences, the quality of the predictions holds up over time compared to other experiments.

Experiments with a global skip connection resulted in worse performance in almost all metrics and the predictions had less pronounced shapes and were slightly noisier. Although the combination of both skip connections did result in an improved MSE, the performance quickly deteriorated with increased leading times and the visualized predictions were less detailed than with the PhyCell connection.

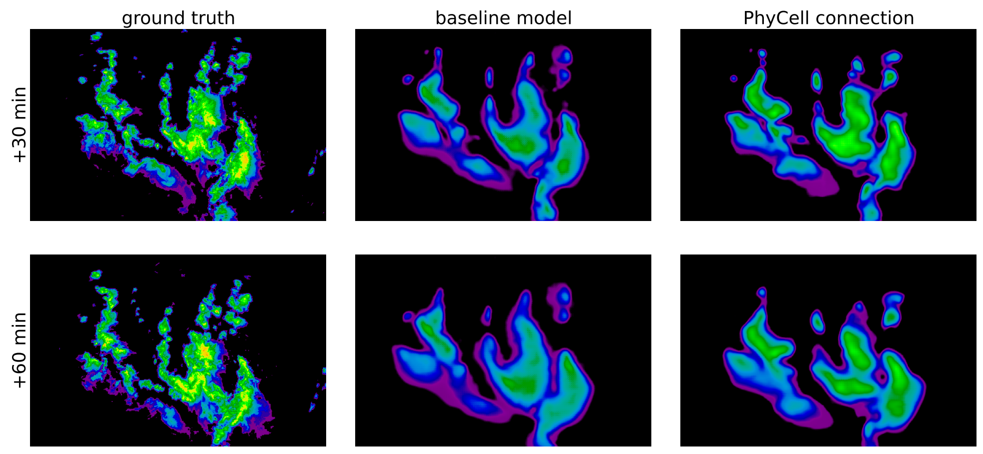

Figure 5.3: Comparison of predictions of PhyDNet with skip connections with an input sequence of 2 hours.

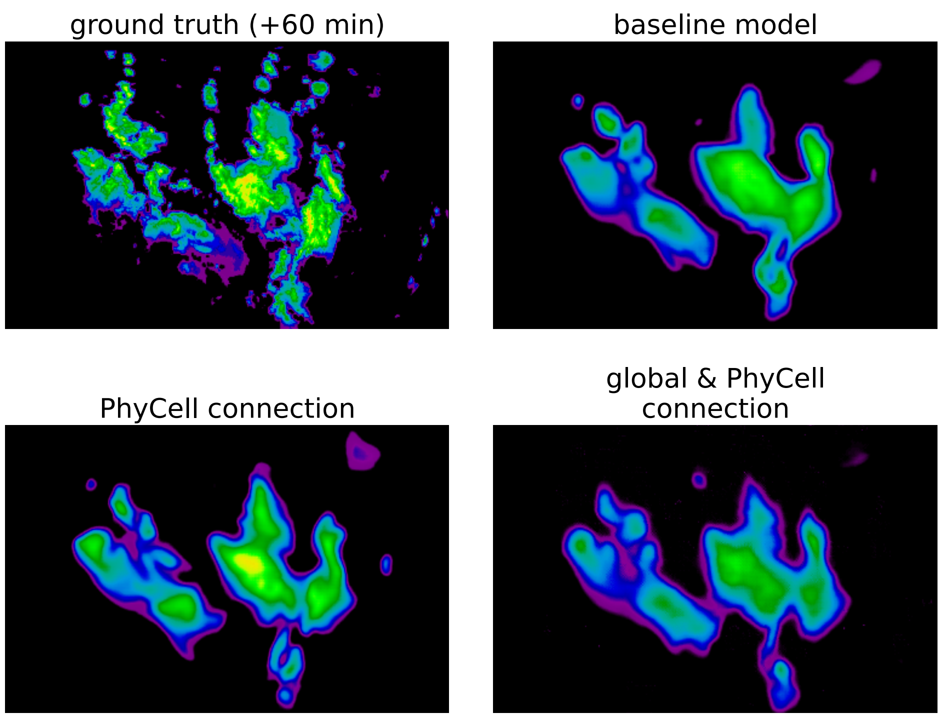

Figure 5.4: Comparison of predictions of PhyDNet with skip connections.

5.4 SA-ConvLSTM

Further experiments focused on replacing the ConvLSTM with a different model for spatiotemporal prediction. The first model we tried was SA-ConvLSTM, which extends the ConvLSTM using the self-attention mechanism. However, as the authors who proposed this model warned [22], the modified self-attention used in this model can have very significant computational requirements for large images. This proved to be a problem in our case, where even after downsampling by the encoder, the images are still relatively large (almost thrice as large as the Moving MNIST images, for example), requiring heavy GPU usage.

The implemented model consisted of 3 SA-ConvLSTM layers with hidden dimensions of 128, 64, and 64, respectively. The network was trained once without any normalization after the convolutional layer in the LSTM layers and once with a GroupNorm layer with 16 channel groups. We used common settings for the rest of the hyperparameters.

Although results in Table 5.4 show that SA-ConvLSTM achieved comparable or better performance in most metrics, the decomposed predictions from both branches showed that the optimizer heavily favored updating the PhyCell over SA-ConvLSTM, which had only minimal training. Using nor-

Table 5.4: Performance of PhyDNet with SA-ConvLSTM.

Model	MSE	MAE	SSIM
SA-ConvLSTM	−4.17%	+0.73%	−0.01%
SA-ConvLSTM + GroupNorm	−2.05%	−0.28%	+0.10%

malization in the SA-ConvLSTM units to combat the potential vanishing of information did not help much as the SA-ConvLSTM branch still showed very limited signs of learning. While this problem could probably be solved by a more complex training procedure or optimizer adjustments, because of the large GPU requirements, we decided to abandon this model.

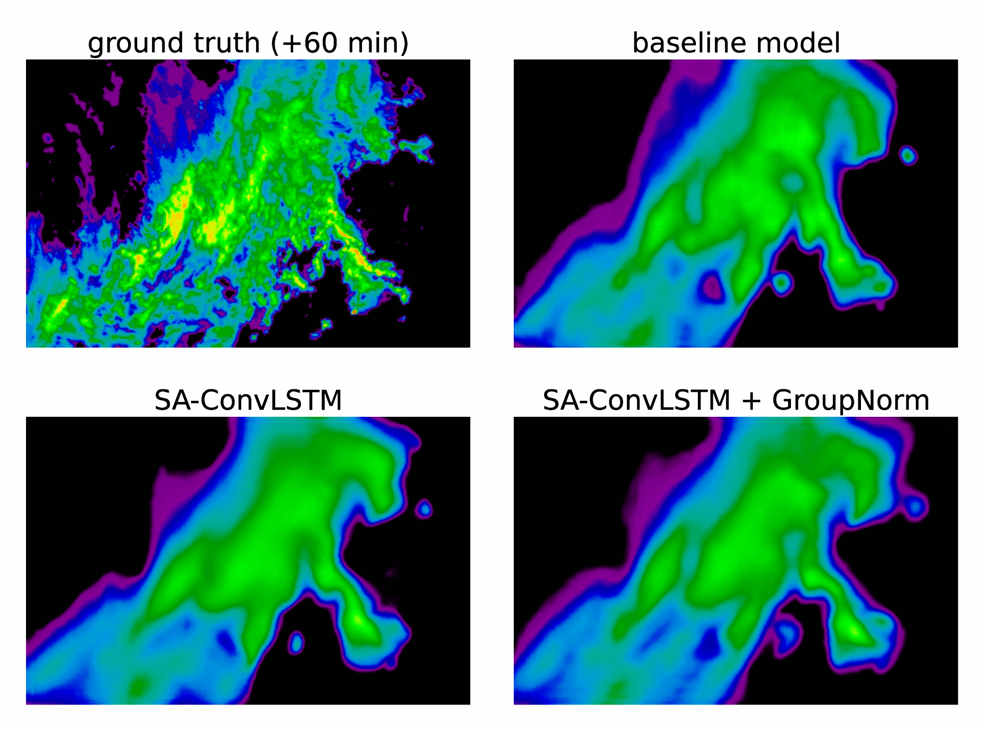

Figure 5.5: Comparison of predictions of PhyDNet with SA-ConvLSTM.

5.5 PredRNN

The last set of experiments tried using PredRNN instead of ConvLSTM in the residual branch. The implemented PredRNN network contained 3 ST-LSTM cells, each with 64-channel hidden states, 3×3 convolutional kernels with a stride of 1. We trained PhyDNet with PredRNN in two different setups, once

Table 5.5: Performance of PhyDNet with PredRNN.

Model	MSE	MAE	SSIM
PredRNN	−1.80%	−0.60%	+0.19%
PredRNN & PhyCell connection	−4.51%	−1.20%	+0.16%

without and once with the skip connection in the physical branch (described in section 5.3).

The results in Table 5.5 show that in both experiments PhyDNet with PredRNN did better in all three metrics. The improvements become more noticeable in long-term predictions, as evident in Figure 5.8, where these networks achieved the best performance for predictions between two and three hours into the future. Visually, the predictions in Figure 5.6 have slightly more accurate high-intensity areas but are quite similar to the baseline model. Figure 5.7 shows the partial predictions from only the ConvLSTM and PredRNN cells.

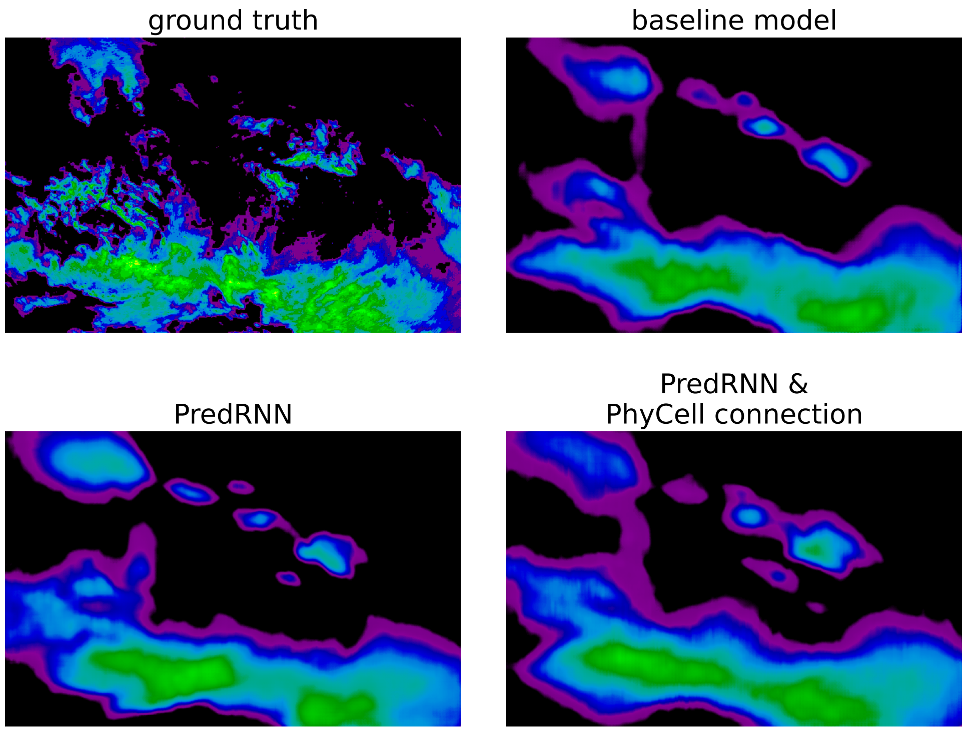

Figure 5.6: Comparison of predictions of PhyDNet with PredRNN.

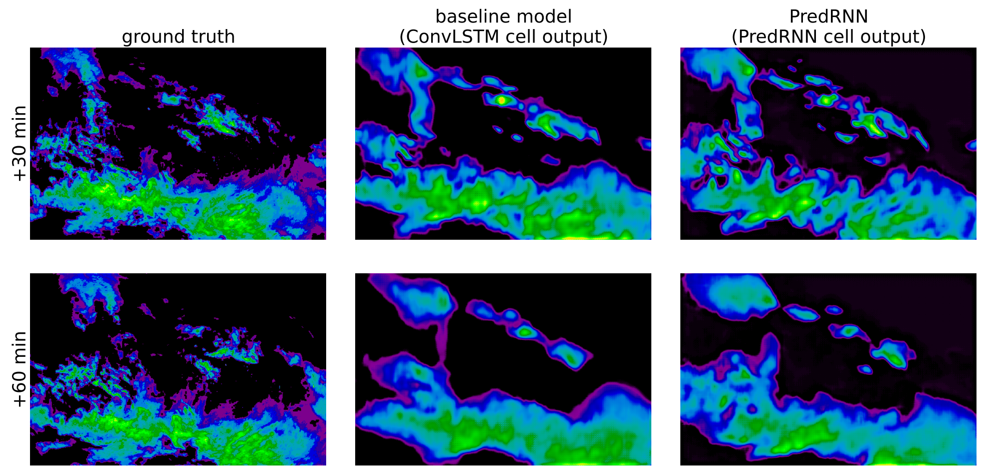

Figure 5.7: Comparison of partial predictions from the ConvLSTM cell and PredRNN cell.

5.6 Summary of experiments

The conducted experiments showed that modifying the residual branch of PhyDNet has a relatively small effect on the overall performance of the network. Table 5.6 compares the best network from each experiment on the test dataset. The experiment which swapped ConvLSTM for PredRNN and added a skip connection to the PhyCell achieved the best MAE, MSE, and SSIM. While some experiments did better than the baseline model, the improvements were generally small.

However, the differences become more significant when we extend the length of the predicted sequence past the trained setting of 60 minutes. The charts in Figure 5.8 show that with PredRNN and the PhyCell skip connection, the quality of predictions doesn't deteriorate as fast as in other experiments. The three experiments with these modifications were the best in all metrics for more long-term predictions. Both of these modifications also performed better in MSE, MAE, and SSIM in the 60-minute predictions.

Visually, the most interesting change was seen with the residual connection for the PhyCell (Figures 5.3 and 5.4), which confirms the quantitative results. The predictions were more detailed and accurate and the model could more precisely identify areas with intense precipitation.

Interestingly, replacing ConvLSTM with SA-ConvLSTM model resulted in the betterment of MSE, MAE, and SSIM, even though the SA-ConvLSTM showed only minimal signs of learning. However, when assessing the long-term performance, the results are less encouraging, especially in SSIM, where this network falls behind the baseline model after the 2-hour mark. Combined with the computational limitations, SA-ConvLSTM didn't show as a promising model for the residual branch.

Table 5.6: Performance of each experiment on the test dataset for 60-minute predictions.

Model	MSE	MAE	SSIM
baseline model	0.001657	0.011011	0.914045
connected branches (addition)	0.001643	0.011197	0.915032
separate decoders	0.001621	0.011077	0.914614
PhyCell connection	0.001653	0.010985	0.915047
SA-ConvLSTM + GroupNorm	0.001623	0.010979	0.915027
PredRNN & PhyCell connection	**0.001582**	**0.010878**	**0.915549**

Overall, the experiments indicate two potential improvements to PhyDNet: adding a skip connection to the physical branch and using PredRNN instead of ConvLSTM, both of which consistently achieved better performance in all three metrics for 60- and 180-minute predictions and generated more detailed images. These could serve as a basis for further research.

The results of other experiments are mixed. While some experiments did improve the performance in one metric, they also did worse in others. In some cases (namely with SA-ConvLSTM and separate decoders), the modifications burdened the model with increased computational requirements, making training slower and more difficult, while delivering little to no performance gain.

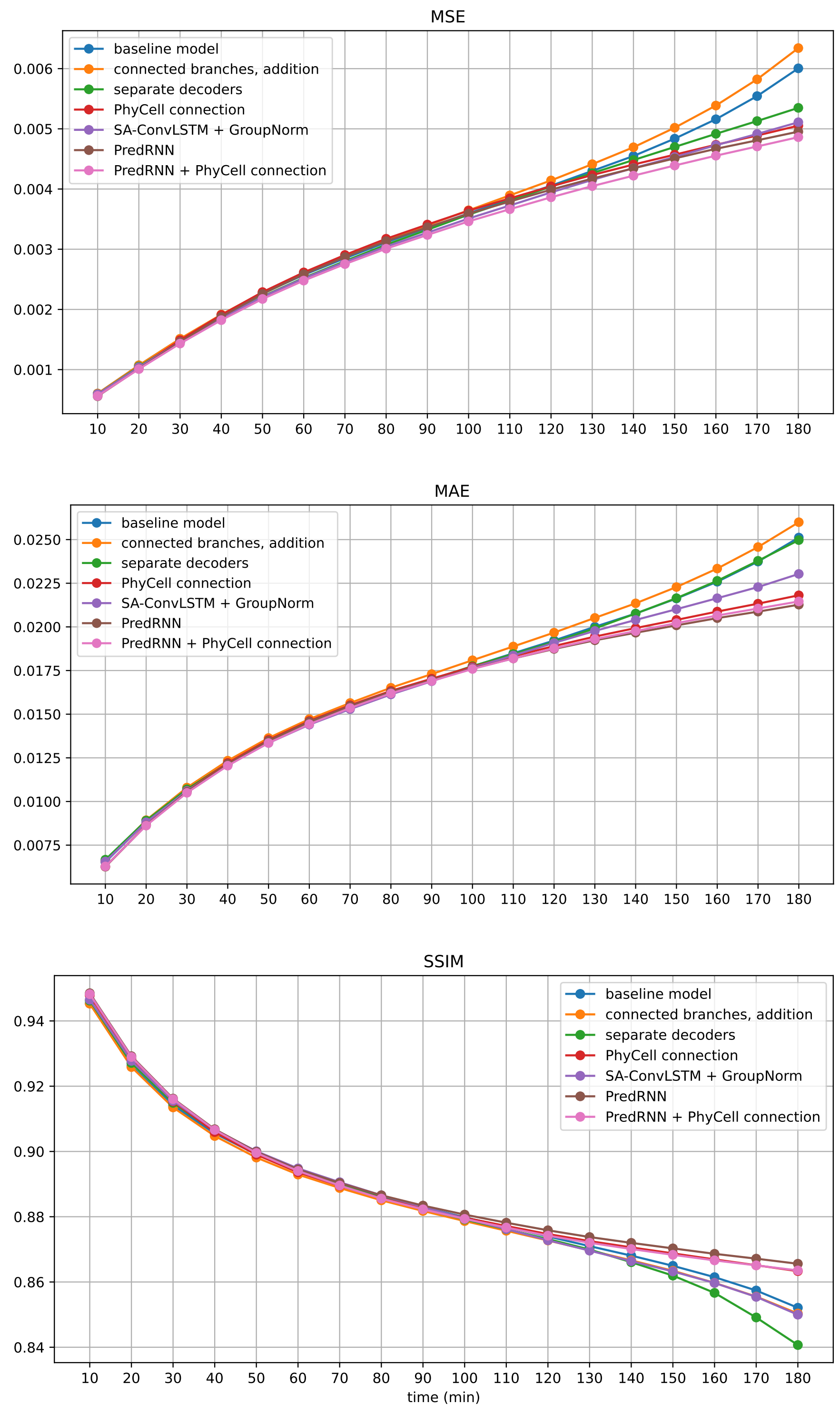

Figure 5.8: Comparison of the average MSE, MAE and SSIM of the prediction sequence over time on test dataset. The predictions were set for 3 hours.

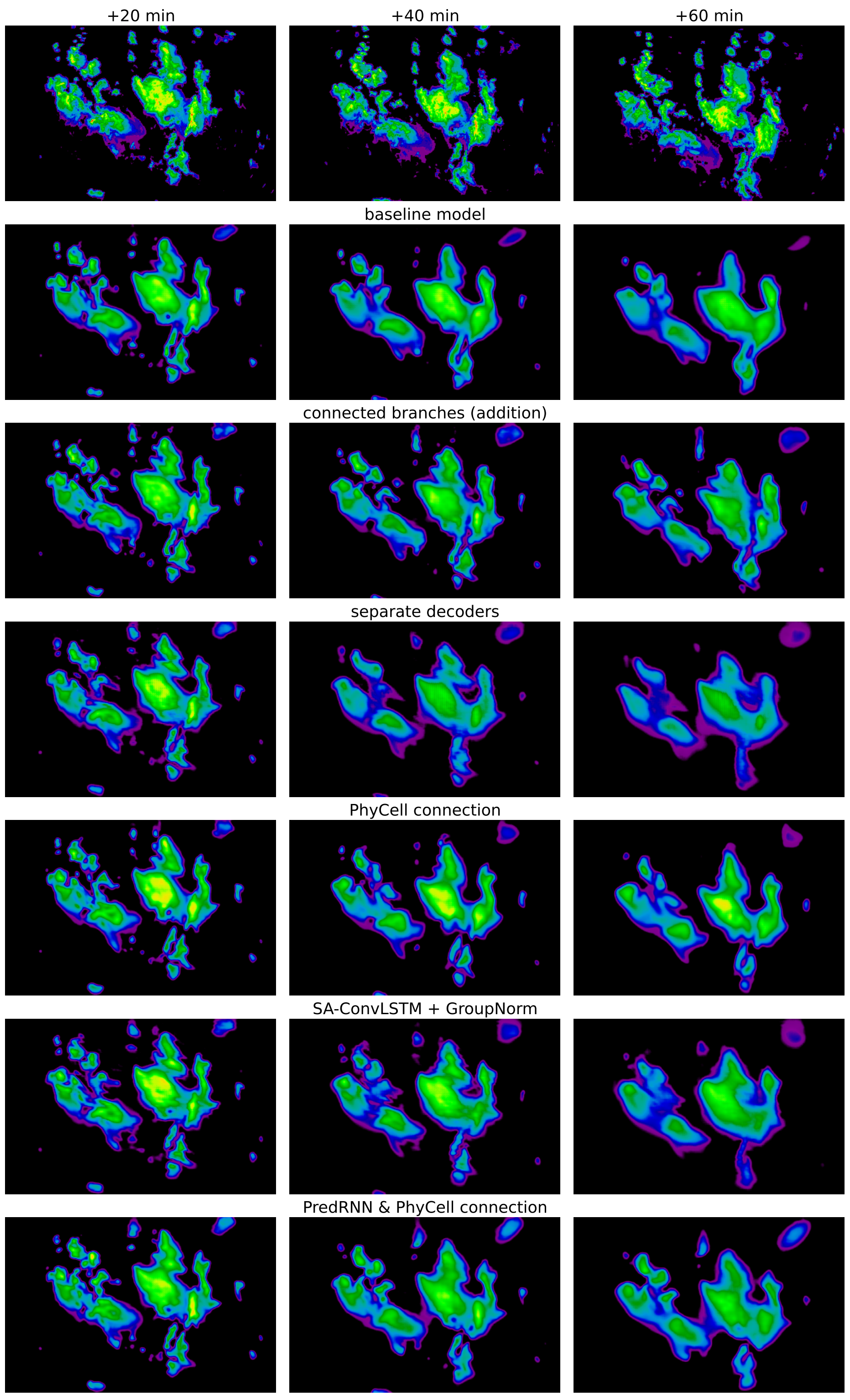

Figure 5.9: Sample predictions from select networks from each experiment. The top row represents the ground truth.

Conclusion

Precipitation nowcasting improves our everyday lives by providing important information about upcoming rainfall, thus protecting many people from severe storms. Deep learning models can be used for precipitation nowcasting based on radar echo maps. The goal of this thesis was to improve the predictive capabilities of one particular model, PhyDNet, by experimenting with different neural network architectures for spatiotemporal prediction.

Chapter 1 introduced the theory and necessary concepts behind deep learning models for spatiotemporal prediction. A detailed description of many of these models was presented in 2. Chapter 3 gave an overview of methods for precipitation nowcasting.

Chapter 4 explored the radar echo dataset and described the training procedure and common hyperparameters for all implemented models. In chapter 5, multiple modifications of the PhyDNet model were trained on the radar echo dataset. These modifications ranged from small enhancements, such as adding skip connections or connecting independent subnetworks together, to experiments with changing the ConvLSTM model inside PhyDNet for different models, specifically SA-ConvLSTM and PredRNN. The experiments were evaluated and discussed.

Two of our modifications to PhyDNet achieved promising results both in the quantitative performance and the quality of the predictions: replacement of ConvLSTM with PredRNN and the addition of a skip connection to the physical branch of PhyDNet. Other experiments had varying results with some falling short of the performance of the baseline model and some delivering slight improvements in one metric but often worsening in another. To conclude, the aim of the thesis was successfully fulfilled – we explored several alterations to PhyDNet when used for precipitation nowcasting and identified two encouraging approaches, which can serve as a basis for further exploration.

Bibliography

1. SHI, Xingjian et al. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2015.
2. SHI, Xingjian et al. Deep Learning for Precipitation Nowcasting: A Benchmark and A New Model. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, vol. 30.
3. NIELSEN, Michael A. *Neural Networks and Deep Learning* [online]. Determination Press, 2015 [visited on 2023-04-03]. Available from: `http://neuralnetworksanddeeplearning.com/`.
4. KINGMA, Diederik P.; BA, Jimmy. Adam: A Method for Stochastic Optimization. 2017. Available from arXiv: `1412.6980 [cs.LG]`.
5. IOFFE, Sergey; SZEGEDY, Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015, vol. 37, pp. 448–456.
6. BA, Jimmy Lei; KIROS, Jamie Ryan; HINTON, Geoffrey E. Layer Normalization. 2016. Available from arXiv: `1607.06450 [stat.ML]`.
7. WU, Yuxin; HE, Kaiming. Group Normalization. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.
8. ZHANG, Aston et al. Dive into Deep Learning. 2023. Available from arXiv: `2106.11342 [cs.LG]`.
9. KARPATHY, Andrej. *Convolutional Neural Networks* [online]. 2016. [visited on 2023-04-03]. Available from: `https://cs231n.github.io/convolutional-networks/`.
10. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning* [online]. MIT Press, 2016 [visited on 2023-04-04]. ISBN 978-0262035613. Available from: `http://www.deeplearningbook.org`.

11. REYNOLDS, Ann H. *Convolutional Neural Networks (CNNs)* [online]. 2019. [visited on 2023-04-04]. Available from: `https://anhreynolds.com/blogs/cnn.html`.
12. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, 2015, pp. 234–241. Available from DOI: `10.1007/978-3-319-24574-4_28`.
13. OLAH, Christopher. *Understanding LSTM Networks* [online]. 2015. [visited on 2023-04-05]. Available from: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.
14. CHO, Kyunghyun et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. 2014. Available from arXiv: `1406.1078 [cs.CL]`.
15. SUTSKEVER, Ilya; VINYALS, Oriol; LE, Quoc V. Sequence to Sequence Learning with Neural Networks. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2014, vol. 27.
16. ADALOGLOU, Nikolas. Intuitive Explanation of Skip Connections in Deep Learning [online]. 2020 [visited on 2023-04-06]. Available from: `https://theaisummer.com/skip-connections/`.
17. BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. 2016. Available from arXiv: `1409.0473 [cs.CL]`.
18. VASWANI, Ashish et al. Attention Is All You Need. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, vol. 30.
19. WANG, Yunbo et al. PredRNN: Recurrent Neural Networks for Predictive Learning Using Spatiotemporal LSTMs. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, vol. 30, pp. 879–888.
20. GAO, Zhihan et al. Earthformer: Exploring space-time transformers for earth system forecasting. In: *NeurIPS 2022*. 2022.
21. GUEN, Vincent Le; THOME, Nicolas. Disentangling Physical Dynamics from Unknown Factors for Unsupervised Video Prediction. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.
22. LIN, Zhihui et al. Self-Attention ConvLSTM for Spatiotemporal Prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2020, vol. 34, pp. 11531–11538. Available from DOI: `10.1609/aaai.v34i07.6819`.

23. WANG, Yunbo et al. PredRNN: A Recurrent Neural Network for Spatiotemporal Predictive Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2021, vol. 45, pp. 2208–2225. Available from DOI: `10.1109/TPAMI.2022.3165153`.

24. WANG, Yunbo et al. PredRNN++: Towards A Resolution of the Deep-in-Time Dilemma in Spatiotemporal Predictive Learning. In: *Proceedings of the 35th International Conference on Machine Learning*. 2018, pp. 5123–5132.

25. WORLD METEOROLOGICAL ORGANIZATION. *Guidelines for Nowcasting Techniques*. WMO, 2017. ISBN 978-92-63-11198-2. Available also from: `https://library.wmo.int/doc_num.php?explnum_id=3795`.

26. BUREAU OF METEOROLOGY OF THE AUSTRALIAN GOVERNMENT. *How Radar Works* [online]. [N.d.]. [visited on 2023-04-26]. Available from: `http://www.bom.gov.au/australia/radar/about/what_is_radar.shtml`.

27. CZECH HYDROMETEOROLOGICAL INSTITUTE. *ČHMÚ nowcasting webportal* [online]. [N.d.]. [visited on 2023-04-28]. Available from: `https://www.chmi.cz/files/portal/docs/meteo/rad/inca-cz/short.html`.

28. LI, L.; SCHMID, W.; JOSS, J. Nowcasting of Motion and Growth of Precipitation with Radar over a Complex Orography. *Journal of Applied Meteorology and Climatology*. 1995, vol. 34, no. 6, pp. 1286–1300. Available from DOI: `https://doi.org/10.1175/1520-0450(1995)034<1286:NOMAGO>2.0.CO;2`.

29. BOWLER, Neill E.; PIERCE, Clive E.; SEED, Alan W. STEPS: A probabilistic precipitation forecasting scheme which merges an extrapolation nowcast with downscaled NWP. *Quarterly Journal of the Royal Meteorological Society*. 2006, vol. 132, no. 620, pp. 2127–2155. Available from DOI: `https://doi.org/10.1256/qj.04.100`.

30. CHOMA, Matej. *Improving Deep Learning Precipitation Nowcasting by Using Prior Knowledge*. 2022. Master's Thesis. Czech Technical University in Prague, Faculty of Information Technology.

31. NAMGUNG, Min. *SA-ConvLSTM* [online]. GitHub, 2022 [visited on 2023-04-23]. Available from: `https://github.com/MinNamgung/sa_convlstm`.

Appendix A

Acronyms

CNN Convolutional neural network

GPU Graphics processing unit

GRU Gated recurrent unit

LSTM Long short-term memory

MAE Mean absolute error

MSE Mean squared error

NWP Numerical weather prediction

PDE Partial differential equation

ReLU Rectified linear unit

RGB Red green blue

RNN Recurrent neural network

SA Self-attention

SAM Self-attention memory

SSIM Structural similarity index measure

Appendix B

Contents of the attachment

```
configs .................. directory with network configurations
├── sample_data .......... sample data for visualizations
├── src
│   ├── model
│   │   ├── layers
│   │   │   ├── covlstm ........... directory with ConvLSTM implementation
│   │   │   ├── phycells .......... directory with PhyCell implementations
│   │   │   ├── predrnn ........... directory with PredRNN implementation
│   │   │   ├── sacovlstm ......... directory with SA-ConvLSTM implementation
│   │   │   └── phydnet_layers.py . encoder-decoder implementation
│   │   ├── base_metric.py ........ base class for metrics
│   │   ├── base_model.py ......... base class for models
│   │   ├── loss.py ............... loss functions
│   │   ├── metric_detail.py ...... implementations of metrics
│   │   └── phydnet.py ............ PhyDNet implementation
│   ├── utils ..................... directory with utility functions
│   ├── dataset.py ................ dataset loader
│   ├── test.py ................... testing procedure
│   ├── tracker.py ................ custom metric tracker
│   └── train.py .................. training procedure
├── text .......................... directory with the thesis text
│   ├── latex ..................... directory with LaTeX source codes and images
│   └── thesis.pdf ................ text of the thesis in PDF format
├── env.yml ....................... specifiaction of Conda environment
├── README.md ..................... file with instruction about the source code
├── train.py ...................... training script
├── trained_model.ckpt ............ checkpoint file with trained baseline model
└── visualize.ipynb ............... example visualizations of predictions
```