FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Quantification of uncertainty of precipitation nowcasting
Student:	Pavel Chudomel
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

Deep learning models are currently very successful in precipitation nowcasting. Currently, the uncertainty of such prediction is usually not monitored or analyzed [2]. We will apply the MC-dropout [3] method to create an ensemble of different predictions.

1) Survey current literature on uncertainty quantification in deep learning (with a focus on MC-Dropout).
2) Understand and describe the used dataset (weather radar data) provided by Meteopress.
3) Train the currently used model (PhyDNet [1]) with dropout and compare the results to PhyDNet without dropout, the MSE, MAE, and SSIM metrics will be compared.
4) Apply the MC-Dropout in various parts of PhyDNet (encoder, decoder, phyCell) and compare the results, using metrics like a prediction interval coverage probability (PICP) and mean prediction interval width.
5) Analyze the results and describe the whole method.

[1] Disentangling Physical Dynamics from Unknown Factors for Unsupervised Video Prediction, Le Guen et al.
[2] Uncertainty-Aware Deep Learning Architectures for Highly Dynamic Air Quality Prediction, Mokhtari et al.
[3] Dropout as a Bayesian approximation: representing model uncertainty in deep learning, Y. Gal, and Z.Ghahramani, "D

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 31 October 2022 in Prague.

Bachelor's thesis

QUANTIFICATION OF UNCERTAINTY OF PRECIPITATION NOWCASTING

Pavel Chudomel

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Mgr. Petr Šimánek
May 11, 2023

Contents

List of Figures

List of Tables

I would like to thank to my supervisor, Mgr. Petr Šimánek for provided expertise, insightful feedback, and patience.
I am grateful to Faculty of Information Technology for providing me with the necessary resources and facilities to carry out this research.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Prague on May 11, 2023 .....

Abstract

Uncertainty quantification is an important aspect of deep learning. Since most of the models cannot explain their predictions, it is essential to express a measure of their uncertainty. In this thesis we discover a multiple ways of achieving this. We implemented MC dropout and quantile regression into the PhyDNet model, a state-of-the-art prediction model for spatio-temporal phenomena — specifically, precipitation measurement in our case. MC dropout proves to be a flexible method, yet it requires meticulous configuration. Quantile regression offers remarkable coverage probabilities, but we found out that this method can underestimate the uncertainty in low precipitation density areas. We present an elaborate overview of these methods and provide readers with a valuable guide for selecting the most suitable method depending on their specific requirements and goals.

Keywords deep learning, uncertainty quantification, PhyDNet, precipitation nowcasting, neural networks

Abstrakt

Hodnocení nejistoty je důležitým aspektem hlubokého učení. Protože většina modelů nedokáže vysvětlit své predikce, je vhodné vyjádřit míru jejich nejistoty. V této práci se zabýváme způsoby jak tohoto dosáhnout. Implementovali jsme metody MC dropout a kvantilové regrese do modelu PhyDNet, což je aktuálně nejmodernější predikční model pro časoprostorové jevy — v našem případě míru srážek. MC dropout se ukazuje být flexibilní metodou, která však potřebuje pečlivou konfiguraci. Kvantilová regrese nabízí pozoruhodnou pravděpodobnost pokrytí, ale zjistili jsme, že tato metoda může podceňovat nejistotu v oblastech nízké hustoty srážek. Představujeme podrobný přehled těchto metod a dáváme tím čtenáři do rukou cennou příručku pro výběr nejvhodnější metody v závislosti na konkrétních požadavcích a cílech.

Klíčová slova hluboké učení, hodnocení nejistoty, PhyDNet, předpověď srážek, neuronové sítě

Abbreviations

RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
PDE	Partial Differential Equation
MSE	Mean Squared Error
MAE	Mean Absolute Error
SSIM	Structural Similarity Index Measure
MPIW	Mean Prediction Interval Width
PICP	Prediction Interval Coverage Probability
MC Dropout	Monte Carlo Dropout
BNN	Bayesian Neural Network
SWAG	Stochastic Weight Averaging with Gaussian approximation
SGD	Stochastic Gradient Descent
SWA	Stochastic Weight Averaging

Introduction

Recent years have brought a lot of innovation in the field of predictive neural networks. Being very strong in representation learning, deep neural networks have become the most popular way to tackle difficult problems in many different areas. One of them is forecast, such as precipitation nowcasting.

The model we work with is PhyDNet[1], introduced in 2020, which has shown state-of-the-art results with its spatio-temporal predictions. It disentangles the underlying physical dynamics of unknown factors. In this work, we will have a look "under the hood" of the model, discovering how it works.

The main focus of this work is measuring uncertainty in the model. This aspect is crucial but frequently disregarded in the prediction process. A prediction with high uncertainty might hold minimal importance, so it is necessary to establish a method for determining it. We aim to explore various approaches for this purpose, with a particular focus on MC Dropout. This simple technique utilizes dropout, typically employed for regularization, to generate an ensemble of diverse predictions. We will try to understand why this approach can be better than other, more complex ones.

The practical aim of this work is to implement the MC Dropout for uncertainty quantification. There are various ways to implement the method, which means we will have to test multiple variants. Along the way, we will also explore how effectively it functions as an additional regularization layer in the PhyDNet model. To gain insight into how it compares to different uncertainty quantification methods, we will attempt to implement quantile regression and use it as such method. This technique estimates the quantiles of the output distribution, rather than predicting the distribution itself.

There is a significant demand for methods to accurately assess the uncertainty of predictions. In developing countries, numerous individuals are affected by extreme weather every year. A few unreliable predictions that fail to materialize may heavily undermine confidence in the model. Another example might be motorsport racing. The teams need a good way to predict the weather to adapt the strategy. Understanding the uncertainty associated with predictions can substantially reduce the risk of losing points due to unforeseen circumstances.

The first chapter of this work will describe the PhyDNet model. We will explore its layers and try to understand the inner workings. Chapter 2 will examine various methods for incorporating uncertainty into prediction models. Chapter 3 will connect these topics and cover everything we need for the final implementation. Finally, chapter 4 will contain a description and analysis of the practical results. Each chapter aims to offer greater understanding and guidance on the implementation process. Chapters 1 and 2 can be used as independent units, each serving as a reference for their respective subjects.

Chapter 1

Deep dive into PhyDNet for spatio-temporal predictions

This chapter describes the architecture of PhyDNet — a state-of-the-art model for spatio-temporal predictions. This model is the core of this thesis so it is necessary to understand it thoroughly. Later in this thesis, we will be using knowledge from this chapter to expand the model with uncertainty quantification.

Spatio-temporal phenomena depend on both space and time. We will represent space as a matrix with time dependent values. Although there are physical laws that govern the behavior of these values, they can be difficult to model accurately due to limitations in our data and understanding of the underlying processes [2].

The task of finding the underlying dependencies in the data is a part of the field of machine learning. In order to make the predictions, we do not look for an explicit expression of these laws, we only need a model that produces a good results according to some given criteria. This model will be incorporating its own laws, and these should accomplish to describe the real world phenomena rather well.

In recent years, deep learning techniques have shown great promise in the field of spatio-temporal prediction. In particular, the PhyDNet model [1] has achieved state-of-the-art results by disentangling the physical dynamics of the system from other unknown factors.

1.1 Recurrent neural networks

Neural networks started in 1943, when McCulloch and Pitts presented their mathematical model of a neuron [3]. Since then, researchers have been working to develop networks that can more closely mimic the behavior of the human brain.

Simple neural networks only work with fixed-size inputs and their output depends only by the current input. Recurrent neural networks (RNNs), on the other hand, are designed to work with sequential data, mapping input sequences to output sequences [4]. This makes them useful for spatio-temporal prediction, used to predict how the system evolves over time.

A recurrent neural network uses internal state, referred to as hidden state, taken into account when evaluating. This state is updated with every input and passed to the next time step, as seen in Figure 1.1. This not only solves the fixed-size input problem, but also offers a short-term memory to the network, represented by a hidden state. We previously mentioned imitating the human brain so some kind of memory is vital. The hidden state of RNN is often used as an output.

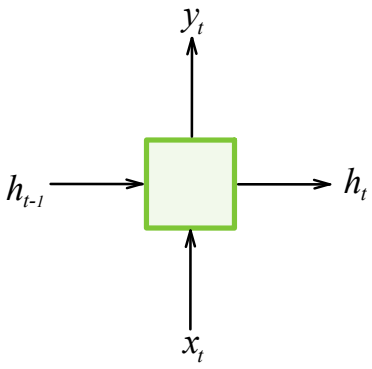

Figure 1.1 Recurrent neural network.

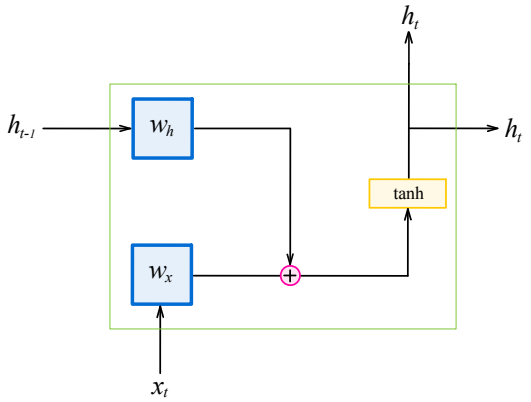

Figure 1.2 RNN with a single activation layer.

An RNN have a form of a chain of repeating modules [5]. These modules have a simple structure consisting of a single non-linear layer, such as single tanh layer. A recurrent neural network can be expressed mathematically [6] as

$$\mathbf{h}_t = F(\mathbf{h}_{t-1}, \mathbf{x}_t, \theta),$$

where $\mathbf{h}_{t-1}$ is current state, $\mathbf{x}_t$ is the input and θ is the collection of used parameters. In a simple RNN, the full expression may be

$$\mathbf{h}_t = \sigma(\mathbf{W}_\mathrm{h}\mathbf{h}_{t-1} + \mathbf{W}_\mathrm{x}\mathbf{x}_t + \mathbf{b}),$$

where σ stands for non-linear function and $\mathbf{W}_\mathrm{rec}$, $\mathbf{W}_\mathrm{in}$ and bias $\mathbf{b}$ are parameters obtained by training the network, referred to collectively as θ. The mathematical expression is depicted in Figure 1.2. The output of the cell is passed in the next time step.

There are two important problems with the training of recurrent neural networks. The vanishing and the exploding gradient problem, detailed in [7]. These problems make recurrent neural networks hard to train properly [6], resulting into loss of learnt dependencies with increasing duration of them being captured.

1.1.1 Long short-term memory

One method addressing the shortcomings of recurrent neural networks is long short-term memory (LSTM) [8]. Their design was later improved by adding forget gates and since then, many variants of this architecture appeared. We will first discuss what is often referred to as vanilla LSTM [9]. By addressing the vanishing and the exploding gradient problem, these networks are able to store information over longer period of time [8].

Like classic RNNs, LSTMs have a chain-like structure. Modules in this structure consist of four neural network layers. We can split it into three parts. [5]

First part of the cell, the forget gate layer, decides what information will be discarded from the cell state.The mathematical expression for this layer is

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f).$$

The $[\mathbf{h}_{t-1}, \mathbf{x}_t]$ is a concatenation of the input and the output from previous cell, $\mathbf{W}_f$ is a matrix of parameters of the forget cell and $\mathbf{b}_f$ is its bias. σ is a sigmoid function — the result of this gate will be a matrix of values between 0 and 1. This tells how much of the current cell state is

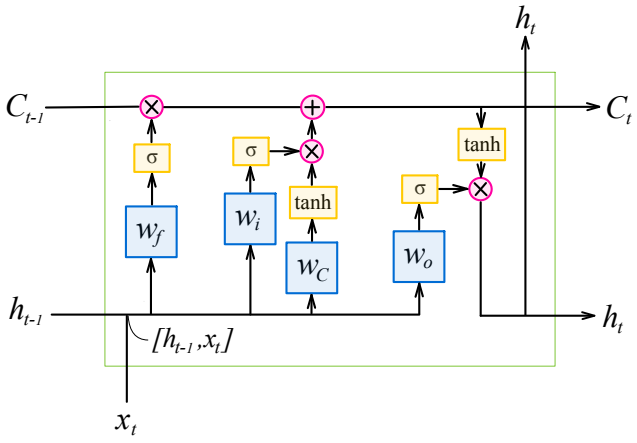

Figure 1.3 LSTM cell.

passed on. For example, if $f_t = \mathbf{0}$ where $\mathbf{0}$ is a zero matrix, the whole cell state will be forgotten (the cell state will be a zero matrix). [5]

Second part presents the information that will be stored in the cell state. This part consists of two parallel neural network layers, the first one is used to create the information and the other one, called the input gate, tells how much of it is actually stored. Mathematically [5]:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C)$$

With that, we have all we need to update the cell state [5]:

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t$$

We use $\odot$ for Hadamard product (element-wise matrix multiplication).

The third layer is used to obtain the output of the cell. It consists of another sigmoid gate which is multiplied by the updated cell state put through tanh. We can see that all of the gates introduced use the same structure — we only change the values of parameters. [5]

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$

The LSTM cell scheme is depicted in Figure 1.3.

Apart of addressing the vanishing/exploding gradient issues, we can see that LSTMs effectively manage the information stored in the hidden state. That is a significant benefit of these networks.

PhyDNet is using a variant of LSTM — ConvLSTM [10], which adds convolution operation into the vanilla LSTM described above. This is extremely useful for processing sequential data with spatial information.

1.1.2 ConvLSTM

Convolution operation is closely related to operations with image data. The reason is "*that in images, the interesting "combinations of features" (pixels) tend to come from pixels that are*

close together in the image" [11]. Convolutional layers are used for capturing spatial features in images, whereas ConvLSTM is a powerful technique to model spatio-temporal dependencies that can be used to predict video data.

ConvLSTM can be expressed mathematically [10]:

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf} * \mathbf{x}_t + \mathbf{W}_{hf} * \mathbf{h}^{\mathbf{r}}_{t-1} + \mathbf{b}_f)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi} * \mathbf{x}_t + \mathbf{W}_{hi} * \mathbf{h}^{\mathbf{r}}_{t-1} + \mathbf{b}_i)$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_{xC} * \mathbf{x}_t + \mathbf{W}_{hC} * \mathbf{h}^{\mathbf{r}}_{t-1} + \mathbf{b}_C)$$

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo} * \mathbf{x}_t + \mathbf{W}_{ho} * \mathbf{h}^{\mathbf{r}}_{t-1} + \mathbf{b}_o)$$

$$\mathbf{h}^{\mathbf{r}}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$

where $\mathbf{h}^{\mathbf{r}}_t$ is the output from the cell. We use this notion because in the PhyDNet model, ConvLSTM is used to predict the residual component. This will be discussed later in this chapter.

Note that an optional expansion adds peephole connections to the gates. The term added might take form $\mathbf{W}_{cf} \odot \mathbf{c}_{t-1}$, in the case of forget gate layer, and it allows the gates to look at the cell state and use the information. Although the ConvLSTM variant with peephole connections has become quite popular, in our case, we will be utilizing the standard ConvLSTM architecture without the peephole connections.

We can see that the leap from the vanilla LSTM is not very big. Note that addition and concatenation are linked. If we have input $\mathbf{x}$, hidden state $\mathbf{h}$ and respective parameter $\mathbf{W}$, we can show the following:

$$\mathbf{W}[\mathbf{x}, \mathbf{h}] = \mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{h}$$

where $\mathbf{W}$ is split horizontally into $\mathbf{W}_1$ and $\mathbf{W}_2$. This means that we only replace matrix multiplication with convolution operation.

1.1.3 PhyCell

PhyDNet's authors proposed PhyCell [1], a physical recurrent cell that performs predictions constrained by partial differential equations (PDEs) in a latent space denoted as $\mathcal{H}$. We can model PhyCell's dynamics using PDE response function $\boldsymbol{\mathcal{M}}_p(\mathbf{h}^{\mathbf{p}}, \mathbf{u})$:

$$\boldsymbol{\mathcal{M}}_p(\mathbf{h}^{\mathbf{p}}, \mathbf{u}) = \Phi(\mathbf{h}^{\mathbf{p}}) + \mathcal{C}(\mathbf{h}^{\mathbf{p}}, \mathbf{u})$$

where $\mathbf{h}^{\mathbf{p}}$ is latent representation of the physical system up to current time step t and thus can be expressed as a function $\mathbf{h}^{\mathbf{p}}(\mathrm{x}, t) \in \mathcal{H}$, where x is set of coordinates. We denote $\mathbf{u}$ as the input stream frame at current time step t, $\mathbf{u} = \mathbf{u}(\mathrm{x}, t)$. The function is divided into two parts — the physical predictor $\Phi(\mathbf{h}^{\mathbf{p}})$ and the correction term $\mathcal{C}(\mathbf{h}^{\mathbf{p}}, \mathbf{u})$. [1]

The physical predictor is based on an assumption of good expressibility of laws of nature using PDEs. It takes the following form [1]:

$$\Phi(\mathbf{h}^{\mathbf{p}}(\mathrm{x}, t)) = \sum_{i,j:i+j\leq q} c_{ij} \frac{\partial^{i+j}\mathbf{h}^{\mathbf{p}}}{\partial \mathrm{x}_i \partial \mathrm{x}_j}(\mathrm{x}, t)$$

where q is given differential order. This can be used to model linear PDEs, which many equations in physics take form of. One of the famous examples is the heat equation. [1]

In the implementation, the physical predictor approximates the solution of the PDEs by using multiple convolutional neural networks to extract features from the hidden state. These features are expected to correspond to the dynamics expressed in the hidden state. The features are then

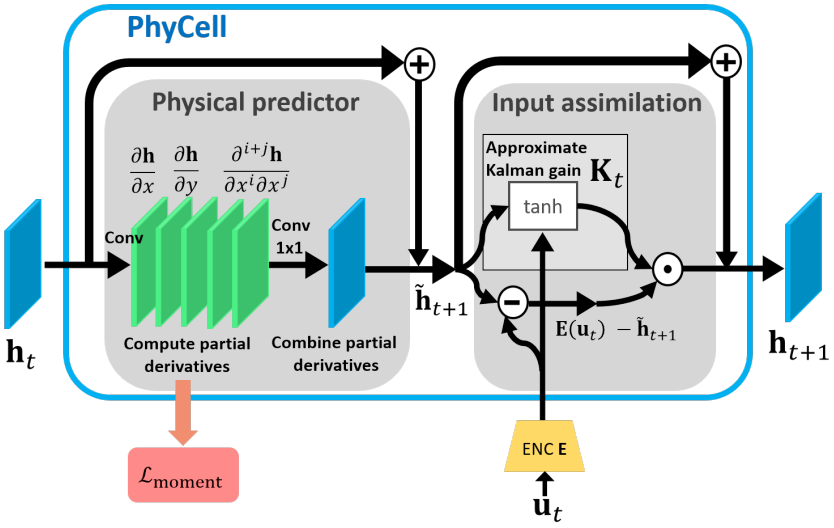

Figure 1.4 Architecture of PhyCell. [1]

flattened into a one-dimensional vector that is fed through a fully connected layer, generating a matrix $\tilde{\mathbf{h}}^{\mathbf{P}}$ in the latent space. This matrix represents the updated hidden state and is then passed into the correction term. [1]

Correction term is modeled as follows [1]:

$$\mathcal{C}(\mathbf{h}^{\mathbf{P}}, \mathbf{u}) = \mathbf{K}(t, \mathrm{x}) \odot [\mathbf{E}(\mathbf{u}(t, x)) - (\mathbf{h}^{\mathbf{P}}(t, \mathrm{x}) + \Phi(\mathbf{h}^{\mathbf{P}}(t, \mathrm{x}))]$$

The correction term is used to update the hidden state using the information about the difference between the updated latent hidden state and the new observed output. This difference is multiplied by gating factor $\mathbf{K}(t, \mathrm{x})$, which can be understood as an approximate Kalman gain. The approximation is done by using convolutional neural networks.

$$\mathbf{K}(t, \mathrm{x}) = \tanh(\tilde{\mathbf{h}}^{\mathbf{P}}(t, \mathrm{x}) * \mathbf{W}_{kh} + \mathbf{E}(\mathbf{u}(t, \mathrm{x})) * \mathbf{W}_{ku} + \mathbf{b}_k)$$

As mentioned earlier, the approximate Kalman gain is used as a gating factor. This means that if this value is $\mathbf{0}$, then the input is not accounted and if it is $\mathbf{1}$, then the latent dynamics in the hidden state is reset. This is similar to LSTMs described earlier. [1]

In summary, PhyCell operates in two steps. The first step is the prediction, taking form:

$$\tilde{\mathbf{h}}^{\mathbf{P}}_{t+1} = \mathbf{h}^{\mathbf{P}}{}_t + \Phi(\mathbf{h}^{\mathbf{P}}{}_t)$$

The other step is used as a correction using the input in the current step encoded into the latent space $\mathbf{E}(\mathbf{u_t})$:

$$\mathbf{h}^{\mathbf{P}}{}_{t+1} = \tilde{\mathbf{h}}^{\mathbf{P}}_{t+1} + \mathbf{K}_t \odot (\mathbf{E}(\mathbf{u_t}) - \tilde{\mathbf{h}}^{\mathbf{P}}_{t+1})$$

The structure of PhyCell is depicted in Figure 1.4.

Note that the physical predictor operates without a knowledge about current input. It predicts a new hidden state based on the current one, and this new hidden state is updated using the input in the correction term. This architecture allows us to predict multiple time steps into future without having direct knowledge of the input at those future time steps. The quality of these predictions depends on how close our PDE approximation is to the real-world laws. [1]

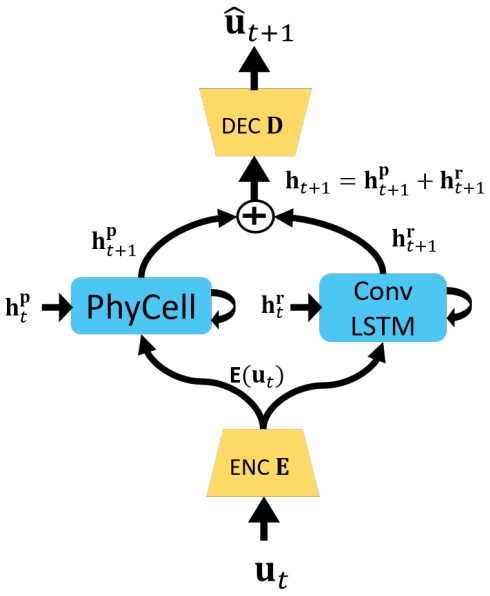

Figure 1.5 Architecture of PhyDNet. [1]

1.2 PhyDNet

Now that we have covered all the necessary building blocks, we can describe the PhyDNet architecture, which was introduced in [1] by Vincent Le Guen and Nicholas Thome in 2020. This architecture has demonstrated state-of-the-art results in video prediction by disentangling the physical dynamics expressible by using PDEs from unknown complementary information. [1]

We will first cover the encoder-decoder architecture used to convert the input data into a latent space and then we will discuss the principle of the disentanglement — key feature in PhyDNet.

1.2.1 The latent space — encoder-decoder architecture

PhyDNet operates in a latent space $\mathcal{H}$. This is done by using the encoder-decoder architecture. The input data $\mathbf{u}_t$ is converted into $\mathbf{E}(\mathbf{u}_t) \in \mathcal{H}$. This representation provides more compressed and abstract form of the input. The reason why we want to encode is because physical laws can't be used to describe the motion of pixels. The PhyDNet model assumes that this pixel representation can be converted into a latent space where some laws can be applied. [1]

The encoder is implemented by using convolutional neural networks. This extracts important features of the input and compresses them into a feature map in the latent space $\mathcal{H}$. We will be using $\mathbf{E}(\mathbf{u}_t) \in \mathcal{H}$ to denote the encoded input. [1]

1.2.2 Disentanglement

The main part of the PhyDNet network is the two-branch architecture which disentangles the physical dynamics from the unknown factors. The encoded input $\mathbf{E}(\mathbf{u}_t)$ is used as an input of two parallel neural networks at once. [1]

The left branch in Figure 1.5 is used to model the latent representation $\mathbf{h}^{\mathbf{p}}$, fulfilling the physical prediction. The prediction is done by approximating the solution of PDEs describing the system behavior using PhyCell, described above. As mentioned, the latent representation of

the physical prediction $\mathbf{h^P}$ is by design the hidden state of the cell. The use of PhyCell leads to use of less parameters as it incorporates the physics of the system as constraints. Another benefit is that by incorporating the physical knowledge, PhyDNet can learn from fewer examples and generalize better. It is like having a built-in intuition. [1]

Since the system of precipitation movement contains too many unknown phenomena to be modelled only using the PDEs, the right branch is used to predict the residual dynamics of the system. This obtains the latent representation of the unknown factors in the system, $\mathbf{h^r}$. Any generic RNN can be used for this task, we use ConvLSTM [10] like the authors of [1]. ConvLSTM is described above. [1]

The latent representation of the final prediction is obtained in the following elegant way:

$$\mathbf{h}_{t+1} = \mathbf{h^P}_{t+1} + \mathbf{h^r}_{t+1}$$

This latent representation should contain the prediction of the current state of the modelled system.

Finally, a deep decoder is used to decode the latent space representation $\mathbf{h}_{t+1}$ to forecast the image, mathematically $\hat{\mathbf{u}}_{t+1} = \mathbf{D}(\mathbf{h}_{t+1})$. This step is implementd by using transposed convolution operation, also referred to as deconvolution. [1] This operation goes in the oppostite direction than convolution, upsampling feature maps. [12]

Chapter 2

Various methods of uncertainty quantification

This chapter provides an overview of the probabilistic methods employed for quantifying uncertainty. It briefly reflects on the significance of uncertainty awareness in decision-making. Two of these methods will be implemented, MC dropout and quantile regression. These two will be put into practice in the next chapter covering the implementation.

Uncertainty in neural networks can arise from various sources. According to [13], there are five main factors:

1. Variability in Real world simulation
2. Error and Noise in Measurement Systems
3. Errors in the Model Structure
4. Errors in the Training Procedure
5. Errors Caused by Unknown Data

We will categorize uncertainty in predictions into two types. The first one is called epistemic uncertainty, which arises due to inaccuracies in the model, or in other words, it reflects what the model does not know. The second type is aleatoric uncertainty, which is inherently present in the observed data [14]. It is hardly surprising that the aleatoric uncertainty will play a crucial role in precipitation nowcasting. The big problem is the chaotic nature of weather, meaning that small inaccuracy in initial conditions grows rapidly. We are unable to measure the perfect initial conditions, so even if we had a perfect model, inaccuracies would quantify over time [15]. Generating an accurate weather forecast is a daunting task due to the multitude of interrelated atmospheric factors and the complex properties of the Earth and its atmosphere [16].

In [17], the authors emphasize the significance of aleatoric uncertainty quantification in the case of large datasets where epistemic uncertainty is lessened and in real-time applications since computationally expensive methods are not necessary. Conversely, they suggest that epistemic uncertainty should be quantified for smaller datasets, where the level of epistemic uncertainty may be greater, and in safety-critical applications where new and unseen data could result in significant issues. It is also emphasized that these uncertainties are not mutually exclusive.

Having knowledge of forecast uncertainty provides an advantage that can lead to better decision-making and risk management. There are numerous practical examples where uncertainty quantification can be beneficial, such as helping farmers mitigate weather-related risks, scheduling launches for space exploration missions and many more.

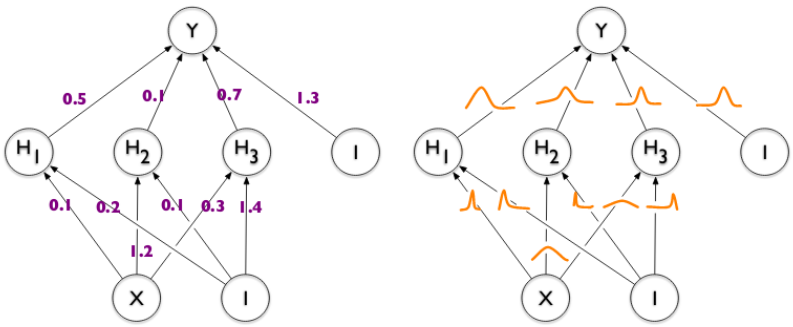

Figure 2.1 Regular neural network on the left uses fixed values for weights. Bayesian neural network on the right treats weights as random variables. [18]

2.1 Bayesian neural networks

Bayesian neural networks (BNN) are a type of probabilistic model that incorporates Bayesian inference. In a BNN, the weights are treated as random variables and assigned a probability distribution over possible values, rather than using fixed-size values. This allows for the incorporation of uncertainty in the model predictions. During each pass of inference, the weights are sampled, generating a non-deterministic prediction, which leads to a slightly different model for each pass. By sampling the weights multiple times and generating multiple predictions, a probability distribution over possible outcomes can be obtained. An illustration is in Figure 2.1. [18]

In the Bayesian paradigm, probability is viewed as a measure of belief in the occurrence of events, where prior beliefs shape the resulting posterior beliefs. This is summarized in Bayes' theorem:

$$P(\mathbf{W} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \mathbf{W})P(\mathbf{W})}{P(\mathcal{D})}$$

where $\mathbf{W}$ is based on hypothesis with prior belief, and $\mathcal{D}$ is data updating our prior belief about $\mathbf{W}$ into the posterior. It is important that we are talking about the Bayesian paradigm, which contrasts with the frequentist paradigm. The Bayes' theorem formula is still true in frequentist interpretation, but $\mathbf{W}$ and $\mathcal{D}$ are considered as the sets of outcomes. Bayesian neural networks can differentiate between epistemic and aleatoric uncertainty, which makes them highly efficient in utilizing data since they do not tend to overfit on small datasets. [19]

Our goal when training a BNN is to infer the posterior distributions over the parameters $\mathbf{W}$, $P(\mathbf{W} \mid \mathcal{D})$. According to Bayes' theorem, we can use the following:

$$P(\mathbf{W} \mid \mathcal{D}) \propto P(\mathcal{D} \mid \mathbf{W})P(\mathbf{W})$$

where $P(\mathcal{D} \mid \mathbf{W})$ is the likelihood of the model given by its parameters $\mathbf{W}$ and $P(\mathbf{W})$ is the prior distribution. [20]

According to [21], "*obtaining explicit posterior densities through Bayesian inference is intractable*". This is true especially for complex models and large datasets because computation of high-dimensional integrals is necessary. However, recent advances in variational inference techniques have made it possible to obtain posterior approximations for BNNs, rendering them computationally feasible. Despite this, BNNs are still generally less efficient than standard neural networks and take longer to converge [21]. Bayesian neural networks achieve state-of-the-art results in uncertainty quantification [22], but for their complexity, we often have to look for alternatives.

2.2 Deep ensembles

Many methods for quantifying uncertainty can be classified as ensemble methods. This section focuses on deep ensembles, which are composed of multiple instances of a single neural network model, each with distinct parameters. This is done by training the models using different initial parameters, which ensures that they are sufficiently independent of each other. [21]

It has been shown that deep ensembles might match other popular probabilistic methods such as Bayesian neural networks or MC dropout [22]. However, there are some drawbacks to this method, such as high computational costs (it may be unfeasible to train large neural network many times) and the need to store many parameter values [21]. An attempt to tackle these disadvantages has been shown in [23], yet this cannot be used for uncertainty quantification, so these disadvantages remain [21].

We already mentioned that many methods of uncertainty quantification can be thought of as ensembles of different models. This is because certain techniques, like Bayesian neural networks or Monte Carlo dropout, produce non-deterministic predictions. During each pass, these techniques generate different versions of the network, which can be seen as different models within the ensemble.

2.3 Stochastic Weight Averaging with Gaussian approximation (SWAG)

One method that has emerged in recent years for the purpose of uncertainty quantification is SWAG (Stochastic Weight Averaging), proposed in [24]. In order to understand how it operates, we must first discuss some concepts used in this method.

2.3.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is an optimization algorithm. Its goal is to minimize a given loss function $\mathcal{L}(\theta)$. This means that it finds the parameters θ that minimize this function. The loss function is usually defined so that it involves the difference between predicted and actual value, thus minimizing this function means making the model's predictions more accurate.

Stochastic gradient descent, unlike batch gradient descent, updates its parameters with every training example. This can be useful for large datasets, as we don't have to go through all the data before updating our model. Because of frequent updates with high variation, the loss function tends to fluctuate heavily. This can make the function deviate from the local minima and help it find the global minima, but it may also cause convergence issues [25]. The fluctuations will prove to be an important part of this method.

If we have gradients of our loss function with respect to the parameters $\nabla_\theta \mathcal{L}(\theta; x_i; y_i)$ calculated, we can update the parameters using the following rule [25]:

$$\theta = \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta; x_i; y_i)$$

where x_i and y_i denote the training example [25]. The update step is repeated until some stopping criterion is met. This criterion can be a maximum number of epochs reached or predefined behavior of recent loss function values, and so on.

2.3.2 Stochastic Weight Averaging (SWA)

The idea behind SWA is remarkably simple. Due to the fluctuations caused during SGD, we can run the SGD on a pre-trained model (this is not necessary, we can run SWA on an untrained

model, but it takes longer to converge [26]) and calculate the parameters by averaging them:

$$\theta_{\text{SWA}} = \frac{1}{T}\sum_{i=1}^{T}\theta_i$$

When using the SWA method to obtain the parameters, we would choose a higher learning rate than usual. This causes higher fluctuations, which means discovering a wider area of the parameter space. After the training, we would use θ_{SWA} as our final parameters. [24]

We can update the parameters after every few epochs:

$$\theta_{\text{SWA}} = \frac{\theta_{\text{SWA}} \cdot n + \theta}{n+1}$$

where n is the number of models used in the average and θ refers to current parameters. This allows us to only remember the last used $\tilde{\theta}_{\text{SWA}}$. [26]

This simple method can achieve improvements in test accuracy over conventional SGD training on a range of state-of-the-art networks. [26]

2.3.3 SWAG for uncertainty quantification

Moving from Stochastic Weight Averaging to Stochastic Weight Averaging with Gaussian approximation is a straightforward extension. In SWA, we obtain multiple samples of the model's parameters that are averaged to get the final parameters. SWAG uses these samples to approximate the parameter distributions as a Gaussian to capture the uncertainty in the parameters. [24]

To implement SWAG, we need to expand the SWA method by calculating the covariance matrix:

$$\Sigma = \frac{1}{T-1}\sum_{i=1}^{T}(\theta_i - \theta_{\text{SWA}})(\theta_i - \theta_{\text{SWA}})^\intercal$$

The covariance matrix captures the statistical properties of the parameters and allows us to approximate their distributions. [24]

When we compare SWAG with the previously discussed BNN, we can see one of its main limitations. BNN uses prior distributions that are updated using the observed data to approximate the posteriors, allowing for a more flexible representation of uncertainty. SWAG assumes the parameters follow Gaussian distribution, which may not always be the case.

Overall SWAG is a relatively new method that has shown promising results on various datasets in comparison with other probabilistic methods. [24]

2.4 Monte Carlo dropout

We will now introduce the first method that we plan to implement, but we first have to discuss some important related topics. Let's first have a look at what dropout is.

2.4.1 Dropout

The knowledge obtained in a deep learning model is given by combinations of features using defined parameters. These parameters are usually obtained during training. Based on these parameters, our model is able to map inputs to outputs in a useful way.

A large model may use the training data to minimize the loss function and find the best settings to make accurate predictions, based on the criteria given by the loss function. The ability of the model to perform well on the training data does not guarantee its performance on

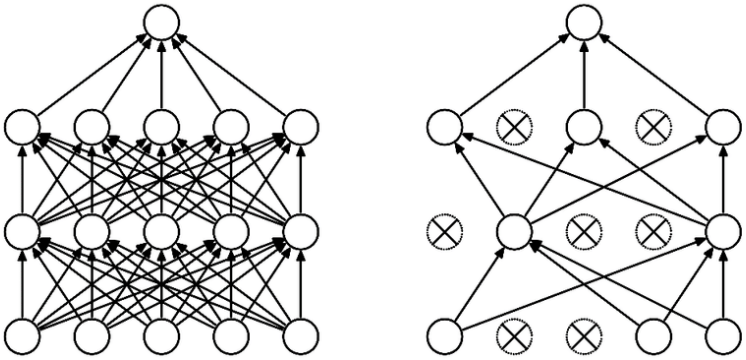

Figure 2.2 Schematic of fully connected network on the left, when we apply dropout, we drop randomly selected units. The illustration is on the right. [28]

any real-world data, even if the training data is a good real-world representation. The ability to generalize is what makes the deep learning models interesting. The measure of generalization can be given by generalization error:

$$\text{generalization error} = \text{test error} - \text{training error}$$

High generalization error means that the model is too specialized for the training data. This is referred to as overfitting. A question of how to minimize this value arises and the answer is by regularizing the parameters. [27]

It is important to understand the idea proposed in [27]. The authors emphasize that the regularization can be split into implicit and explicit and the implicit regularization should be the important factor, giving constraints to the parameters so that they don't become too specialized to the training data. This refers to the architecture of the model or the optimization algorithm used to train it. The explicit regularization is described as follows: "*Explicit regularization may improve generalization performance, but is neither necessary nor by itself sufficient for controlling generalization error.*" [27]

Dropout is one of the methods of explicit regularization. Its key idea is to drop units of neural network randomly, as depicted in Figure 2.2. The basic intuition is that by always using different combinations of parameters, we prevent the units from co-adapting. [28]

The only parameter of dropout is the dropout rate, specifying the probability of the unit being dropped. When picking this value, one of the approaches is to test multiple variants and select the one that minimizes the error using some validation data.

2.4.2 Spatial dropout

Dropout makes the most sense when used on fully connected layers, that's what was intended when the method was proposed in [28]. We, however, want to use dropout in convolutional neural networks.

The straightforward approach would be to use the regular dropout. In [29], the authors argue that the main problem of dropout when used on convolutional networks is the random dropping of features since they are spatially correlated. They state the following: "*When the features are correlated, even with dropout, information about the input can still be sent to the next layer, which causes the networks to overfit. This intuition suggests that a more structured form of dropout is needed to better regularize convolutional networks.*" [29]

The convolutional network learns spatial correlations of features from the input data and maps them into feature maps. A feature map usually represents a single specific feature in the data. As an example of what a feature map can be, we can think of an edge detector, which is

implemented using the convolution operation. As we get deeper, the concept of feature maps becomes very abstract, which enables the network to understand deep dependencies.

What we do by using the spatial dropout instead of the regular one is we do not randomly drop units, but whole feature maps instead. This should prevent the feature maps from co-adapting to the training data and has been shown to increase the model's performance. [30]

Since we work with image data and convolutional networks, we will be using this variant of dropout instead of the regular one.

2.4.3 MC dropout for uncertainty quantification

The MC dropout method was proposed in [31] as a way to tackle the prohibitive computational cost of Bayesian models.

The main idea of MC dropout is to apply dropout during inference, generating an ensemble of sampled outputs that can be used to approximate the distribution of the output. The authors show that MC dropout is mathematically equivalent to an approximation of the probabilistic deep Gaussian processes, a non-parametric, computationally expensive method for uncertainty quantification. [31]

One of the key advantages of MC dropout is its simplicity. It can be easily integrated into existing deep learning frameworks without requiring significant modifications. There is often no need to retrain the model to enable uncertainty quantification. Since dropout is often used as a regularization technique, enabling it during the inference is all we have to do. If dropout wasn't employed during training, a question arise whether or not to retrain the network with it. An argument speaking in favor of retraining the model is the improvement in robustness, making the model better prepared for the dropped units. However, this might also cause decrease in diversity of the output.

After obtaining samples using multiple inference steps, we are able to obtain the confidence interval with a given significance level α:

$$(\mathbf{L}, \mathbf{U}) = \left(\bar{X}_n - \frac{t_{n-1}(\frac{\alpha}{2})s}{\sqrt{n}}, \bar{X}_n + \frac{t_{n-1}(\frac{\alpha}{2})s}{\sqrt{n}} \right)$$

where n is a number of samples, $\bar{X}_n$ is the sample mean, s is the sample standard deviation, and finally t_{n-1} refers to the student's t-distribution This gives us the range in which the true mean is likely to lie with the probability of $(1 - \alpha)$.

2.5 Quantile regression

When training a neural network to make it learn from the training data, L1 or L2 loss functions are often used. These take the following form, respectively:

$$\mathcal{L}_1(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} |\mathbf{y}_i - \hat{\mathbf{y}}_i|$$

$$\mathcal{L}_2(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$$

It is known, that if we had a function $f(\mathbf{y} \mid \mathcal{D}, \mathbf{x})$ expressing the probability density of the output $\mathbf{y}$ given the training data $\mathcal{D}$ and input $\mathbf{x}$, we can show that by minimizing the L2 loss, the predictions made by this model would correspond to the conditional mean of this distribution while minimizing the L1 loss would make the model learn its conditional median. [32]

We define quantile as value q_a for which $P(X \leq q_a) = a$. Median, for example, is $q_{0.50}$, since:

$$P(X \leq q_{0.50}) = 0.5$$

The question arises: can we find a certain loss function with parameter a such that, by minimizing it, our model would predict the q_a quantile of the aforementioned probability density? The answer was proposed in [33], with the following loss function:

$$\mathcal{L}_Q(\mathbf{y}, \hat{\mathbf{y}}) = \left[\sum_{i \in \{i : \mathbf{y}_i \geq \hat{\mathbf{y}}_i\}} a|\mathbf{y}_i - \hat{\mathbf{y}}_i| + \sum_{i \in \{i : \mathbf{y}_i < \hat{\mathbf{y}}_i\}} (1-a)|\mathbf{y}_i - \hat{\mathbf{y}}_i| \right]$$

By minimizing this function, we are able to find the model that predicts the value of sample quantile q_a. We can see that minimizing this loss function is equivalent to minimizing the L1 loss if $a = 0.5$

The rest is straightforward: we select upper and lower bounds and train two models for them. Using the two models, we will find the prediction interval and its width should be our confidence measure.

It is important to note that while other methods, such as MC Dropout, are intended to approximate Bayesian inference, quantile regression employs the frequentist paradigm, as mentioned in Section 2.1. This provides a distinct perspective on quantifying and interpreting uncertainty in the model.

There is a big limitation of this method. Previous methods were usually able to approximate the distribution of the output and by changing parameters, we were able to tune the method to show wider or narrower intervals. By using the quantile regression, we must train two models, which might be time-consuming and once we have it, there is no natural way to change the predicted intervals other than training two new models.

Apart from the need to train two models, during inference, we only need to obtain two predictions. This is faster than the other methods that require many more passes to obtain useful statistics.

Chapter 3

Uncertainty quantification with PhyDNet

This chapter covers everything we need to know to start quantifying uncertainty with PhyDNet for precipitation nowcasting. We will describe the dataset we will be working with as well as everything that is used during the training process. The goal of this chapter is to provide all things necessary to start our implementation.

The previous chapters have covered the description of the PhyDNet model, as well as explanations of multiple methods for uncertainty quantification. This chapter connects these two topics by explaining how uncertainty quantification can be achieved with the PhyDNet model. Throughout the chapter, we will also discuss various topics related to the training and deployment of the model, with or without uncertainty quantification. After reading this chapter, the reader should understand our implementation and decisions we made.

3.1 Dataset

Since supervised machine learning models learn from data, dataset is a vital part of the training process. The dataset should be a good representation of the real-world phenomena we are trying to predict for the model to generalize well.

The dataset we use was provided by Meteopress. It consists of 24784 data samples collected using weather radar, each consisting of 12 single-channel precipitation rate images, with the data in the range 0–1. We will use our model to make 6 predictions and we will be using 6 images as the input sequence. The images are each 10 minutes apart. An example can be seen in Figure 3.1.

The problem with forecasting precipitation is the diversity in the data. Some samples may have very little or no precipitation in them. We can expect the model to predict with little to no error (and very little uncertainty) on these examples. On the other hand, we can expect the samples with lots of precipitation to be more difficult to predict for the model. This leads to two limitations in our implementation:

- We must always use the same subset of the dataset for any evaluation to be able to compare the models.
- When quantifying uncertainty, we might expect the mean of the interval widths to be very small. We will have to evaluate the variants of uncertainty quantification not only by using

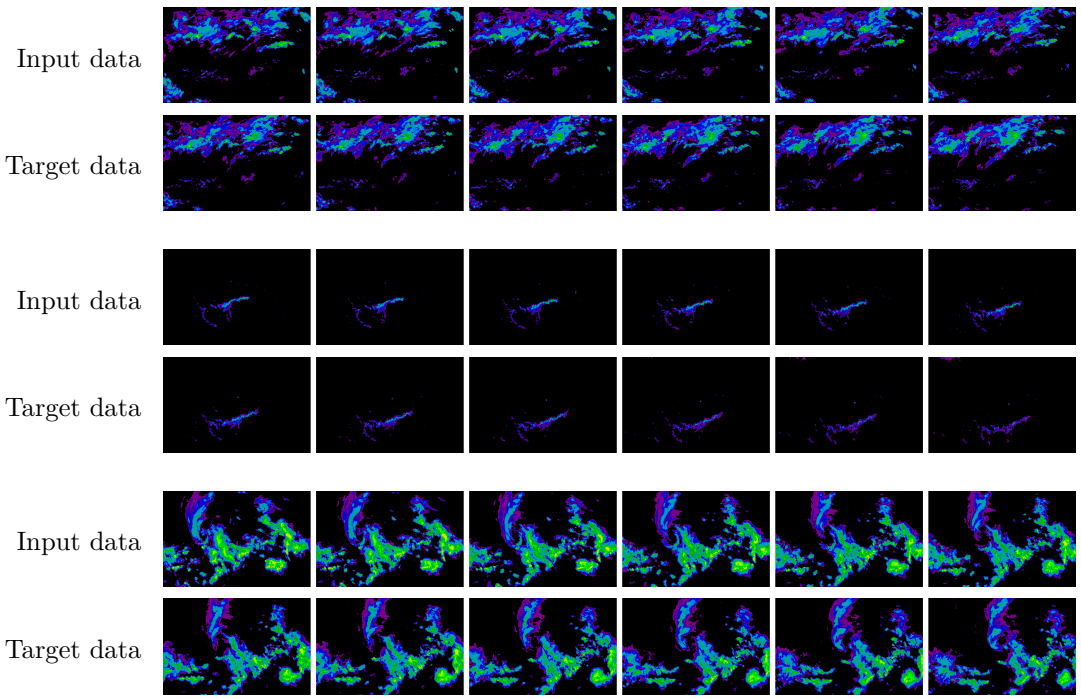

Figure 3.1 Examples from the dataset. Images are 10 minutes apart, the target data follow the input data.

the metrics but also by visualizing the results to see if, for some cases, the intervals aren't unnecessarily wide.

As mentioned earlier, inaccuracies in the data may cause uncertainty in the model. As it was said in the previous chapter, we don't have the resolving power necessary to get the exact data, which is inherently a problem when predicting a chaotic system the precipitation movement is [15]. Resolving power means the ability to measure the initial conditions of the system with infinite precision.

Since we have a good amount of data, the dataset will be divided into three parts. The training dataset will be used for the training of the model. This one will be the largest. Smaller chunks of data will be used as validation datasets. During training, we will be using it to see the ability of the model to generalize. We will also use this dataset to obtain the metrics of the methods that will be compared with each other. The last chunk of data, the testing dataset, will remain unused. This dataset is typically reserved for assessing the final model's performance on unseen data.

One may ask why we do not use the testing dataset for comparing different models. This approach is incorrect — selection of the model is part of the training process and by using the data, they are no longer an unseen representation of the real-world phenomena.

3.2 Possible issue with dropout implementation

In [28], dropout is introduced as a regularization technique for fully connected layers in deep learning networks. Since then, it has become one of the most popular techniques for regularizing deep learning networks. We've already explained why it isn't suitable for convolutional networks and found an alternative, proposed in [30], called spatial dropout.

We've also discussed the encoder-decoder architecture of PhyDNet, where the use of spatial dropout for regularization could be beneficial. A possible issue is that these blocks are already employing another technique, group normalization. While using normalization and regularization techniques together sounds reasonable (some normalization techniques even lead to its regularization [34]), authors in [35] show that using dropout and batch normalization together might lead to worse performance. To decide whether we should replace group normalization with spatial dropout or use these techniques together, we must understand how this combination works and see if it can cause issues. Therefore, we will make a small detour to take a look at the case of dropout and batch normalization.

3.2.1 Disharmony between Dropout and Batch Normalization

We will discuss the use of dropout and batch normalization together and see, what may be the problem when using normalization and regularization techniques together. Note that in this section, we refer to the regular dropout rather than its spatial variant. We will also talk about batch normalization, which is not used in the PhyDNet. Our goal is to see how can the use of regularization and normalization layers together lead to worse performance.

The dropout layer drops units in neural networks with probability p, which is a hyperparameter of the network. We can interpret the output of the dropout layer using the following notation [28]:

$$\tilde{x}_i = Z \cdot \frac{1}{1-p} \cdot x_i$$

where $Z \sim \text{Bernoulli}(1-p)$. The term $\frac{1}{1-p}$ is used for scaling so that the dropout doesn't change the magnitude of the output. The dropout is not used during inference, leaving the input untouched:

$$\tilde{x}_i = x_i$$

The expected value (mean) remains the same:

$$\mathbf{E}\left[\tilde{x}_i\right]_{\text{train}} = \mathbf{E}\left[Z\right] \cdot \frac{1}{1-p} \cdot x_i = (1-p) \cdot \frac{1}{1-p} \cdot x_i = x_i$$

$$\mathbf{E}\left[\tilde{x}_i\right]_{\text{test}} = x_i$$

This doesn't apply to the variance, as shown in [35]:

$$\text{Var}_{\text{train}}\left(x_i\right) = \frac{1}{1-p}\left(c^2 + v\right) - c^2$$

$$\text{Var}_{\text{test}}\left(x_i\right) = v$$

c and v are used to demonstrate the variance shift between training and testing that arises when dropout is used. The variance shift is not much of a problem for our model by itself.

When using batch normalization [34], training is done in mini-batches of data. The mean and variance of these mini-batches are calculated and used to normalize the inputs:

$$\tilde{x}_i = \frac{x_i - \overline{x}_{\mathcal{B}}}{\sqrt{s_{\mathcal{B}}^2 + \epsilon}}$$

$\overline{x}_{\mathcal{B}}$ and $s_{\mathcal{B}}^2$ are sample mean and sample variance of the mini-batch $\mathcal{B}$, respectively. ϵ is added to avoid division by zero. A running mean and running variance are calculated to be used during inference, so it takes the following form:

$$\tilde{x}_i = \frac{x_i - \overline{x}_R}{\sqrt{s_R^2 + \epsilon}}$$

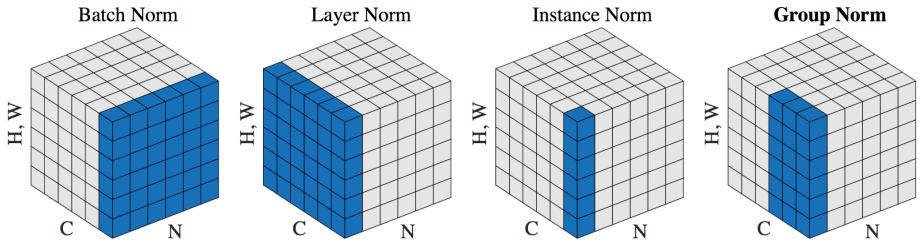

Figure 3.2 Visualization of different normalization techniques used in machine learning. [34]

where s_R and $\overline{x}_R$ are the running mean and running variance obtained during training. Thanks to the normalization of the data, batch normalization can accelerate the training process.

Finally, due to the variance shift between the training and testing phases caused by dropout, the statistics used in batch normalization deviate from the real-world statistics, which leads to incorrect normalization and thus the decrease in the model's performance. [35]

To sum it up, dropout changes the distribution of the output of the layer during training, affecting its variance. The batch normalization method uses the approximate variance obtained from the training data (using running variance), which is influenced by the dropout. Since dropout is not used during inference, the shifted approximate variance causes the problem.

3.2.2 Group Normalization

Group normalization was proposed in [36], introducing a simple alternative to batch normalization. One of the main disadvantages of batch normalization is its dependence on batch size, leading to a worse performance with small batches of data. This problem is solved in group normalization as it is independent of the batch size.

Group normalization normalizes the outputs within a group of G feature maps in the input tensor. Each of these groups contains C/G channels, where C is the total number of channels. Next, we obtain the mean and variance of these groups and we normalize the values over groups in the same manner as in batch normalization. We can see the difference between batch normalization and group normalization visualized in Figure 3.2. The other methods in the figure are in fact just the edge cases of group normalization, setting $G = 1$ or $G = C$ (they actually existed before the group normalization). [36]

We can also see that the normalization procedure is consistent during training and inference, so the concerns of incorrect statistics we have with batch normalization do not apply, which suggests that using spatial dropout and group normalization together could be safe. Based on this reasoning, we will use both techniques together in our experiments. It is worth noting, however, that further empirical evaluation may be necessary to fully validate the effectiveness of using spatial dropout and group normalization together. This is beyond the scope of our work.

Additionally, in the original paper, the authors find that group normalization has lower regularization power than batch normalization. [34]

3.3 Uncertainty quantification

Measuring uncertainty in the predicted images is different than in the single-point scenario. We are actually dealing with many points that are not independent of each other — if we have high uncertainty in one pixel, we can also expect high uncertainty in the pixels around it. We will be using the Mean Prediction Interval Width (MPIW) and Prediction Interval Coverage Probability

(PICP), mathematically expressed as [21]:

$$\text{MPIW}(\hat{\mathbf{L}}, \hat{\mathbf{U}}) = \frac{1}{T}\sum_{t=1}^{T}(\hat{\mathbf{U}}_t - \hat{\mathbf{L}}_t)$$

$$\text{PICP}(\hat{\mathbf{L}}, \hat{\mathbf{U}}, \mathbf{y}) = \frac{1}{T}\sum_{t=1}^{T}\mathbb{1}\left(\mathbf{y}_t - \hat{\mathbf{L}}_t\right)\mathbb{1}\left(\hat{\mathbf{U}}_t - \mathbf{y}_t\right)$$

Where $\mathbf{L}$ and $\mathbf{U}$ are the lower and upper bounds of the predicted interval, respectively. $\mathbb{1}$ is a function, returning 1 for positives and 0 for other values. The best uncertainty quantification would be maximizing the coverage while keeping the width as low as possible.

The MPIW and PICP metrics were not designed for spatio-temporal data; however, in many use case scenarios, they effectively capture the desired information. A possible use case involves an individual selecting a single location and the model would warn them in case of a certain forecast of extreme weather conditions. We can see that in this case, we do not directly use the spatio-temporal nature of the data. Instead, we measure an uncertainty of a single-value prediction, for which MPIW and PICP were designed. However, we must keep in mind that in this particular use case, we aim to find a way to quantify uncertainty so that it minimizes the prediction width in points where there is high precipitation. This may be difficult task because of the skewness of the data and visualization can help.

We have already discussed various methods of implementing uncertainty quantification in Chapter 2. We will now discuss the use of two of these methods, Monte Carlo dropout and quantile regression, in the PhyDNet model.

3.3.1 MC dropout

We must discuss how dropout will be implemented in the PhyDNet architecture. We will dissect its parts individually and tackle this question by parts.

We can think of each part of PhyDNet as a module — by implementing the dropout, we express the uncertainty contained in each of these. For example, when we talk about uncertainty in the encoder, we can think of a possibility of error between the conversion of the input into latent space — we address the expected imperfection of encoding the information, which may be caused by inaccuracies in the input, like time delays or inaccurate measurements, or by limitations of the encoder architecture. This example can be extrapolated into other parts of the PhyDNet model.

Spatial dropout (like regular dropout) has one hyperparameter — the dropout rate which determines the probability of an element being dropped. We will try multiple values and determine their impact on the intervals using the results.

3.3.1.1 Dropout in the encoder/decoder

Implementing the dropout in the encoder and decoder is relatively straightforward since they consist of simple convolutional and deconvolutional neural networks, respectively. During training, these parts of PhyDNet become physics-informed and they tend to extract features that will be disentangled into physical expression of the latent dynamics of the system.

We will use the spatial dropout previously discussed to drop the whole feature maps. This should lead to making various combinations of them, slightly changing the model in each pass during inference.

We are using the spatial dropout implemented in PyTorch [37] (*torch.nn.Dropout2d*). The dropout is located after the convolution (deconvolution) operation, as the documentation suggests. In fact, we use it even after the group normalization and activation layer. Firstly, it doesn't matter whether we use the dropout after or before the activation function, because we

use leaky ReLU which maps zero values to zeros. The spatial dropout is used after the group normalization, since it ensures preservation of the statistical properties for group normalization.

3.3.1.2 Dropout in PhyCell

We've already discussed PhyCell's architecture in Section 1.1.3. PhyCell is a physically constrained cell designed to model the partial differential equations (PDEs) representing the physical dynamics of a system. Based on the architecture of the PhyCell, implementing random drops of feature maps may not be appropriate for several reasons:

- Dropout might lead to loss of spatio-temporal information that is vital for the prediction.
- Use of dropout might disrupt the constraints given by the PDEs.

The drops would affect the members of the PDEs which would lead to incorrect representation of the physical dynamics. For the aforementioned reasons, we will not implement dropout in PhyCell at all. There might be some careful, controlled ways that don't directly involve the PDEs, like implementing dropout on the input of PhyCell, but it is likely that this wouldn't lead to much variation on the output and the uncertainty in this can be expressed by using dropout in the encoder.

We must note that the structure of PhyCell is different from the structure of other networks. The physical constraints allow the use of fewer parameters than what must be used in other methods, like in the ConvLSTM. This makes it a powerful tool for predicting physical-based phenomena, but it might also perform worse in cases where the physical bounds are redundant.

3.3.1.3 Dropout in ConvLSTM

We expect the recurrent cell to have a high impact on the uncertainty of the model since it is used for residual dynamics that cannot be expressed using the physical prediction layer. We want to find a way that would be the best for our purposes — minimizing the MPIW and maximizing the PICP. We will do this by using multiple variants and comparing them.

The use of dropout in recurrent neural networks, especially in the LSTM, has been closely discussed in various works. We will try to use what has worked for the LSTM and try to convert it to its convolutional variant.

We will take inspiration from the implementation in [38]. This work collects the previous attempts and also proposes its own method of implementing the dropout into the LSTM cell, which exceeds the previous methods. We will be using the following ConvLSTM variant:

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf} * (\mathbf{x}_t \odot \mathbf{z}_{xf}) + \mathbf{W}_{hf} * \left(\mathbf{h}^{\mathbf{r}}_{t-1} \odot \mathbf{z}_{hf}\right) + \mathbf{b}_f)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi} * (\mathbf{x}_t \odot \mathbf{z}_{xi}) + \mathbf{W}_{hi} * \left(\mathbf{h}^{\mathbf{r}}_{t-1} \odot \mathbf{z}_{hi}\right) + \mathbf{b}_i)$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_{xC} * (\mathbf{x}_t \odot \mathbf{z}_{xC}) + \mathbf{W}_{hC} * \left(\mathbf{h}^{\mathbf{r}}_{t-1} \odot \mathbf{z}_{hC}\right) + \mathbf{b}_C)$$

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo} * (\mathbf{x}_t \odot \mathbf{z}_{xo}) + \mathbf{W}_{ho} * \left(\mathbf{h}^{\mathbf{r}}_{t-1} \odot \mathbf{z}_{ho}\right) + \mathbf{b}_o)$$

$$\mathbf{h}^{\mathbf{r}}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$

The $\mathbf{z}_{x:}$ and $\mathbf{z}_{h:}$ are masks for the dropout operations. Note that these masks are the same in every time step of the recurrent cell, which is one of the main differences between this and the other methods.

Note that the original work [38], which we use as an inspiration, uses vanilla LSTM and regular dropout, while we are trying to implement the spatial dropout into the convolutional variant of the LSTM. This makes our method quite different from the method in the paper.

In our base implementation, we utilize what in [38] is referred to as the tied-weights LSTM (ConvLSTM in our case). The aforementioned untied-weights variant of the model has been shown to yield slightly better results, so we will adopt this approach when implementing dropout in the ConvLSTM with locked dropout masks. We must, however, keep this change in the model's architecture in mind. The other approaches will stick to the PhyDNet's original implementation, as we want to find an easy way of implementing uncertainty quantification into our model.

Resources on adding the spatial dropout into ConvLSTM for uncertainty quantification are limited. We will also test two other approaches — adding the spatial dropout to the input of the recurrent network layer and applying the dropout on the hidden state and the cell state of each ConvLSTM's cell.

3.4 Training the model

Training of a deep learning model is a process of adjusting its parameters and fine-tuning the internal structure of the model so that it maps the input into the desired output in the most accurate way, based on some criteria. This is usually done by minimizing the error and maximizing its ability to generalize to make the model perform well on unseen data.

3.4.1 PhyDNet hyperparameters

Hyperparameters of the model are configurable parameters defining its behavior. Unlike the standard parameters, values of hyperparameters are set by a person training the model. This allows us to fine-tune the model and find the ideal configuration of the model.

We can divide PhyDNet's hyperparameters by its subcomponents. For example, we can set the number of convolutional neural networks in the physical predictor in PhyCell, or the number of hidden feature maps in the ConvLSTM layer. There are also general hyperparameters, like used loss function or optimizer. It is not necessary to cover the hyperparameters used in individual layers, as they are inherently determined by the properties of the layer itself. We will have a closer look at the general hyperparameters — loss function and optimizer.

3.4.2 Loss function

The loss function determines if the model is good or bad and allows us to compare different setups. The usable loss function is defined so that it is minimal if the model predicts in an optimal way. This means that our goal when training the model is to minimize the loss function.

There are many loss functions that are being used, depending on the task. We are dealing with the regression problem, and the most often used loss functions are L1 and L2 loss, discussed in Section 2.5. Our model uses the sum of these:

$$\mathcal{L}\left(\mathbf{y}, \hat{\mathbf{y}}\right) = \mathcal{L}_1\left(\mathbf{y}, \hat{\mathbf{y}}\right) + \mathcal{L}_2\left(\mathbf{y}, \hat{\mathbf{y}}\right)$$

L1 loss penalizes all errors equally, which makes a good loss function when dealing with outliers, L2 loss on the other hand penalizes larger errors more.

3.4.3 Optimizer

Gradient descent is one of the most widely used optimization algorithms, and we previously discussed a specific variant of it, the Stochastic Gradient Descent (SGD) optimizer, in Section 2.3.1. This method is used very often, but we might need a different approach for large and complex networks since SGD has its shortcomings. For example, it might be difficult to select the right learning rate. [25]

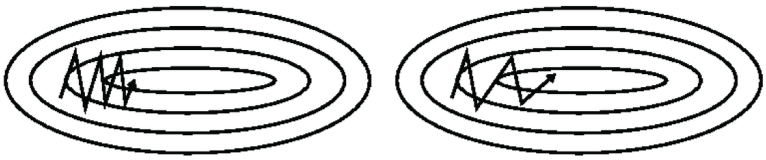

Figure 3.3 Comparison between regular SGD (left) and SGD with momentum (right). [39]

The mathematical expression of the SGD method takes the following form:

$$\theta = \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta; x_i; y_i)$$

There are two problems that may occur. The first one is possible convergence into local minima. With lower values of the learning rate, the probability of this increases. A high learning rate is also not optimal, as it may cause the model not to converge at all (in fact, even the fluctuations caused by a high learning rate can cause problems). The other problem we have happens when gradients are very steep in one dimension, yet very flat in the other one. This causes fluctuations in the steep dimension and slow descent in the flat dimension [25]. This behavior is depicted on the left side in Figure 3.3. The time of training also depends on the learning rate — lower learning rate leads to higher training time. [39]

A possible solution to both of the aforementioned problems is introducing momentum into the equation:

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla_\theta \mathcal{L}(\theta; x_i; y_i)$$

$$\theta = \theta - v_t$$

Momentum γ accelerates the learning process in the direction of the gradients [25]. This is depicted on the right side in Figure 3.3. This is similar to the Newtonian equations, so we can liken it to a ball in a hilly landscape [40]. The momentum method leads to faster convergence and reduction in oscillation. We can enhance this by using Nesterov accelerated gradient:

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla_\theta \mathcal{L}(\theta - \gamma \cdot v_{t-1}; x_i; y_i)$$

$$\theta = \theta - v_t$$

This method allows for a more accurate estimate of the gradient and faster convergence. [25]

The next step forward is adaptive gradients, used by methods like Adagrad, Adadelta, RMSprop, and finally, a method we will be using, Adam. Its main idea is adjusting the learning rate of each parameter during training based on the gradients obtained in previous time steps of the training. The Adagrad method is the original adaptive gradients implementation and its goal is to give a smaller learning rate for frequently occurring parameters and a higher learning rate for infrequent parameters. We will see that the same idea is used in Adam. [25]

We set the $g_{t,i}$ to be the gradient of the loss function with respect to the parameter θ_i at time step t:

$$g_{t,i} = \nabla_{\theta_i} \mathcal{L}(\theta_{t,i})$$

For example, the SGD equation can be expressed as follows:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

The g_t would then be a vector of these gradients computed for all parameters θ_i at time step t. We adopt this notation from [25], as it is more expressive.

Adaptive Moment Estimation (Adam) combines the previously mentioned ideas, momentum, and adaptive gradients. It does so by estimating the first and second moments. The first-moment estimation takes the following form:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

The first moment represents the mean, which in our case acts similarly to momentum. The expression of the second moment estimation is the following:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

where β_1 and β_2 parameters are decay rates and they are specified manually (authors propose the default values $\beta_1 = 0.9$ and $\beta_2 = 0.999$). This, on the other hand, represents the uncentered variance. Variance is higher with higher changes in the parameters, so we can use it for the adaptive gradients [25].

The authors of Adam in [41] found, that initializing the estimations to zero leads to them being biased towards zero. This happens especially during the initial time steps and with the decay rates close to 1. The solution is to use the bias-corrected first and second-moment estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Finally, the update rule for the parameters is as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

The ϵ value helps avoid the division by zero and the authors propose a value of 10^{-8} [25]. Adam has been shown to outperform the aforementioned methods in [41].

3.5 Evaluating the model

The evaluation of uncertainty quantification has been discussed in Section 3.3. Now we will cover the evaluation of the point predictions obtained directly from the model. We will have to evaluate the models to see whether or not the use of dropout during training improves the performance of the model.

3.5.1 Metrics

We have already discussed L1 and L2 loss functions in Section 2.5. These minimize widely used metrics, mean absolute error and mean square error, respectively:

$$\text{MAE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

$$\text{MSE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

where N is the number of samples, $\hat{y}_i$ is the prediction and y_i is the target value. Similarly to the respective loss functions, MSE penalizes higher errors more, while MAE penalizes all errors equally.

In the case of precipitation nowcasting, the aforementioned metrics can be useful for evaluating the performance of a precipitation rate prediction model if we treat each pixel as a point with a specific value representing the precipitation rate. The metrics would provide an overall measure of the accuracy of the prediction. On the other hand, we want to visualize these values as images. The human eye is highly adapted for extracting structural information from a scene [42] and the standard metrics aren't able to capture this very well.

Structural Similarity Index Measure (SSIM) was proposed in [42] and it is designed to take into account the structural information of the image. This is done by comparing 3 important features of the image, luminance, contrast, and structure. We first have to obtain the mean intensity and the standard deviation of the values of a pixel. These take the following form, respectively:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(x_i - \mu_x\right)^2}$$

The mean value serves for luminance comparison, so the function $l(x, y)$ will be comparing μ_x and μ_y. Standard deviation is used as an approximation of the contrast, so the function $c(x, y)$ will be comparing σ_x and σ_y. Finally, the structure comparison function uses the covariance between the luminance values of the two images we are comparing, σ_{xy}. This value can be estimated as follows:

$$\sigma_{xy} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(x_i - \mu_x\right)\left(y_i - \mu_y\right)}$$

With this equation, we have all we need to put the SSIM together. [42]

The comparison of these values is done as follows:

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$$

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

where C_i values are used to avoid instability when the values in the denominator are very close to zero. Finally, the SSIM metrics take the following form:

$$\mathrm{SSIM}(x, y) = [l(x, y)]^\alpha + [c(x, y)]^\beta + [s(x, y)]^\gamma$$

where α, β, and γ are used to define the importance of each comparison function. [42]

3.5.2 Visualization

The input and the output of our model are weather radar images and their prediction. These take the form of a matrix containing values 0–1 determining the precipitation rate. A better approach to visualize the data is by using images. We have already done this in Figure 3.1 and we can see that this representation seems intuitive and informative.

The natural way would be to convert the 0–1 value to the intensity of the pixel, which would lead to single-channel (grayscale) images. This approach is usable, however, it could prove difficult for the human eye to distinguish between the levels in the image. A solution is to convert

Figure 3.4 Color map used for the visualization.

the grayscale image data to a color map. The color map we use is *nipy_spectral* from Matplotlib [43], depicted in Figure 3.4. The comparison between grayscale and color map visualization approaches can be seen in Figure 3.5. This color map retains 0 as black and 1 as white, utilizing levels in different colors to effectively distinguish the data.

There are some limitations to the color map approach as well. For example, by using different color map, we might get a different interpretation of the data. This may cause two different weather forecasts to look more different than they actually are.

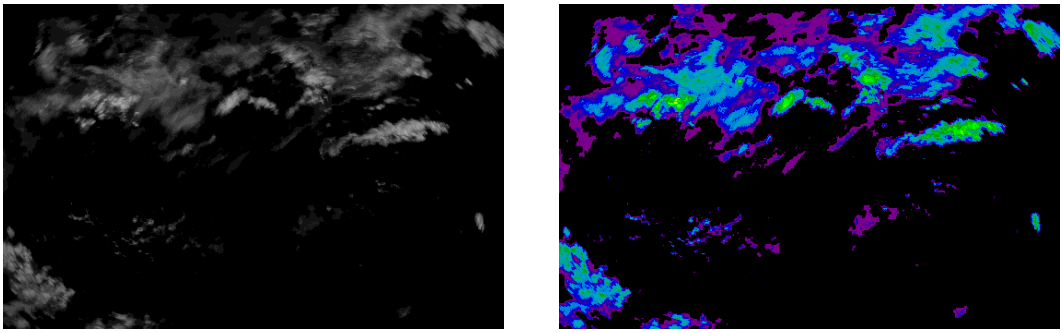

Figure 3.5 Comparison between grayscale and colormap image visualization.

Chapter 4
Experiments

This chapter summarizes the practical results of our efforts. We compare various methods of utilizing MC dropout for quantifying uncertainty, and we also compare MC dropout with quantile regression. Additionally, we provide visualizations of the network outputs using these techniques.

In the first section of this chapter, we will discuss dropout as a regularization technique and examine how incorporating dropout affects the accuracy of our model in point predictions. Later on, we will delve into the topic of uncertainty quantification, where we will review the outcomes of our implementation of MC dropout and quantile regression.

Finally, we will compare the two methods using metrics and visualizations described in the previous chapter. We will use the colormaps to visualize the widths of the prediction intervals obtained by these methods. This should allow us to see the strengths and weaknesses of the methods.

4.1 Dropout as a method of regularization

We trained models with spatial dropout applied at various stages of PhyDNet and compared their performance using evaluation metrics described in Section 3.5.1. Additionally, we included a comparison with the regular model, without dropout, to assess its impact.

Based on the results presented in Table 4.1, it seems that the addition of dropout results in little to no significant improvement or degradation in the model's performance. This may be caused by the possible redundancy of additional regularization, as discussed in 2.4.1. Not only we can expect the model to have a solid implicit regularization, but it also uses an explicit regularization technique, group normalization, discussed in 3.2.2.

Note, that from these results we cannot definitely state that the use of dropout as a regularization technique in PhyDNet is redundant, even though our experiments did not show a significant difference in performance with or without it. Further investigation of the theoretical grounding, as well as providing test results on multiple datasets may be necessary to determine whether dropout may be beneficial or not.

We have already mentioned the slightly changed architecture of the models using dropout in locked locations in Section 3.3.1.3. As we can see, this change has a minimal effect on the performance. We should, however, keep it in mind. It is possible that the model works differently than the other one, so we have to make a clear distinction between them.

Dropout	Drop rate	MSE ↓	MAE ↓	SSIM ↑
— (v1)	—	0.00153	0.01057	0.91661
— (v2)	—	0.00150	0.01052	0.91763
Encoder	0.2	0.00157	0.01078	0.91628
Encoder	0.4	0.00158	0.01071	0.91567
Decoder	0.2	0.00158	0.01074	0.91649
Decoder	0.4	0.00160	0.01091	0.91492
Encoder + Decoder	0.2	0.00158	0.01076	0.91616
Encoder + Decoder	0.4	0.00154	0.01056	0.91716
ConvLSTM (cell)	0.2	0.00152	0.01054	0.91738
ConvLSTM (cell)	0.4	0.00154	0.01069	0.91577
ConvLSTM (input)	0.2	0.00155	0.01062	0.91682
ConvLSTM (input)	0.4	0.00152	0.01048	0.91720
ConvLSTM (cell, locked) *	0.2	0.00149	0.01040	0.91761
ConvLSTM (cell, locked) *	0.4	0.00151	0.01054	0.91722
Encoder + ConvLSTM (input) + Decoder	0.2	0.00154	0.01069	0.91684
Encoder + ConvLSTM (input) + Decoder	0.4	0.00153	0.01060	0.91691
Encoder + ConvLSTM (cell) + Decoder	0.2	0.00155	0.01066	0.91678
Encoder + ConvLSTM (cell) + Decoder	0.4	0.00164	0.01104	0.91398
Encoder + ConvLSTM (cell, locked) + Decoder *	0.2	0.00151	0.01064	0.91684
Encoder + ConvLSTM (cell, locked) + Decoder *	0.4	0.00160	0.01087	0.91459

Table 4.1 Evaluation of models trained with dropout. The v2 model of PhyDNet without dropout refers to the variant used for dropout with fixed locations, models using this architecture are marked with *.

4.2 MC dropout for uncertainty quantification

We have used the MC dropout method on models from previous sections. Their ability to quantify uncertainty was tested using the metrics mentioned in Section 3.3.

From the results listed in Table 4.2, we see that the selection of where dropout is used has a significant impact on the resulting intervals. It seems that using dropout in the encoder does not have as much impact as, for example, dropout in the decoder layer. Using dropout in the decoder layer also leads to a higher PICP, which suggests that introducing uncertainty in it can better capture the uncertainty in the model, with the cost of wider intervals (MPIW).

We also see that MPIW does not directly correlate to the prediction interval coverage probability. In some cases, even the same model with different values of dropout probability lead to wider intervals, yet less coverage. It is necessary to test multiple dropout probability values before making the decision of which model to employ.

Very interesting fact lies in the very narrow prediction intervals in some cases when using dropout inside the recurrent cell, which is used to predict the residual dynamics of the system. We see that in these models, changes in the ConvLSTM doesn't affect the output very much. Now let's find out why.

We used dropout for training, which regularizes the ConvLSTM. The goal of regularization is to make the model generalize better, which means to learn the consistent phenomena that occurs in the real world. The residual dynamics layer tries to learn the constant patterns that appear in the residual dynamics. We hypothesize that applying excessive dropout, in this case, may lead to underfitting since finding constant patterns in residual dynamics is a challenging task. The underfit ConvLSTM would then cause an error on the output, so the model learns to ignore it.

We can verify this by dropping the contribution of each layer separately. In the model trained without any additional regularization, we see that the final prediction depends a lot on the output

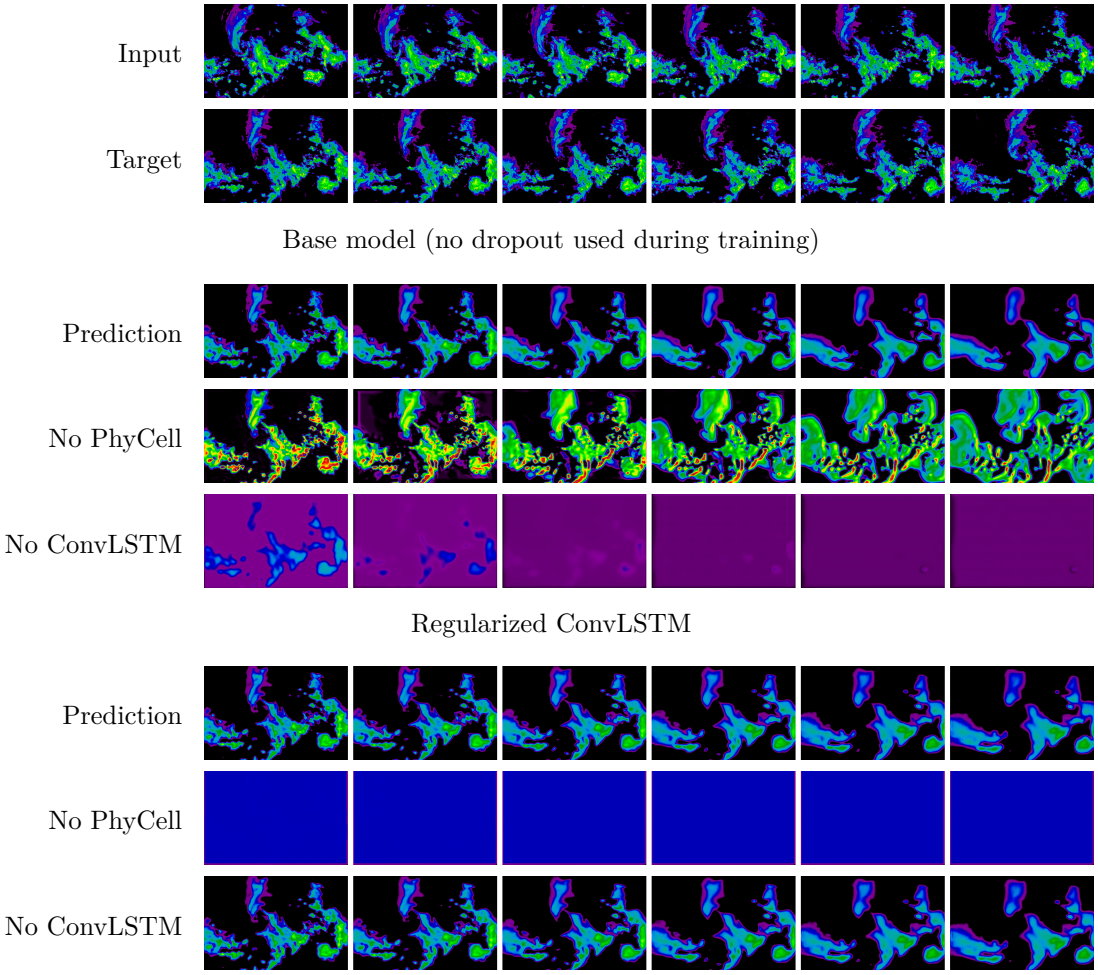

Figure 4.1 Some ways of regularizing the ConvLSTM may lead to its demise in the prediction.

of the ConvLSTM layer. However, after the regularization of this layer, the prediction is based solely on the physical prediction, the PhyCell layer. This is visualized in Figure 4.1 and it shows a possible redundancy of the ConvLSTM layer in the PhyDNet model when used on our data since the regularized model does not perform worse, as seen in Table 4.1. Since we work with precipitation data, we may believe that it can be modeled well only by using the PhyCell in PhyDNet. The figure support this claim.

We will now examine the intervals produced by these variants more closely. We selected predictions with the highest mean squared error in the validation dataset, as this is where we might expect the highest uncertainty of the model. The results can be seen in Figure 4.2 and seem consistent with what we would expect from Table 4.2. When using dropout in the encoder or decoder (or both), the width of the interval appears to remain very similar across every time step, indicating that the uncertainty in these modules does not depend on time. We can say that the interpretation of the uncertainty in these modules is up for debate. We might prefer it to be time-dependent because their input is the previous prediction, which inherently contains uncertainty. However, the modules themselves serve to convert the input into a latent space in the case of the encoder, regardless of the data, providing a reason why we might want the uncertainty in these modules to be time-independent. We also observe that the use of MC Dropout in ConvLSTM leads to either very narrow intervals or intervals that widen over time.

Dropout	Drop rate	MPIW ↓	PICP ↑
Encoder	0.2	0.0007	0.0495
Encoder	0.4	0.0006	0.0526
Decoder	0.2	0.0030	0.6527
Decoder	0.4	0.0048	0.4819
Encoder + Decoder	0.2	0.0032	0.7292
Encoder + Decoder	0.4	0.0046	0.4272
ConvLSTM (input)	0.2	0.0020	0.0914
ConvLSTM (input)	0.4	0.0018	0.0527
ConvLSTM (cell)	0.2	0.0030	0.1377
ConvLSTM (cell)	0.4	0.0000	0.0018
ConvLSTM (cell, locked) *	0.2	0.0031	0.1018
ConvLSTM (cell, locked) *	0.4	0.0032	0.0784
Encoder + ConvLSTM (input) + Decoder	0.2	0.0040	0.5976
Encoder + ConvLSTM (input) + Decoder	0.4	0.0132	0.8533
Encoder + ConvLSTM (cell) + Decoder	0.2	0.0078	0.8371
Encoder + ConvLSTM (cell) + Decoder	0.4	0.0046	0.6743
Encoder + ConvLSTM (cell, locked) + Decoder *	0.2	0.0038	0.2407
Encoder + ConvLSTM (cell, locked) + Decoder *	0.4	0.0027	0.7895

Table 4.2 MC dropout - 15 samples, alpha=0.02, models are regularized using dropout. Models using the alternative architecture are marked with *.

Intuitively, the residual dynamics modeled by ConvLSTM should become more uncertain with time, making these results seem reasonable.

The narrow intervals caused by the regularization of the ConvLSTM are a potential problem. The dropout used in training makes the contribution of the residual dynamics predictor to the final prediction negligible. In search of a solution for this, we will implement dropout into the residual predictor in a model that was trained without dropout and see whether this will lead to more useful results.

The data in Table 4.3 shows that MC dropout method on model trained without using dropout leads to wider intervals and higher coverage probabilities. It might seem to be the better option, as we are able to get a better coverage (up to around 93%), but the interval widths are also a lot wider in most cases. To see whether the prediction interval width is a problem, we will have a look at the visualization of the widths.

We have to keep in mind that we use different model for the locked locations dropout. The results suggest that this model has learned to rely more on the PhyCell layer, so that the contribution of the residual predictor to the output is less important, yet it is still present, unlike in some models with regularized ConvLSTM.

By examining the widths in Figure 4.3, we can see that in most cases, the prediction intervals are wider. The contribution of the non-regularized ConvLSTM is higher, so using MC dropout results in a more diverse range of outcomes. We also notice that the widths of the prediction intervals using dropout in the encoder/decoder are time-dependent, as they widen over time. This may not be desirable; therefore, a combination of regularized and non-regularized models could potentially be employed. Utilizing the non-regularized model with the MC dropout method could be beneficial in situations where there is little room for error. Although the wider prediction intervals suggest increased uncertainty, the high coverage ensures that the actual predictions are likely to fall within these intervals.

We can also see that in Table 4.3, the interval width dependence on the value of the dropout rate is far more obvious than in the regularized model. This may help in adjusting and fine-tuning the parameters of the model when employing this method.

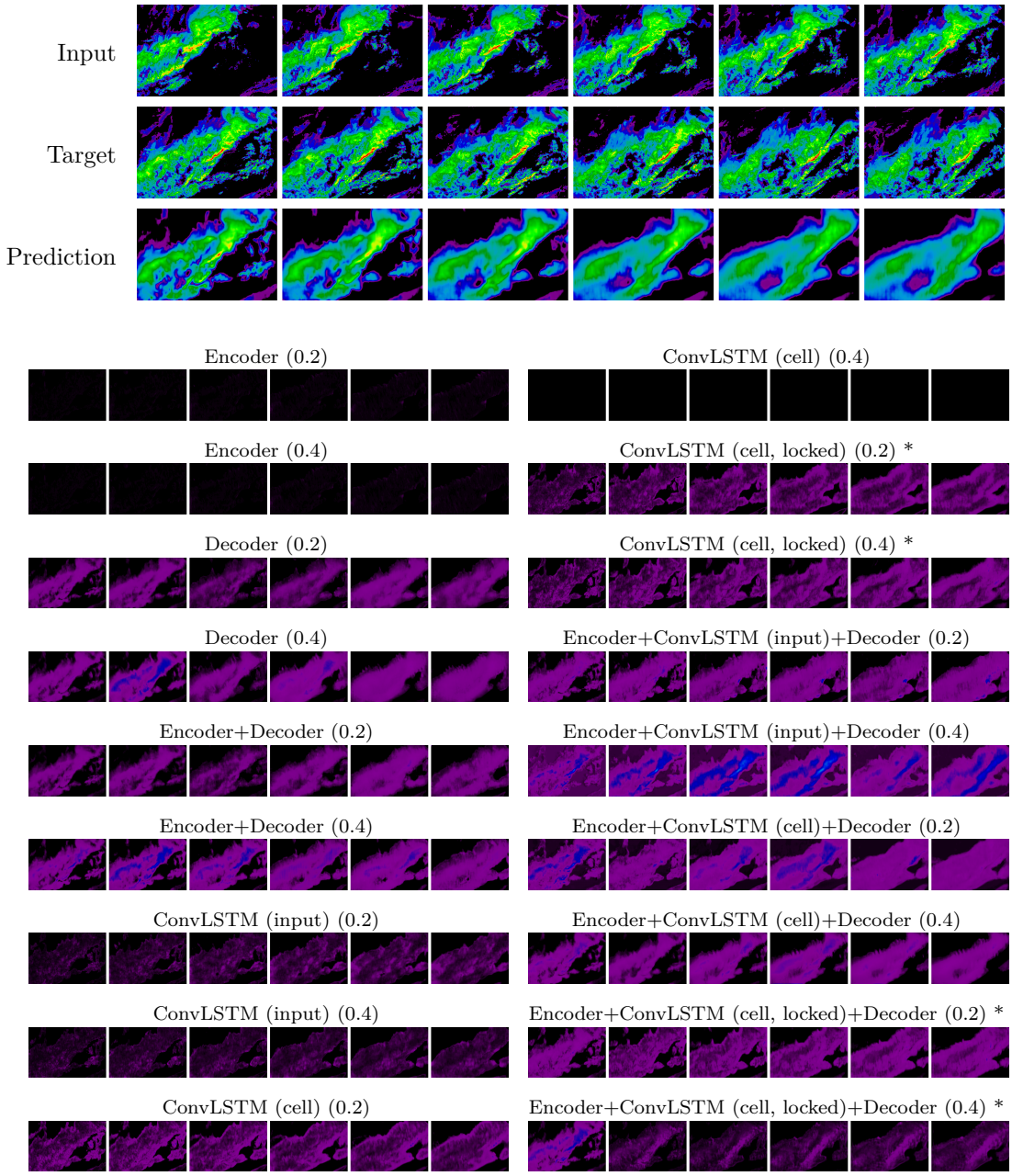

Figure 4.2 Widths of prediction intervals obtained from MC dropout using model regularized with dropout. Models using the alternative architecture are marked with *.

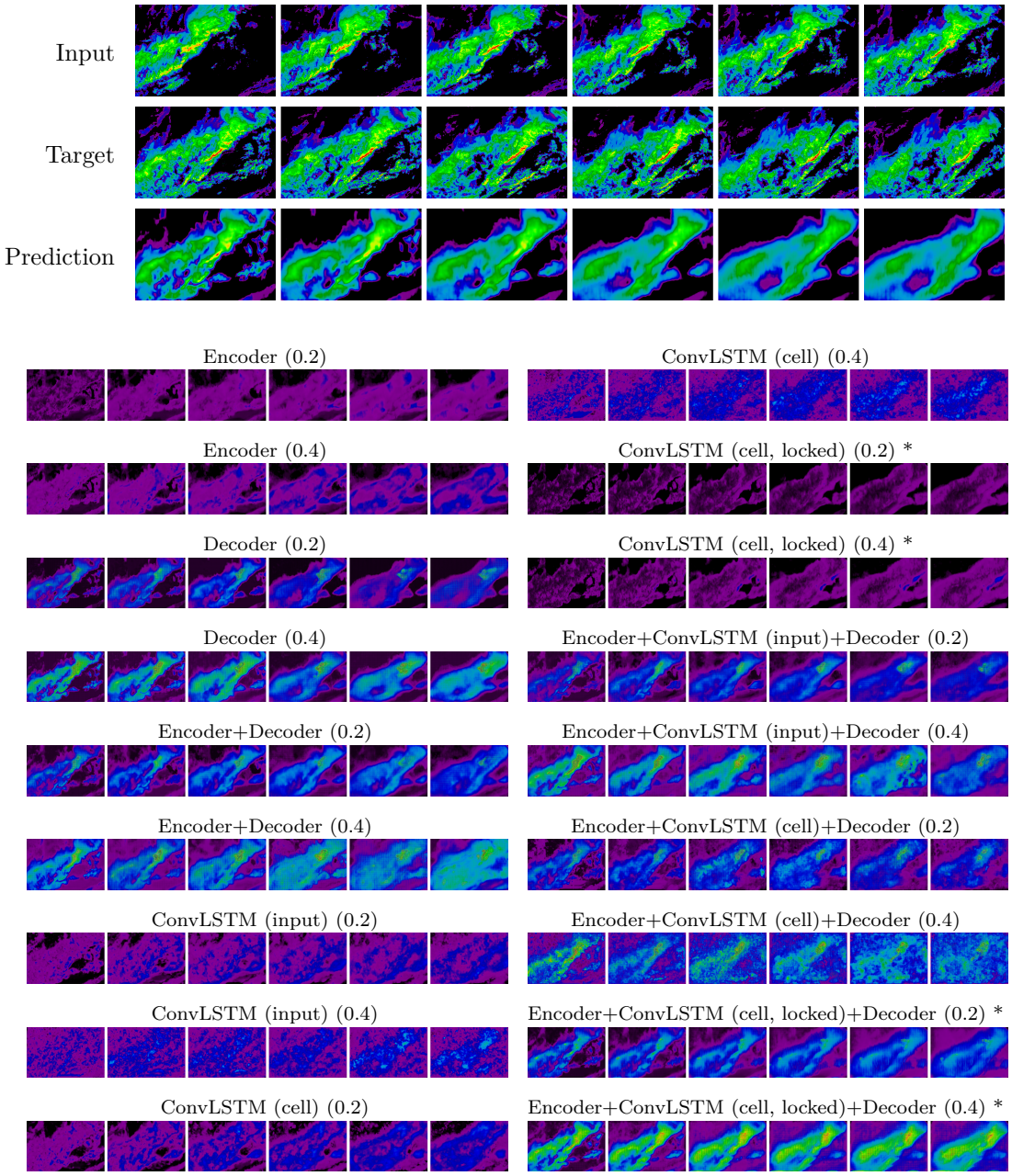

Figure 4.3 Widths of prediction intervals obtained from MC dropout using a model trained without dropout. Models using the alternative architecture are marked with *.

Dropout	Drop rate	MPIW ↓	PICP ↑
Encoder	0.2	0.0100	0.2332
Encoder	0.4	0.0172	0.4057
Decoder	0.2	0.0273	0.8906
Decoder	0.4	0.0473	0.9231
Encoder + Decoder	0.2	0.0309	0.8999
Encoder + Decoder	0.4	0.0530	0.9306
ConvLSTM (input)	0.2	0.0159	0.4405
ConvLSTM (input)	0.4	0.0336	0.7698
ConvLSTM (cell)	0.2	0.0473	0.8506
ConvLSTM (cell)	0.4	0.0819	0.2467
ConvLSTM (cell, locked) *	0.2	0.0025	0.0811
ConvLSTM (cell, locked) *	0.4	0.0037	0.1506
Encoder + ConvLSTM (input) + Decoder	0.2	0.0352	0.9072
Encoder + ConvLSTM (input) + Decoder	0.4	0.0642	0.9323
Encoder + ConvLSTM (cell) + Decoder	0.2	0.0654	0.9077
Encoder + ConvLSTM (cell) + Decoder	0.4	0.1371	0.5744
Encoder + ConvLSTM (cell, locked) + Decoder *	0.2	0.0475	0.9220
Encoder + ConvLSTM (cell, locked) + Decoder *	0.4	0.1216	0.9465

Table 4.3 MC Dropout - 15 samples, alpha=0.02, model trained without dropout. Models using the alternative architecture are marked with *.

MC dropout proves to be a useful method. Through fine-tuning, we may be able to identify the optimal configuration for our specific use case. We believe that the versatility and value of the method lie in its configurability in various ways.

4.3 Quantile regression for uncertainty quantification

Implementing the quantile regression method into the training process is relatively straightforward, as discussed in 2.5. By selecting the desired bounds and training the models using the specialized quantile loss function, we can achieve our goals.

The results in Table 4.4 suggest that quantile regression is a superior method, with extremely high coverage probability and, in some cases, lower mean prediction interval widths compared to MC dropout. Given its fast inference time, which requires only two predictions, it's reasonable to question the necessity of other methods. We will address this in the following parts of this section.

Initially, one might assume that the width of the bounds set during the training of quantile regression models would directly represent the proportion of data contained within them. For instance, if a quantile regression model predicts the 0.15 to 0.85 interval, we would expect it to encompass 70% of the data, which corresponds to the width. As we can see, this is not the case, as the coverage probability is far higher.

Quantile values (predicted bounds)	MPIW ↓	PICP ↑
0.15 – 0.85	0.0263	0.9522
0.05 – 0.95	0.0397	0.9771
0.01 – 0.99	0.0686	0.9916

Table 4.4 Results of using quantile regression.

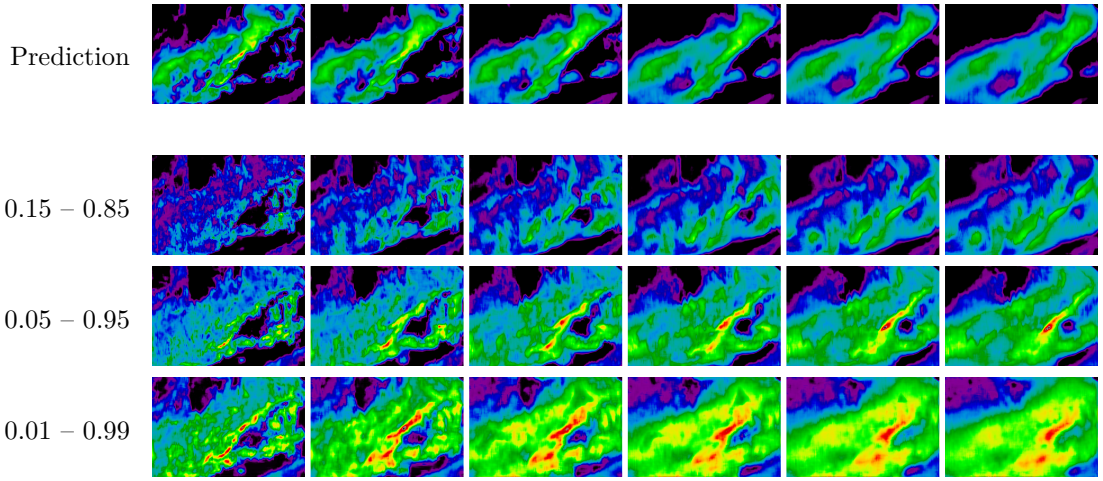

Figure 4.4 Widths of prediction intervals with the quantile regression.

In Figure 4.4, the interval widths appear to be significantly larger than those shown in Figures 4.2 and 4.3, which represent the MC dropout results. However, this observation doesn't align with the data in Table 4.4, as some of the MC dropout results have higher MPIWs than the 0.01 to 0.99 quantile regression model. We must explore a different example to see why is it so. We will compare it to the MC dropout method.

4.3.1 Comparing quantile regression to MC dropout

Considering the MPIW and PICP metrics, one might think that to achieve a similarly high PICP, the width of MC dropout prediction intervals would generally be larger than the width of the prediction intervals derived from quantile regression.

In Section 3.1, we mentioned the importance of visualization, caused by the skewness of data. We anticipate that the dataset contains many zero values. If the quantile regression model can cover these points with high certainty, it could explain the relatively low MPIW compared to the MC dropout method. Figure 4.5 supports this observation. We compare quantile regression to the *Encoder + ConvLSTM(cell, locked) + Decoder* variant of MC dropout with 0.2 and 0.4 dropout probabilities, as they also exhibit strong PICP and could be useful in specific use cases.

We have already mentioned the difference between frequentist and Bayesian paradigm. Figure 4.5 illustrates this: MC dropout is able to develop uncertainty in every point, while quantile regression keeps it very low in certain areas where there was little to no precipitation. This characteristic is advantageous for MC dropout as it allows for a more comprehensive understanding of uncertainty throughout the entire spatial domain.

Even with the high MPIW and PICP metrics of quantile regression, MC dropout seems to model the uncertainty in the model better, as quantile regression could potentially underestimate the uncertainty in the low precipitation density areas.

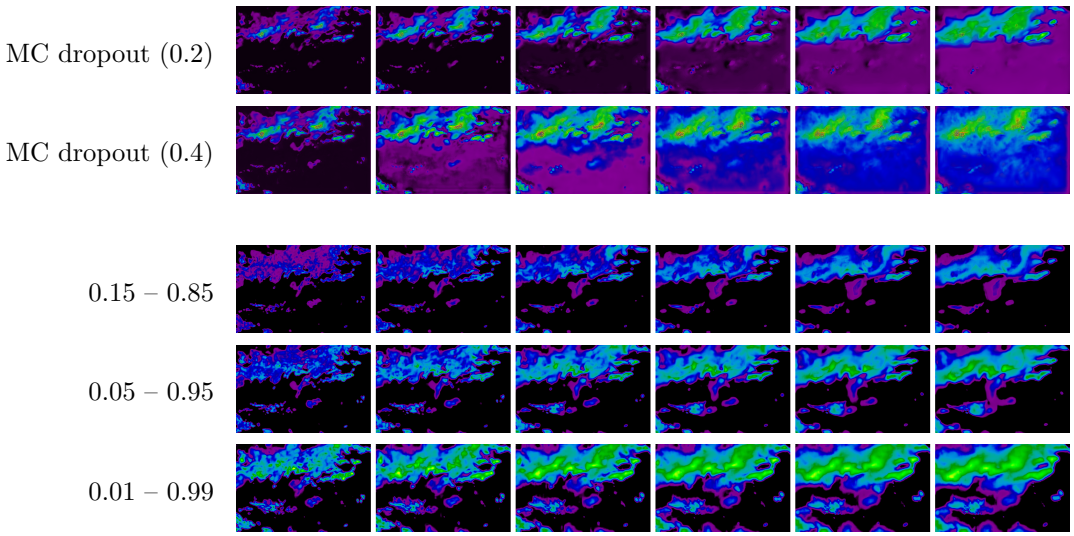

Figure 4.5 Comparison of MC dropout interval widths and the interaval widths obtained by using the quantile regression.

Conclusion

In this work, we summarized various methods used for uncertainty quantification, with emphasis on MC dropout and quantile regression. Our goal was to gain a deeper understanding of different approaches, their benefits, and their limitations. We have discussed implementing the uncertainty in the PhyDNet, a state-of-the-art model for video prediction tasks. Not only have we discussed how this model operates, but we also discussed possible obstacles when implementing the uncertainty in it.

Since methods of predicting the distribution of the output may be computationally expensive, approximate techniques like MC dropout serve as a simple, yet effective alternative. It enables the integration of uncertainty quantification into existing deep learning models without significantly increasing the computational burden. This makes it a practical choice for researchers and practitioners seeking to enhance their models' performance while managing resource constraints.

In our implementation, the use of dropout didn't bring any improvements when used as a regularization technique during training. This is most likely due to sufficient implicit regularization of the model and other explicit regularization methods that have previously been employed.

The MC Dropout method offers a straightforward approach to uncertainty quantification. We have examined two distinct strategies: incorporating dropout during both training and inference or solely during the inference process. The first approach leads to very narrow intervals with rather small coverage probabilities, suggesting that the second option or a combination of the two might be more effective.

We have found that for our specific problem, the prediction of the model doesn't have to rely on the residual predictor of PhyDNet too much, so implementing dropout into it during training may cause its demise in the final prediction. This also speaks for the second strategy of implementing MC dropout, since high dependence on PhyCell leads to very narrow intervals, with almost zero coverage probability.

With quantile regression, we were able to achieve notably high coverage probability. Although the metrics indicate that the widths of the prediction intervals resemble those of the MC dropout method, our analysis demonstrates that when employing quantile regression, the prediction intervals tend to be very narrow with the prediction near zero and wider anywhere else. This is caused by the frequentist nature of the method, and may cause underestimation of uncertainty in areas with low precipitation density. MC dropout, on the other hand, is able to quantify uncertainty throughout the entire spatial domain.

We have also discussed another drawback of quantile regression — to change the model slightly, we need two new models and their training may take a lot of time and resources.

In this work, we proposed two different methods of implementing uncertainty quantification into the PhyDNet model for spatio-temporal predictions. We have compared the methods and by presenting empirical results alongside real-world examples, we have provided a comprehensive understanding of these approaches' impact on uncertainty quantification.

Bibliography

1. 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*. IEEE, 2020, pp. 11471–11481. ISBN 978-1-7281-7168-5.
2. HRABOVSKY, George; SUSSKIND, Leonard. *Classical mechanics: the theoretical minimum*. Penguin UK, 2020. ISBN 978-0-141-97622-8.
3. MCCULLOCH, Warren S; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. In: *The bulletin of mathematical biophysics*. Springer, 1943, vol. 5, pp. 115–133. Available from DOI: 10.1007/BF02478259.
4. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning* [online]. MIT Press, 2016 [visited on 2023-04-19]. Available from: http://www.deeplearningbook.org.
5. OLAH, Christopher. *Understanding LSTM Networks* [online]. 2015. [visited on 2023-04-14]. Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.
6. PASCANU, Razvan; MIKOLOV, Tomas; BENGIO, Yoshua. On the difficulty of training recurrent neural networks. In: *International conference on machine learning*. Pmlr, 2013, pp. 1310–1318. Available from DOI: 10.48550/arXiv.1211.5063.
7. BENGIO, Yoshua; SIMARD, Patrice; FRASCONI, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*. 1994, vol. 5, no. 2, pp. 157–166. Available from DOI: 10.1109/72.279181.
8. HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long short-term memory. *Neural computation*. 1997, vol. 9, no. 8, pp. 1735–1780.
9. GREFF, Klaus; SRIVASTAVA, Rupesh K; KOUTNÍK, Jan; STEUNEBRINK, Bas R; SCHMIDHUBER, Jürgen. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*. 2016, vol. 28, no. 10, pp. 2222–2232.
10. SHI, Xingjian; CHEN, Zhourong; WANG, Hao; YEUNG, Dit-Yan; WONG, Wai-Kin; WOO, Wang-chun. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*. 2015, vol. 28.
11. WEIDMAN, Seth. *Deep Learning from Scratch: Building with Python from First Principles*. O'Reilly Media, 2019.
12. DUMOULIN, Vincent; VISIN, Francesco. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*. 2016.

13. GAWLIKOWSKI, Jakob; TASSI, Cedrique Rovile Njieutcheu; ALI, Mohsin; LEE, Jongseok; HUMT, Matthias; FENG, Jianxiang; KRUSPE, Anna; TRIEBEL, Rudolph; JUNG, Peter; ROSCHER, Ribana, et al. A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*. 2021.

14. MATTHIES, Hermann G. Quantifying uncertainty: modern computational representation of probability and applications. In: *Extreme man-made and natural hazards in dynamics of structures*. Springer, 2007, pp. 105–135.

15. BUIZZA, Roberto. *Chaos and Weather Prediction*. ECMWF, 2002.

16. WISTON, Modise; MPHALE, Kgakgamatso. Weather Forecasting: From the Early Weather Wizards to Modern-day Weather Predictions. *Journal of Climatology & Weather Forecasting*. 2018, vol. 06. Available from DOI: 10.4172/2332-2594.1000229.

17. KENDALL, Alex; GAL, Yarin. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In: GUYON, I.; LUXBURG, U. Von; BENGIO, S.; WALLACH, H.; FERGUS, R.; VISHWANATHAN, S.; GARNETT, R. (eds.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, vol. 30. Available also from: https://proceedings.neurips.cc/paper_files/paper/2017/file/2650d6089a6d640c5e85b2b88265dc2b-Paper.pdf.

18. BLUNDELL, Charles; CORNEBISE, Julien; KAVUKCUOGLU, Koray; WIERSTRA, Daan. Weight uncertainty in neural network. In: *International conference on machine learning*. PMLR, 2015, pp. 1613–1622.

19. JOSPIN, Laurent Valentin; LAGA, Hamid; BOUSSAID, Farid; BUNTINE, Wray; BENNAMOUN, Mohammed. Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users. *IEEE Computational Intelligence Magazine*. 2022, vol. 17, no. 2, pp. 29–48. Available from DOI: 10.1109/MCI.2022.3155327.

20. IZMAILOV, Pavel; VIKRAM, Sharad; HOFFMAN, Matthew D; WILSON, Andrew Gordon Gordon. What are Bayesian neural network posteriors really like? In: *International conference on machine learning*. PMLR, 2021, pp. 4629–4640.

21. MURAD, Abdulmajid; KRAEMER, Frank Alexander; BACH, Kerstin; TAYLOR, Gavin. Probabilistic deep learning to quantify uncertainty in air quality forecasting. *Sensors*. 2021, vol. 21, no. 23, p. 8009.

22. LAKSHMINARAYANAN, Balaji; PRITZEL, Alexander; BLUNDELL, Charles. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*. 2017, vol. 30.

23. HINTON, Geoffrey; VINYALS, Oriol; DEAN, Jeff. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*. 2015.

24. MADDOX, Wesley J; IZMAILOV, Pavel; GARIPOV, Timur; VETROV, Dmitry P; WILSON, Andrew Gordon. A simple baseline for bayesian uncertainty in deep learning. *Advances in neural information processing systems*. 2019, vol. 32.

25. RUDER, Sebastian. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*. 2016.

26. IZMAILOV, Pavel; PODOPRIKHIN, Dmitrii; GARIPOV, Timur; VETROV, Dmitry; WILSON, Andrew Gordon. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*. 2018.

27. ZHANG, Chiyuan; BENGIO, Samy; HARDT, Moritz; RECHT, Benjamin; VINYALS, Oriol. *Understanding deep learning requires rethinking generalization*. 2017. Available from arXiv: 1611.03530 [cs.LG].

28. SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 2014, vol. 15, no. 56, pp. 1929–1958. Available also from: `http://jmlr.org/papers/v15/srivastava14a.html`.
29. GHIASI, Golnaz; LIN, Tsung-Yi; LE, Quoc V. Dropblock: A regularization method for convolutional networks. *Advances in neural information processing systems*. 2018, vol. 31.
30. TOMPSON, Jonathan; GOROSHIN, Ross; JAIN, Arjun; LECUN, Yann; BREGLER, Christoph. Efficient object localization using convolutional networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 648–656.
31. GAL, Yarin; GHAHRAMANI, Zoubin. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.
32. HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J.H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009. Springer series in statistics. ISBN 9780387848846.
33. KOENKER, Roger; BASSETT JR, Gilbert. Regression quantiles. *Econometrica: journal of the Econometric Society*. 1978, pp. 33–50.
34. IOFFE, Sergey; SZEGEDY, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: *International conference on machine learning*. pmlr, 2015, pp. 448–456.
35. LI, Xiang; CHEN, Shuo; HU, Xiaolin; YANG, Jian. Understanding the disharmony between dropout and batch normalization by variance shift. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 2682–2690.
36. WU, Yuxin; HE, Kaiming. Group normalization. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.
37. PASZKE, Adam et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. Available also from: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.
38. GAL, Yarin; GHAHRAMANI, Zoubin. A theoretically grounded application of dropout in recurrent neural networks. *Advances in neural information processing systems*. 2016, vol. 29.
39. ORR, Genevieve; SCHRAUDOLPH, Nici; CUMMINS, Fred. *Momentum and Learning Rate Adaptation* [online]. 1999. [visited on 2023-05-03]. Available from: `https://www.willamette.edu/~gorr/classes/cs449/momrate.html`.
40. QIAN, Ning. On the momentum term in gradient descent learning algorithms. *Neural networks*. 1999, vol. 12, no. 1, pp. 145–151.
41. KINGMA, Diederik P; BA, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2014.
42. WANG, Zhou; BOVIK, Alan C; SHEIKH, Hamid R; SIMONCELLI, Eero P. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*. 2004, vol. 13, no. 4, pp. 600–612.
43. HUNTER, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007, vol. 9, no. 3, pp. 90–95. Available from DOI: `10.1109/MCSE.2007.55`.

Contents of attachments

- `models` ..... directory of implemented PhyDNet architectures
  - `base` ..... standard PhyDNet implementation
  - `locked-dropout-masks` ..... PhyDNet implementation with locked dropout masks
- `train.py` ..... training script
- `README.md` ..... basic instructions for running the attached programs
- `phydnet-hyperparams.yaml` ..... file with configuration for the model
- `metrics.py` ..... script for model evaluation
- `inference.py` ..... script for model uncertainty evaluation and image logging
- `env.yml` ..... specification of conda virtual environment
- `dataset.py` ..... script for work with data
- `text`
  - `src` ..... directory with thesis LaTeX source files
  - `thesis.pdf` ..... thesis text in PDF format