UNIVERSITY OF BERGEN

MASTERS THESIS

MetZoom: A CNN/LSTM hybrid based model for water reservoir inflow prediction

---

Candidate:

Halvor H. BARNDON

Supervisor:

Dr. Troels Arnfred BOJESEN

UNIVERSITY OF BERGEN

A thesis submitted in fulfillment of the requirements

for the degree of Masters of Science

in the

Machine Learning Group

Department of Informatics

UNIVERSITY OF BERGEN

Abstract

Faculty of Mathematics and Natural Sciences

Department of Informatics

Masters of Science

MetZoom: A CNN/LSTM hybrid based model for water reservoir inflow prediction

by Halvor H. BARNDON

Hydropower reservoir volumes fluctuate as water levels increase or decrease according to precipitation, valve output and inflow through water retained in the surrounding area. Predicting these fluctuations with *machine learning* is possible through the use of an Artificial Neural Network (ANN) architecture proposed in this thesis. The neural network model aims to forecast the changes in relative water level for a reservoir managed by Saudefaldene, a hydropower company in Rogaland, Norway. The predictions are made through the use of radar images reflecting the precipitation rate, and a dataset provided by Saudefaldene. The provided dataset contains the precipitation history, valve-opening records and relative water levels across 2014-2021. Such a forecast can have various impacts on hydropower reservoir management, which lay the foundation for the thesis.

The architecture proposed in this thesis, namely MetZoom, contains a Convolutional Neural Network (CNN) architecture which predicts future precipitation rates in the form of radar image replications and precipitation

up to 12 hours ahead. The use of radar images is motivated by the intent to forecast precipitation as a tool for predicting changes in the relative water level. The predictions made by the CNN are forwarded to a Recurrent Neural Network (RNN) in the form of a Long Short-Term Memory (LSTM) network to learn the fluctuations of reservoir water levels. The architecture of Met-Zoom is a result of several tested CNN and RNN models and a combination of these.

Acknowledgements

Throughout writing this thesis I have received support and guidance, and for this I would like to thank the following.

Firstly I would like to thank my supervisor, Dr. Troels Arnfred Bojesen, whose patience, guidance and availability has been outstanding. Not only by formulating and concretizing the research questions, but also providing insight and advice which made this thesis possible. I would specifically like to thank Dr. Bojesen for his humour and spirit which has provided an ambiance loaded with positivity that has helped immensely with motivation.

I would like to thank the staff at Saudefaldene, who welcomed us to the hydropower facility and offered a guided tour. A special thanks to Ole Håkon Hovland for his interest, availability and assistance. The idea behind the thesis belongs to Ole Håkon, and I greatly appreciate the opportunity given to me by his eagerness to explore the possibilities of machine learning.

I would like to acknowledge my fellow students who have taken part in many thorough discussions and have provided technical assistance from time to time. I would like to thank Hans Martin Theigler Johansen specifically for his open ear and extensive support, which has helped me overcome many technical obstacles along the way. Also a special thanks to Knut Thormod Aarnes Holager, Mathias Larsson Madslien and Emir Zamwa for many great pieces of advice and joyful moments.

Finally, I want to give gratitude to my parents, Randi Barndon and Rolf Helland, whose counsel and support has provided a great deal of sympathy and motivation. None of the paths taken towards even starting this thesis would have been possible without them, and for that I am thankful.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN Artificial Neural Network. 2, 10, 11, 13, 26, 29, 41, 42, 61, 64, 108, 109

ARIMA AutoRegressive Integrated Moving Average. 7, 8, 89, 90, 94, 95, 101, 109

BPTT backpropagation through time. 44

CNN Convolutional Neural Network. 3, 4, 29, 30, 36, 38, 39, 64, 65, 66, 67, 68, 69, 71, 72, 73, 74, 75, 76, 77, 78, 83, 106

FFNN Feed-Forward Neural Network. 41, 61, 62

HREF High-Resolution Ensemble Forecast. 59

HRRR High-Resolution Rapid Refresh. 59

HRW Highest Regulated Water Level. 51

LSTM Long Short-Term Memory. 42, 43, 44, 63, 64, 65, 75, 76, 78, 79, 89, 90, 97, 101, 106, 109

MAE Mean Absolute Error. 87

MET The Norwegian Meteorological Institute. 45

MSE Mean Squared Error. 87

NetCDF Network Common Data Form. 45, 46

NNAR Neural Network Autoregression. 8

NWP Numerical Weather Prediction. 4, 9, 59, 60

RNN Recurrent Neural Network. 3, 4, 41, 42, 44, 60, 62, 64, 75, 99

SE Standard Error. 87

TDS THREDDS Data Service. 45

Chapter 1

Introduction and motivation

The hydropower company Saudefaldene, in Sauda, Rogaland, manages a network of water reservoirs and water transportation tunnels. These reservoirs and tunnels, combined with measuring stations (which record a variety of measurements at different locations), present the opportunity for a machine learning model which can learn an embedded pattern in this data. The aim is to predict changes in the relative water level for a water reservoir through these patterns. The relative water level is the surface height relative to a set zero value. The model should *ideally* consider recent precipitation, water retention in the surrounding area, temperature causing evaporation and the rate of snow melting – which all affect the relative water level. More realistically, attempts will be made to focus on precipitation as a main source of inflow, and consider as a future development the inclusion of the other sources of inflow. Note that although these sources of water are omitted from the dataset, a machine learning model can indirectly learn these sources. The exact nature of the data is described in Chapter 4.

In this thesis only one reservoir is targeted by the proposed machine learning model, whilst in theory, a more comprehensive model could provide forecasts for any such reservoir, or a collection of reservoirs. The targeted reservoir is called Helgedalsvatnet (Figure 1.1), which lies at the highest altitude of all reservoirs managed by Saudefaldene. The main goal of the project

will be predicting hourly changes of the relative water level in Helgedalsvatnet, relative to a set zero value. This predictor value is physically dependent on precipitation, which can be represented with a set rate, bound by the geographical precipitation field as seen in Figure 1.1.

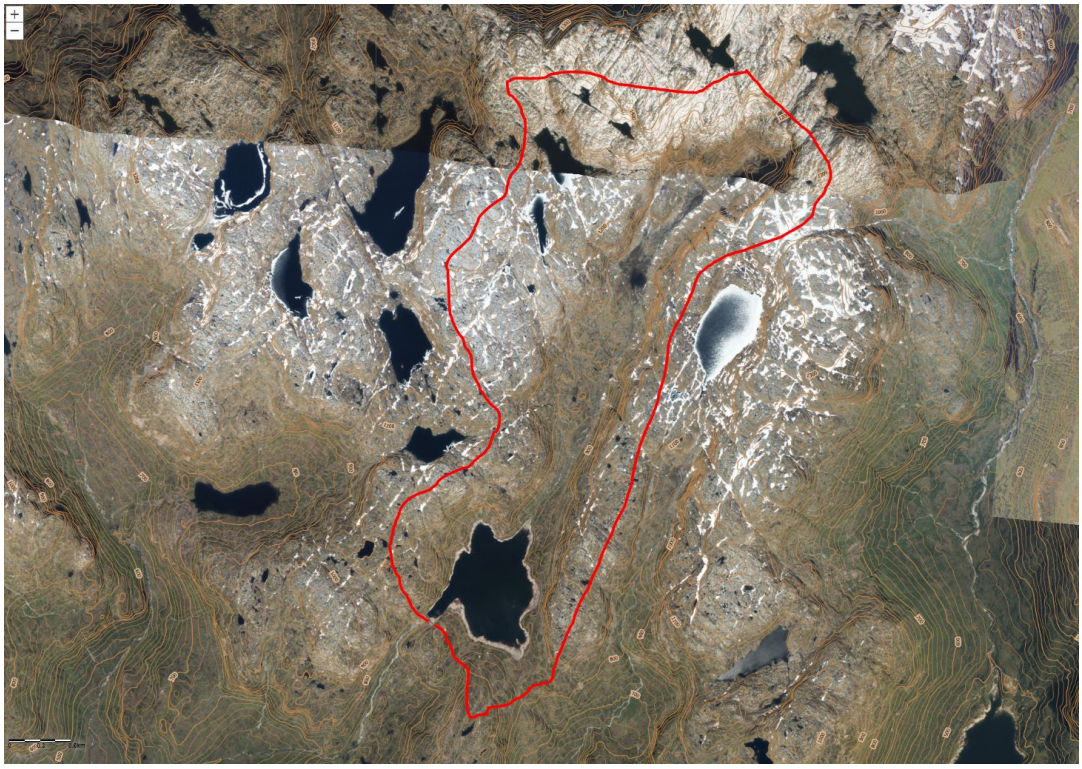

FIGURE 1.1: An image of Helgedalsvatnet with an outline of its precipitation field. The image is provided by Saudefaldene.

The altitude of the reservoir excludes water inflow from other reservoirs, as well as a low general runoff time. This simplifies the complexity of the task, as fewer physical dependencies are accounted for by the proposed model. This thesis is a pilot project suggested by Saudefaldene for reviewing whether an Artificial Neural Network (ANN) can predict these changes. The project also incorporates radar images which reflect precipitation. By using the service provided by the Norwegian Meteorological Institute, Thredds, it is possible to obtain a time series of radar images reflecting precipitation rate across the relevant area.

The radar images can be used as an asset along with a dataset provided by Saudefaldene, as the radar images reflect precipitation coming into and traveling across the relevant geographical area providing an amount of precipitation. The dataset provided by Saudefaldene contains information from 2014 through 2021, including recorded precipitation and temperature in Sauda, along with relative water levels and valve openings for Helgedalsvatnet. Whilst working with data in a time series structure it is important to find a solution for combining the data from several sources, and fitting them into a dataset with temporal cohesion between the variables. There is also a set of assumptions made in this thesis, as the main focal point is such a small geographical area. It is assumed that the weather that affects the water levels most is local weather, and therefore the radar images only cover a small area as pictured in Chapter 4, Figure 4.2. Chapter 4 explains how the dataset is stipulated to match in the desired time intervals, and these alterations include the assumption of a smoother relationship between weather and inflow than what is expected of the chaotic nature of weather.

A machine learning model which predicts changes in relative water level can prove to be beneficial for both management and scheduling for the hydropower company. Said model could also be applicable in other scenarios, including recognition of dam overflow or risk of dangerous flooding caused by heavy precipitation. In this thesis the main focus is capturing changes of water levels in the imminent future, within a few hours or a day.

The architecture proposed in this thesis employs two neural network model types, which will be further discussed in Chapter 3. To achieve a forecast using machine learning and the available data a combination of a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) is used. Both have proven to be efficient at the two tasks at hand, namely image processing with a CNN and sequence prediction with a RNN.

Convolutional Neural Networks have proven to perform well on image

segmentation and classification problems, as well as value regression problems from images - such as counting the amount of trees in an image of a forest, or the distance between cities on a map. As weather forecasting is predicting spatiotemporal features, the product of said prediction must be mapping these features to a observable meterological feature (e.g temperature, precipitation, humidity, wind, etc.). The main meteorological feature in this thesis is precipitation, which is likely the easiest to observe and measure. Rather than combatting Numerical Weather Predictions (NWPs) and proposing a neural network that learns patterns of precipitation, the study proposes the learnability of how the precipitation rate in radar images affects the change in relative water levels in reservoirs.

Recurrent Neural Networks are both fast and accurate for sequence prediction and are recognized as proficient in token alignment problems, for example learning the next letter in a word, or the next word in a sentence.

The model in this thesis is a combination of the two. It uses a RNN for the time series data to understand the long term relationships in water inflow, and a CNN for capturing expected imminent changes through the radar images. The thought process behind building the CNN is capturing the precipitation moving towards the water reservoir. The RNN can use this information to reinforce the relationship between the captured precipitation reflected by the radar images and the following change in relative water level.

The base motivation behind creating such a model, and hence the thesis itself, is providing information on both the accuracy and potential of a machine learning model trained on radar images, meterological- and reservoir data for prediction of water levels in hydropower reservoirs.

Chapter 2

Time series, Forecasting and Weather

This chapter will explain the conceptual structure of time series, traditional methods of forecasting, and review problems when forecasting and predicting weather. Understanding forecasts provides a starting point for handling the problem in this thesis. Throughout this chapter time series and forecasting are largely explained on the basis of *Forecasting: principles and practice* (Hyndman and Athanasopoulos, 2018).

2.1 Forecasting using time series

"Forecasting is about predicting the future as accurately as possible, given all of the information available, including historical data and knowledge of any future events that might impact the forecasts" (Hyndman and Athanasopoulos, 2018, p. 14).

2.1.1 Time series

A time series is any form of information captured over a period of time, preferably at regular intervals. Often the information captured in a time series displays a pattern which can be *cyclic*, *seasonal*, *trending* or a combination

of these.

Cyclic information exhibits some increase and decrease without a fixed frequency. Cyclic fluctuations can be seen in both long- and short time frames, such as weekly or annually. Seasonal patterns occur when the information exhibits changes based on periodic factors such as the time of the day, the day of the week or the month of the year. As these intervals are fixed, the frequency is known and can be accounted for. A trend describes a dominating increase or decrease in the information covered by the time series data. A trend can also change, but remains a trend if it does not repeatably change at recognizable intervals.

Notably, more often than not, time series present patterns that are a combination of cyclic, seasonal or trending data.

2.1.2 Time series forecasting

The objective of time series forecasting is to exploit the patters present in the time series to make predictions for the future. Quantitative forecasting methods are used when adequate data is available for representing a long enough period of the past, and it is reasonable to assume that patterns apparent in the numerical information of the past will continue for the period that is covered by the forecast. When deciding on a quantitative forecasting method one must consider which properties of the data are important. The predictor variable is often known prior to deciding on forecasting, while other variables that might affect the accuracy of the forecast are more obscure. By considering variables that are more important for an accurate forecast, the prediction interval (range of certainty) can be narrowed. Time series data is used in most problems where quantitative forecasting methods are appropriate (Hyndman and Athanasopoulos, 2018, p. 15-17).

If one can accurately predict an outcome in the future based on time series data - the goal of time series forecasting is achieved. More complex time series models capture more information than one-variable trends or patterns. Some models used in this type of forecasting are; decomposition models exponential smoothing models and AutoRegressive Integrated Moving Average (ARIMA) models (Hyndman and Athanasopoulos, 2018). The latter will be used as a comparison for measuring the performance of the Neural Network proposed in this thesis (see Chapter 6).

2.1.3 ARIMA models

ARIMA models are a combination of an autoregressive and a moving average model. Autoregressive models provide an output from a linear regression on a set of previous values and a stochastic term. The following description is from Hyndman and Athanasopoulos, 2018, p. 221-270. ARIMA models are used for describing autocorrelations in data. The model is regressive on the predictor variable itself. These models are flexible and can handle a range of different time series. An autoregressive models predictions for time t with order p can be written as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \epsilon_t \tag{2.1}$$

where ϵ_t is a white noise term and $\phi_1, \ldots, \phi_p$ are the parameters of the model.

A moving average model uses past forecast errors in a similar regression. Here the value y_t is the weighted moving average with order q, and $\theta_1, \ldots, \theta_p$ are the parameters of the model:

$$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q} \tag{2.2}$$

When combined with computation of differencing between consecutive

observations the two form the ARIMA model. The integration is the reverse of differencing. Such a model can be written as:

$$y'_t = c + \phi_1 y'_{t-1} + \phi_2 y'_{t-2} + \ldots + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q} + \epsilon_t \tag{2.3}$$

Where y'_t is the differenced series. According to Hyndman and Athanasopoulos, 2018, this is recognized as an ARIMA(p, d, q) model where p is the order of the autoregression, d is the degree of differencing of the series and q is the order of the moving average. The most fitting values for p, d and q are selected either by inspecting the data for a problem and differencing until the data becomes stationary, or using built-in automatic functions with ARIMA libraries in programming.

2.1.4 Use of neural networks in forecasting

Neural networks (which are covered in depth in Chapter 3) are used in forecasting as models for representation of complex nonlinear relationships. With the lagged values in time series data a neural network can be used to autoregressively forecast values, such as weather. These neural networks are often referred to as Neural Network Autoregression (NNAR) models, and are nearly equivalent to ARIMA models. NNAR models, in contrast to ARIMA models, are free of restrictions of parameters (Hyndman and Athanasopoulos, 2018, p. 335-337).

Conventional forecasting models are applied in iterations. For every prediction a step forward in time is taken, and all the historical information up until this point is available. A prediction is calculated both with past information and the outputs of the model. This operation is repeated until all values of the forecast are predicted. This is an iterative forecasting method. Neural networks rather use a direct forecasting method, and uses n-output

nodes as a representation of n-steps prediction. This allows for neural networks to only rely on past information for predictions (Coulibaly, Anctil, and Bobée, 2000).

2.2 The chaotic nature of weather data

Prior to consuming information on weather predictions and forecasts it is important to understand the chaotic nature of the data behind these predictions. Because of the physics and dynamics of the natural weather system and its inherent unpredictability on a longer time scale, even tiny differences in initial conditions, or the *initial state of the atmosphere*, can grow vastly different. This provides a set of challenges in prediction of weather. Even in commonly used methods, such as NWP models, uncertainties can grow as prior forecasts are not deterministic, and often sparse or low quality information is shuffled between high quality and dense weather data (Buizza, 2002, p. 2-5).

In weather data the initial conditions are all approximately known as they depend on instrumental accuracy (Buizza, 2002, p. 3). Errors made by instruments that measure wind, precipitation or other meteorological data can provide small uncertainties. Two separate sets of initial states of the atmosphere can diverge from each other quickly. Even with a set of perfect and replete data, meaning it covers all time steps and have the correct information, one cannot simulate all of the factors, or molecules at hand. As a result it is nearly impossible to achieve perfect results in weather prediction for any space and time scale.

Even the flow, movement and amount of weather is very unpredictable and as such even state of the art forecasting models have not achieved near perfect results.

Chapter 3

Machine Learning

Machine learning models analyze data through the use of pattern identification and decision making. Machine learning models are taught to produce representations in data autonomously through learning. The learning process is based on building experience from and understanding the given data. Arthur Samuel coined the term in 1959, proposing that machine learning allows computers to complete tasks without being explicitly programmed to do so (Samuel, 1959). Today machine learning has many applications, and is considered a promising tool for the suggested problem of the thesis. To better understand the machine learning tools used in this thesis, this chapter will explain some of the basic functionalities in artificial neural networks, the foundation for the neural network architecture used in this project. The focus is directed towards the operations of convolutional- and recurrent neural networks, both of which are deemed efficient on their own tasks.

3.1 Artificial Neural Networks

ANNs are versatile tools and are often used for approximating nonlinear functions in n-dimensional data. The architecture of a neural network describes how a networks components connect to each other. Throughout this thesis the focus is supervised learning as opposed to other machine learning

approaches. Supervised learning is based on a set of inputs X, their corresponding labels Y and the networks parameters *weights* W and *bias* B. Supervised learning models learn a parameterized function $f : X \rightarrow Y$ which given a sample x_i provides an output $f(x_i) = \hat{y}_i$, and is trained through altering its weights according to the label y_i. In this section the focus is the components of an ANN and how a network can be trained to provide more precise outputs in the form of predictions $\hat{Y}$.

3.1.1 Artificial Neurons

A neural network is a set of linked *neurons*. The primary objective for a neuron is to pass an input signal to an output signal, as shown in Figure 3.1.

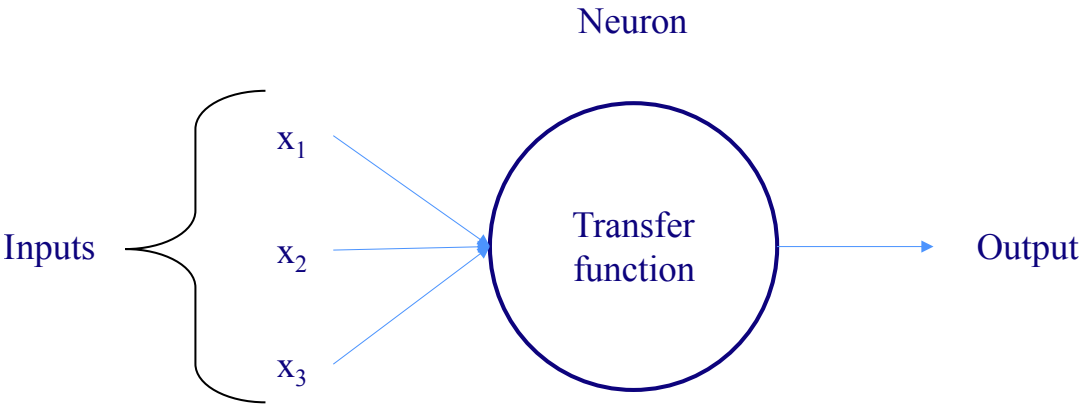

FIGURE 3.1: A neurons transfer function receives an input signal consisting of several features and produces an output. In this example there are three inputs to the node, and the output is the weighted sum of the neurons weights and the input.

Information is passed through the networks neurons. Every neuron has an internal *transfer function*. The transfer functions for a set of neurons are not necessarily equal, which allows for flexibility in computation. The transfer function for a single neuron can be written as:

$$t_n(X) = \sum_i x_i w_i^{(n)} \tag{3.1}$$

The neuron ns input signals for the neuron $x_0, \ldots, x_i \in X$ are multiplied with the networks weights for that neuron $w^{(n)} \in W$, and the transfer function t sum up the different weighted values, as shown in Equation (3.1). The weights aim to define which features of the input data are important for the neurons. To achieve this, the weights increase the important features and reduce the non important features, pushing the network towards a correct prediction (Haykin and Network, 1999).

In many neural network applications, some bias is introduced. Bias is similar to the intercept of a linear function, and helps to adjust the output along with the weights. The bias can either be learned, or added as a constant to best fit the data.

3.1.2 Activation Functions

The weighted sum of every neuron is passed through an *activation function* as shown in the figure below (Figure 3.2) (Maass, 1997).

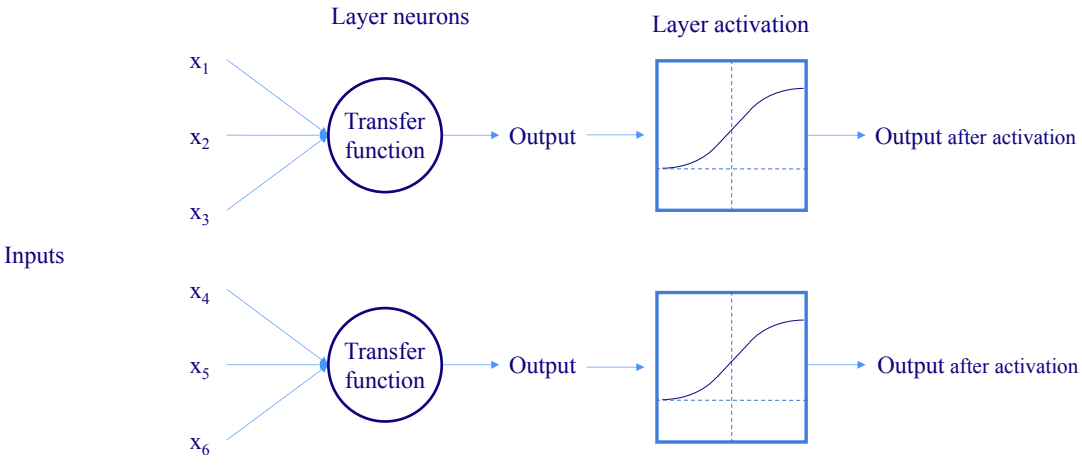

FIGURE 3.2: Output from two neurons in a layer is passed through the activation function before being passed to the next layer of the network

The activation function A (or set of such functions) in a neural network transform an input signal to an output z. The input signal to an activation function is the direct output of a neurons, or set of neurons (as shown in

Figure 3.2 and in Equation (3.2)). The neurons included in the equation are all *activated*.

$$z = A(b + \sum_i x_i w_i) \tag{3.2}$$

This output signal is called the *activation* of a neuron, and is the weighted sum of the neurons inputs and weights, along with the bias b (as seen in Equation (3.2)).

A review of the nonlinearity of ANNs lies in the process of neurons described by Krenker, Bešter, and Kos, 2011 in Equation (3.3):

$$y = A\Big(\sum_{i=0}^{m} w_i \cdot x_i + b\Big) \tag{3.3}$$

Where x_i is the input value where i goes from 0 to m, w_i is the weight value, b is bias, A is an activation function and y is the output value.

The following section is largely based on the work in "Activation functions in neural networks" by Sharma, Sharma, and Athaiya, 2017.

The activation function affects the performance and accuracy of a neural network. Most common are non-linear activation functions, as these are the basis of nonlinearity in neural networks.

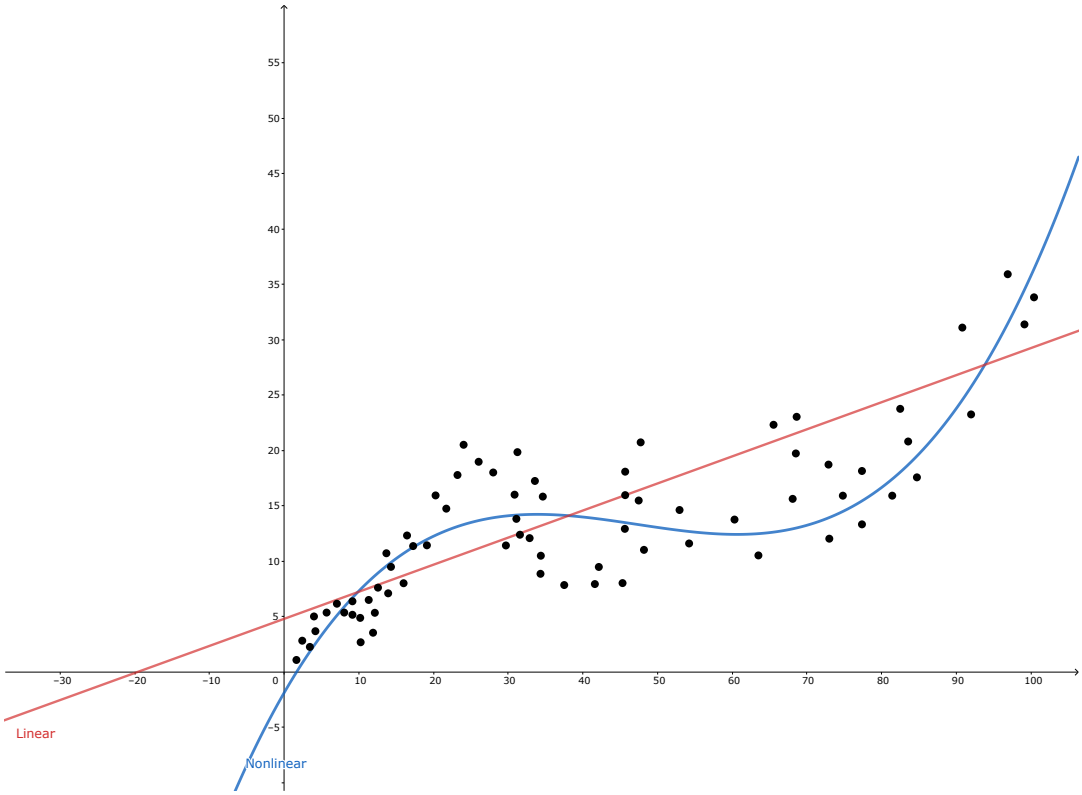

FIGURE 3.3: A linear function fits the data points poorly compared to a nonlinear function, as the trend fluctuates more than a linear function can represent.

Without nonlinear activation functions, a network resembles a linear regressor which is unable to capture nonlinearity of its input data, as shown in Figure 3.3. Similarly, using nonlinear activations for approximating a linear relationship is also sub-optimal, and for some problems a combination of linear and nonlinear activations amidst layers are preferable.

Some of the more essential activation functions for this thesis are the Sigmoid, Tanh, Rectified Linear Unit and Leaky Rectified Linear Unit functions. For more activation functions prominent in artificial neural networks, see "Activation functions in neural networks" by Sharma, Sharma, and Athaiya, 2017.

The *Sigmoid* activation function is used in many machine learning applications and can be defined as

$$A(x) = \frac{e^x}{1 + e^x} \tag{3.4}$$

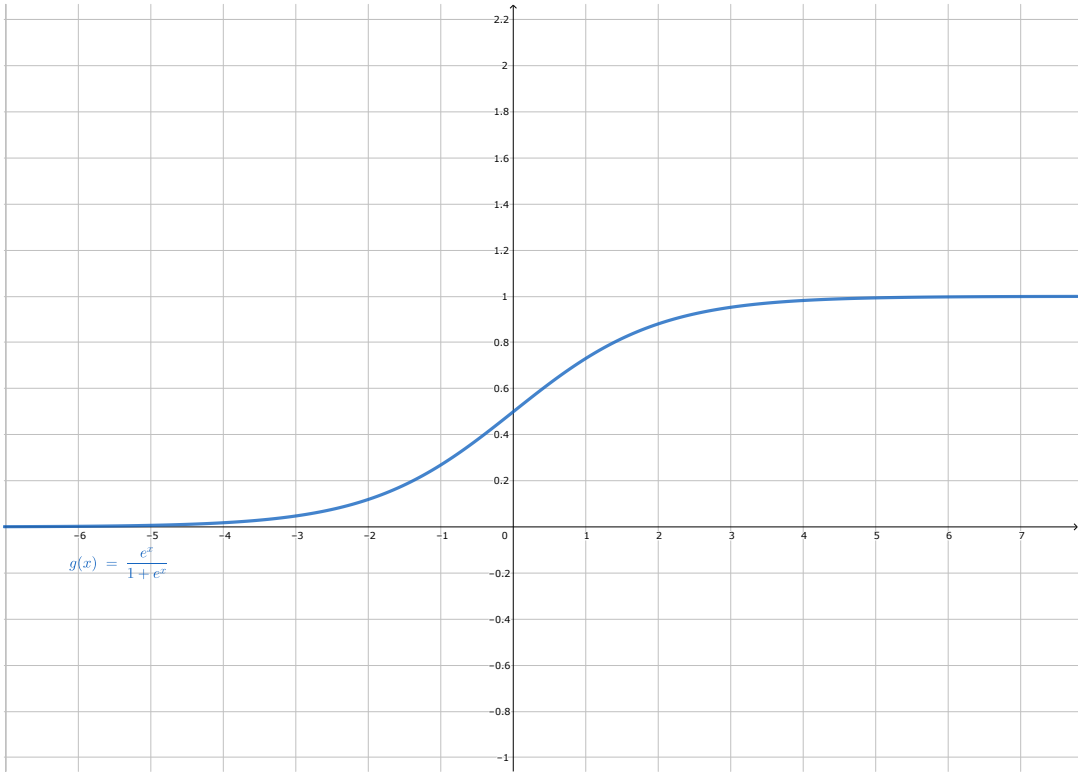

FIGURE 3.4: The Sigmoid Activation Function scales values to fit between 0 and 1.

It is non-linear and transforms values to the range $[0, 1]$. It is continuously differentiable and provides an *s*-shaped curve. Whilst the sigmoid function is symmetric around $(x, y) = (0, 0.5)$, it can be scaled and shifted such that it is symmetric around other values, as to affect the output values. The sigmoid function is often used in problems targeting probabilities, or representation of a level of confidence, as the values are within $[0, 1]$.

The *Tanh* activation function, or the *Hyperbolic Tangent*, resembles the sigmoid activation. In opposition to the sigmoid activation, the tanh function is symmetric around a predefined origin, normally zero. The tanh function is defined as

$$A(x) = \frac{\sinh(x)}{\cosh(x)}. \tag{3.5}$$

It retains the same *s*-shaped curve as the sigmoid function (as seen in Figure 3.5), but has values in the range $[-1, 1]$.

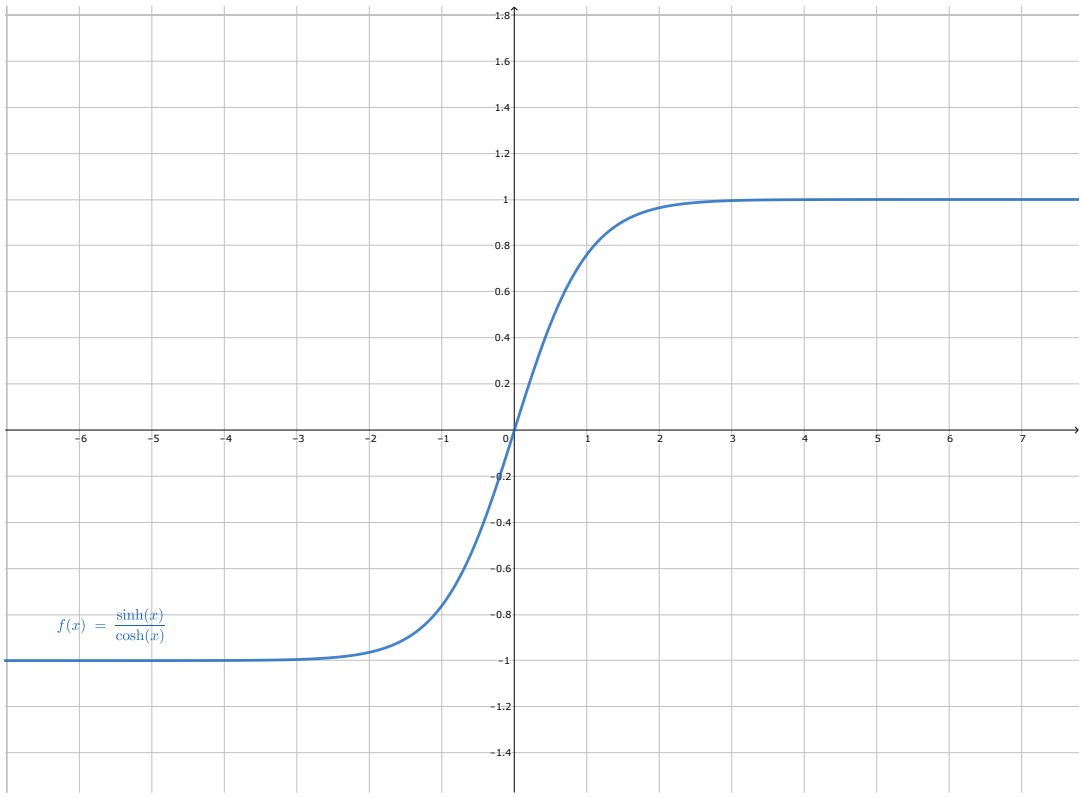

FIGURE 3.5: The Tanh Activation Function scales values to fit between -1 and 1.

The *Rectified Linear Unit*, or *ReLU*, activation function does not necessarily activate all neurons simultaneously. It activates inputs above 0 and is defined as

$$f(x) = \max(0, x) \tag{3.6}$$

and illustrated in Figure 3.6.

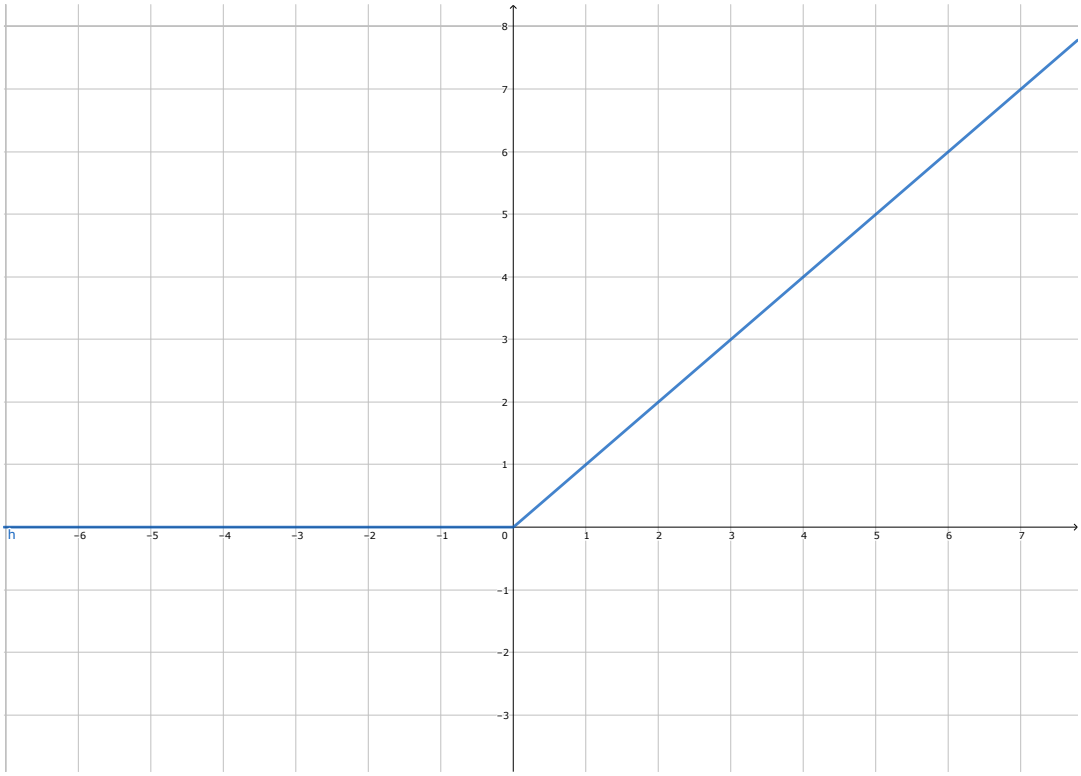

FIGURE 3.6: The ReLU Activation Function linearly activates non-zero values.

Several activation functions are closely related to ReLU, notably *Leaky ReLU*, which will be used along-side traditional ReLU in this thesis. The leaky version changes values of x that are below zero to a small component of x, e.g. $0.1x$ (as illustrated in Figure 3.7).

$$f(x) = \max(0.1x, x) \tag{3.7}$$

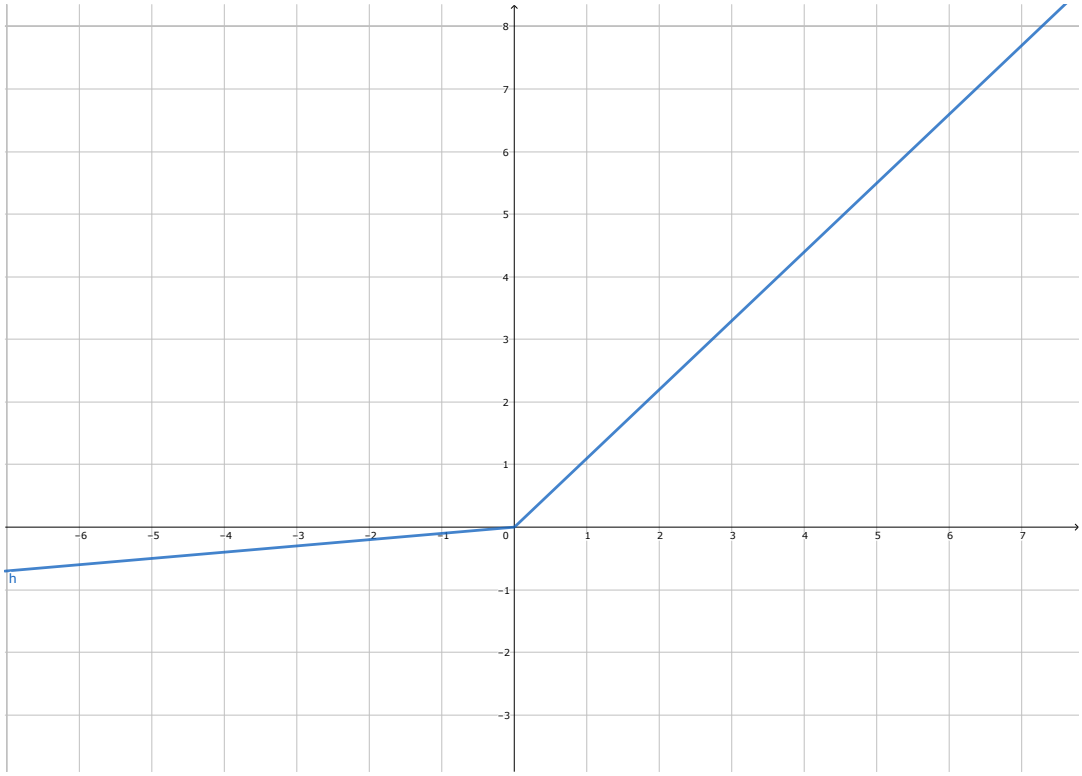

FIGURE 3.7: The Leaky ReLU Activation Function allows for a small component of negative values, in contrast to the standard ReLU.

Leaky ReLu avoids mapping negative inputs directly to zero by also allowing a nonzero gradient for negative inputs. The Leaky ReLU function avoids slow training of neural networks due to constant zero gradients (Gu et al., 2018, p. 358).

A technique known as *batch normalization* is widely appreciated as a proficient tool in neural networks along with activation functions. Batch normalization is a regularization strategy which counteracts saturated nonlinearity in training data. Essentially the technique normalizes every training batch separately, rather than normalization of the entire data set simultaneously (Ioffe and Szegedy, 2015, p. 2-4). This happens after the activation function in every layer, if applied. The details of the process are largely covered in "Batch normalization: Accelerating deep network training by reducing internal covariate shift", Ioffe and Szegedy, 2015.

3.1.3 Layers

The neurons in a neural network are interconnected and transfer information in parallel, often in layered structures, as illustrated in Figure 3.8. The neurons of a layer are connected to the neurons in both the consecutive and previous layer. In a layer with four neurons and three inputs shared for all neurons, the output (or activation units) of the layer can be described as:

$$\begin{aligned} t_0(X) &= A(x_0 w_{0,0} + x_1 w_{0,1} + x_2 w_{0,2}) \\ t_1(X) &= A(x_0 w_{1,0} + x_1 w_{1,1} + x_2 w_{1,2}) \\ t_2(X) &= A(x_0 w_{2,0} + x_1 w_{2,1} + x_2 w_{2,2}) \\ t_3(X) &= A(x_0 w_{3,0} + x_1 w_{3,1} + x_2 w_{3,2}) \end{aligned} \tag{3.8}$$

or as a collective activation for the entire layer l with n nodes and i inputs:

$$l = \begin{bmatrix} A(x_0 w_{0,0} + & \dots & + x_i w_{1,i}) \\ & \vdots & \\ A(x_0 w_{n,0} + & \dots & + x_i w_{n,i}) \end{bmatrix} \tag{3.9}$$

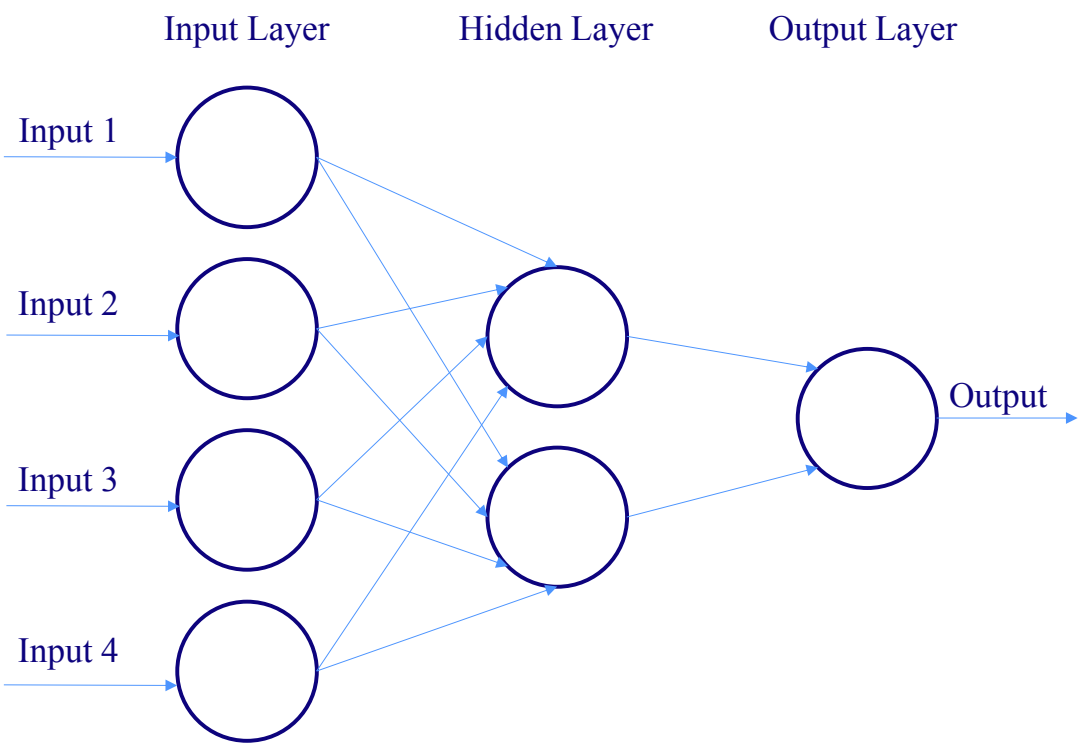

FIGURE 3.8: Several neurons structured in 3 layers, with one hidden layer between the input and output layer. Figure after O'Shea and Nash, 2015, p. 2.

As information is passed through a layer the neurons are activated to different extents depending on the input, as the neurons detect features within the input.

3.2 Training ANNs

Proper generalization in neural networks imply a networks ability to correctly classify or describe data which is unseen and similar to the data of which the network has been *trained* on. Training neural networks is based around improving the ability to generalize by training and testing the network in iterations. This section will review the important parts of training and testing, including a standard way of learning, correction techniques and metrics for assessing a networks performance.

Sample

Possible labels

{Car}

{Toyota}

{Toyota Corolla}

{Car, Toyota, Corolla}

FIGURE 3.9: A sample with an image of a car, with 4 possible label choices. A neural network can be trained to recognize both general and specific information. *Car from U.S. National Highway Traffic Safety Administration, Public domain, via Wikimedia Commons (12.01.2022)*

A neural networks weights are adapted during training such that $\hat{y}_i$ resembles y_i as closely as possible for every input sample. In Figure 3.9 an image of a car is a *training sample* with corresponding labels that a neural network can learn. In this case, the neural network learns to classify the input sample correctly, often these classifications are accompanied by a level of certainty (e.g. the image is classified as a car with a 90 percent certainty).

Sample:

Month	**Average Temperature (Celcius)**
January	3.4
February	4.2
March	5.7
April	8.2
May	12.5
June	18.3

Label: July $\rightarrow$ 22.5

TABLE 3.1: A list of temperatures in the past months can be a sample for a label containing the temperature in the following month.

In Table 3.1 the average temperatures in the first six moths of the year are

listed as the sample, and the label is the temperature in July. In this problem the neural network learns to regress towards the expected temperature.

Both of the cases are supervised learning problems. Supervised learning can be applied to both structured and unstructured data. Structured data implies that the data comes from a database where columns, rows and feature names give a well defined meaning to the data points (such as temperatures and months). In contrast, unstructured data (such as images) are not as categorized and leave more room for interpretation. In both cases, accurately labeling inputs x with proper labels y is crucial for the accuracy of the network.

3.2.1 Loss functions

Calculating the error between the predicted values $\hat{y}_i$ and the ground truth labels y_i is done using a *loss function*. The loss function determines the aim for the network in regards to what it is supposed to learn. When building neural networks, different loss functions can introduce adaptability and flexibility to a neural network architecture (Janocha and Czarnecki, 2017). This section will explain the two loss functions directed at value regression used in this thesis. Other loss functions often seen in deep neural networks are elaborated on in "On loss functions for deep neural networks [..]" by Janocha and Czarnecki, 2017.

The **L1**- or the **Least Absolute Deviations** loss function calculates the mean normalized sum of all deviations between a set of predictions $\hat{y}$ and ground truths y in a sample (with length N) as:

$$L1 = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{3.10}$$

The **L2**- or the **Mean Squared Error** loss functions evaluates a set of predictions $\hat{y}$ and ground truths y (with length N) element-wise, and calculates

the mean of the squared errors throughout the set:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.11}$$

The two loss functions are very similar, but might be useful for different applications. The L1 loss is very robust to outliers, in contrast to the L2 loss. In cases where capturing the error of outliers is important the MSE loss contributes to maximizing the value of these errors, hence adding focus to outliers. Accordingly the L1 loss function allows for reducing attention to outliers.

Loss functions are used in machine learning tasks where the goal is to minimize errors made by the network through penalizing, and are a part of a larger group of functions known as *objective functions*. The combination of loss functions, recognized as a *cost function* - represents loss in the network for different features. When referencing the objective function going forward note that it includes the interpretation of loss functions above.

3.2.2 Gradient descent

As mentioned earlier, the networks weights are adapted during training. This is done through *gradient descent*. An objective function O can be visualized as the height of a landscape with hills and slopes, and gradient descent is a method for finding the lowest point of the plane (like the plane in Figure 3.10) representing the point of where the network makes the least amount of errors. The visualization accounts for two parameters. In reality the problem is often more complex.

Gradient descent aims to find the direction of which the network should shift it's weights to further minimize errors along the steepest possible path (as seen in Figure 3.10).

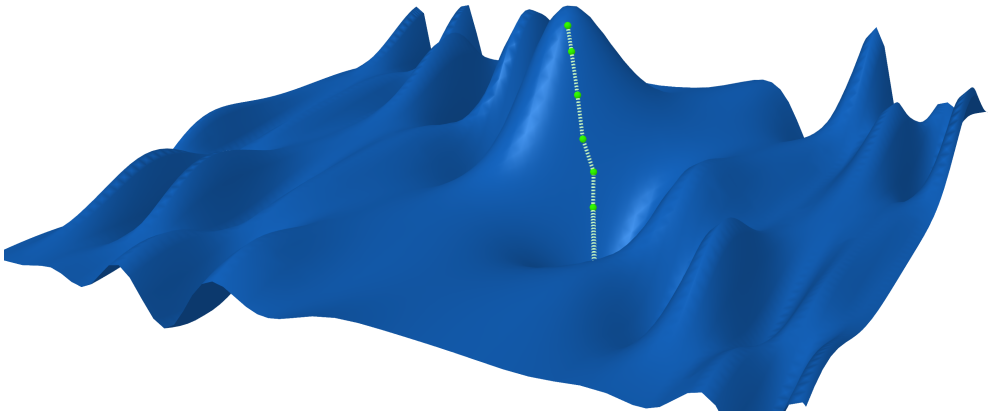

FIGURE 3.10: Gradient descent path towards a minimum. The top of the plane illustrates the starting point of which the network starts to learn, and as it gradually descends down the slope, the amount of errors made by the network decrease.

The gradient descent method allows a neural network to alter its own weights in an end-to-end differentiable system. The gradient of the objective function O is denoted $\nabla O(\mathbf{w})$ for the current set of weights $\mathbf{w}$. Stochastic gradient descent initiates a vector $\mathbf{w}_0$ which it updates for $k = 0, \ldots, N$ in steps of length α and N samples which guide the objective function to a minimum:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla O(\mathbf{w}_k) \tag{3.12}$$

Common modern *optimizers* also seek the most efficient optimization of the loss function, and have hyperparameters which allow for fine-tuning of the network. In this thesis, the two hyperparameters which have been adjusted is the *learning rate* and the *weight decay*. The learning rate determines to what extent a gradient descent iteration updates the current values of the weights. The weight decay also affects the updates of weights, by adding a term which push the weights towards zero if no other update is made directly.

The optimizer used in this thesis is called Adam, which was introduced in "Adam: A method for stochastic optimization", Kingma and Ba, 2014. Adam is reliable for problems with large amounts of data and parameters. It is proficient when working with complicated stationary objectives, such as the distribution of possible movements of precipitation in this thesis. Adam can handle both noisy and sparse gradients (Kingma and Ba, 2014).

The optimizer has hyperparameters that can increase or decrease the amplitude of updates through backward passes. This allows the network to converge more efficiently, given a sufficiently small learning rate. A network converges towards a locally optimal objective function, and at this point the gradient is zero, and updates in future iterations do not update the parameters a considerable amount on average. A network can also converge by reaching a local minimum and hence stop learning too soon, or diverge due to errors in the architecture or overfitting. Gradient descent is largely covered by Netrapalli, 2019 in "Stochastic gradient descent and its variants in machine learning".

3.2.3 Samples and Batches

A training sample is a single input data point $x \in X$ where X is the entirety of the input data of length N. A set of such samples is recognized as a batch $B_n = x_0, \ldots, x_n \subset X$ where $n < N$, as illustrated in Figure 3.11.

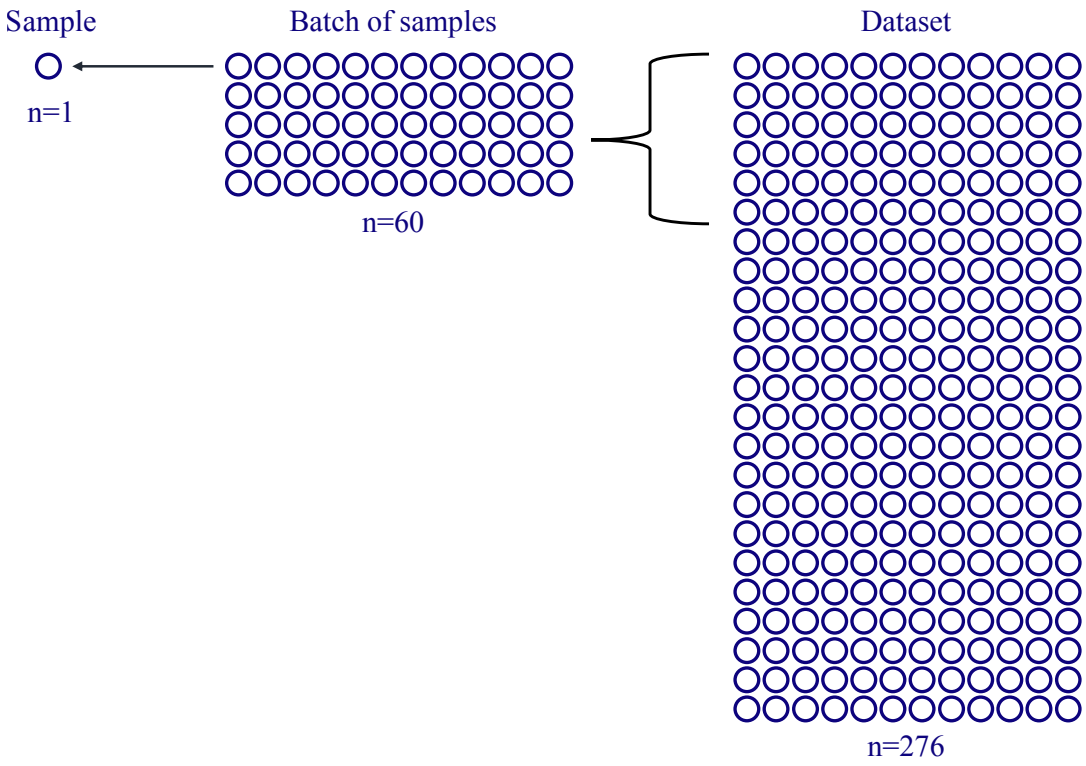

FIGURE 3.11: An example of a sample and a batch drawn from a larger dataset. The batch size is arbitrary for this illustration, but can be important in implementation of ANNs.

Stochastic, as in the explanation above, refers to splitting the data of which a neural network trains with into several smaller chunks, known as *samples* and *batches*. Splitting data into samples and batches is important for training, as the network can calculate loss for either every sample, or a batch of samples.

3.2.4 Forward Propagation

The samples are sent into a *forward propagation* either in batches or one at a time, in succession. Forward propagation is the calculation of values of intermediate variables from the input to the output layer. For simplification purposes, the processes of a forward propagation is described with one hidden layer (as pictured in Figure 3.8).

In forward propagation (Figure 3.12) the input x is multiplied with the weights of the hidden layer $\mathbf{w}^{(1)}$ and stored as an intermediate variable which an activation function is applied to, as in Equation (3.2). This intermediate variable is passed through the hidden layer and becomes a hidden variable (which is also an intermediate variable), and is further multiplied with the weights of the output layer $\mathbf{w}^{(2)}$.

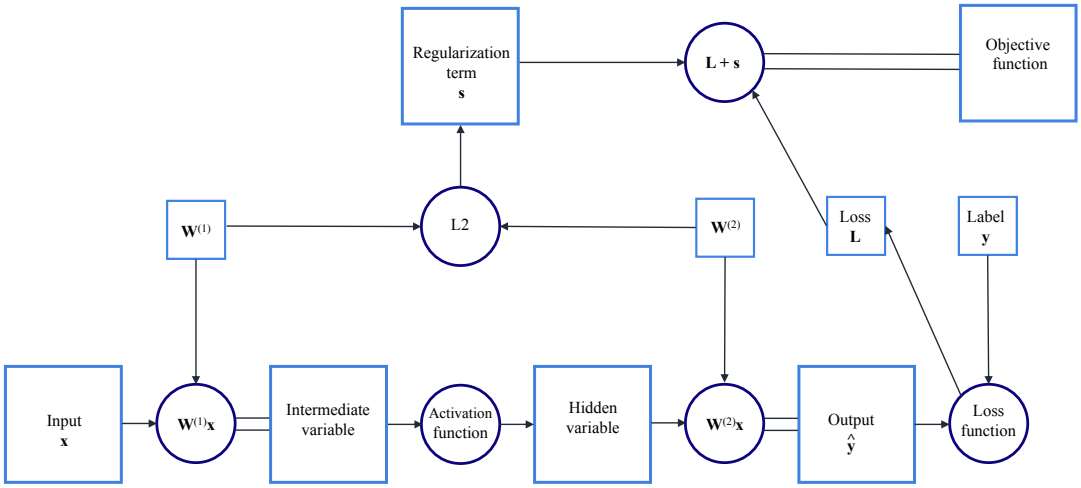

FIGURE 3.12: The process of forward propagation. The input x is along with the weights (the weighted sum / intermediate variable) is activated and passed to a hidden layer. An output is provided and a loss term calculated. This information is passed alongside a regularization term to describe the objective function.

Further, the prediction $\hat{y}$ is compared to the ground truth label y and a loss function provides a loss term, which along with a regularization term describes the objective function. The process of forward propagation through layers $l^{(1)}$ and $l^{(2)}$ (recalling the activation function A, the bias b and weight matrix W) can be described as:

$$\begin{aligned} l^{(1)} &= \hat{f}^{(1)}(\mathbf{x}) = A(b + W^{(1)}x) \\ l^{(2)} &= \hat{f}^{(2)}(l^{(1)}) = A(b + W^{(2)}l^{(1)}) \end{aligned} \tag{3.13}$$

3.2.5 Backpropagation

In order to use gradient descent the gradient of the objective function must be calculated with respect to its parameters. This is done efficiently through *backpropagation*, which makes calculations through the elements of network in reverse (i.e from output to input). In essence, this action describes the process of assessing errors made in a forward pass (Werbos, 1990). Recalling the network described in Figure 3.8 with one hidden layer, and the weight parameters $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}$ from forward propagation in Figure 3.12, backpropagation aims to compute the gradients $\frac{\partial O}{\partial \mathbf{w}^{(1)}}$ and $\frac{\partial O}{\partial \mathbf{w}^{(2)}}$, where O is the objective function, using the chain rule from calculus (Zhang et al., 2021). Backpropagation starts its calculations with the loss term, which is available as output from forward propagation. The networks performance is dependent on both propagations through the network.

3.2.6 Training loop

The forward- and backward propagations occur several times during training, once for each gradient descent iteration step. The number of propagations, more commonly referred to as *passes* through the network, are specified through *epochs*. An epoch iterates through the entire training data by loading subsets of training data in batches (as in Figure 3.11). The batches are passed forward through the network with said subsets and the network produces a prediction for every sample in the subset. Accordingly, a backward pass updates the networks parameters based on the accuracy of said predictions.

The amount of training significantly affects how the network is fitted. If a network is not trained enough, or the capacity of the network is too small, it results in an underfit. An underfit describes a function which poorly represents the nonlinearity of the input data. In contrast, overfitting is training too much, which results in a function that represents the data in the training set

too well, and unless the desired predictions perfectly match the training data will result in poor predictions as well. These fits are illustrated in Figure 3.13

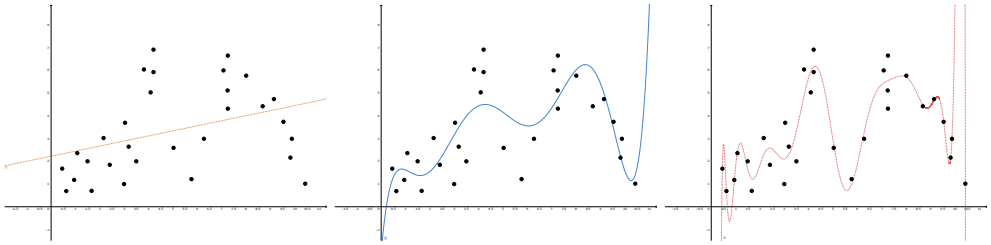

FIGURE 3.13: Three functions fitted to a set of training data points. The leftmost linear function is underfitted, and does not properly represent the nonlinearity of the data. The rightmost function fits the data too well to generalize from the training data, and is an overfit. The middle function shows a function which better represents the nonlinearity of the data, and as such can generalize to non-training data.

3.3 Convolutional Neural Networks

This section will review a type of ANN known as CNNs which are often used for processing data in a grid like structure. Examples of such grids are; time-series (1D), images (2D) or video (3D). In this description of CNNs the input is described as 2D. CNNs are used for both classification and regression problems, for example identifying cats in an image, or the number of cars on a road. CNNs do this through exploiting translational equivariance in the data and sharing weights across the layers of the network.

Equivariance means that a CNN can detect similar features in several images without a fixed position of these features. As the weights are shared for all inputs, they can be used for all outputs. This is used for detecting edges, textures and shapes. It is assumed that the data fed to a CNN share properties.

Any network that uses convolutions in one or more layers is a CNN (O'Shea and Nash, 2015). In a typical CNN there are three different types of layers; convolutional layers, pooling layers and fully connected layers.

Throughout this section, the inputs of the discussed network is to be regarded as an image, or a set of images.

3.3.1 Convolutions

The **Input Matrix** of a convolutional layer is an array. In the case of an image every grid cell of the input array has some pixel color values (Red-Green-Blue (RGB)) and/or pixel brightness value (gray-scale images in case of only brightness). The pixel color or brightness values at spatial pixel indices of the image form the depth dimension of the input array (e.g. an image of size 5×5 pixels with RGB coloring forms an input array with shape $3 \times 5 \times 5$) as shown in Figure 3.14. The amount of pixel values (such as RGB) or amount of images in a series translates to the input channels of a CNN. A convolutional layer either increase og decrease the amount of channels before the next layer. The amount of channels defines the depth of the matrix in the convolution operation. The values of the input are regionally computed as the convolutional layer calculates the scalar products between the local values of the input and the corresponding weights of the network (O'Shea and Nash, 2015).

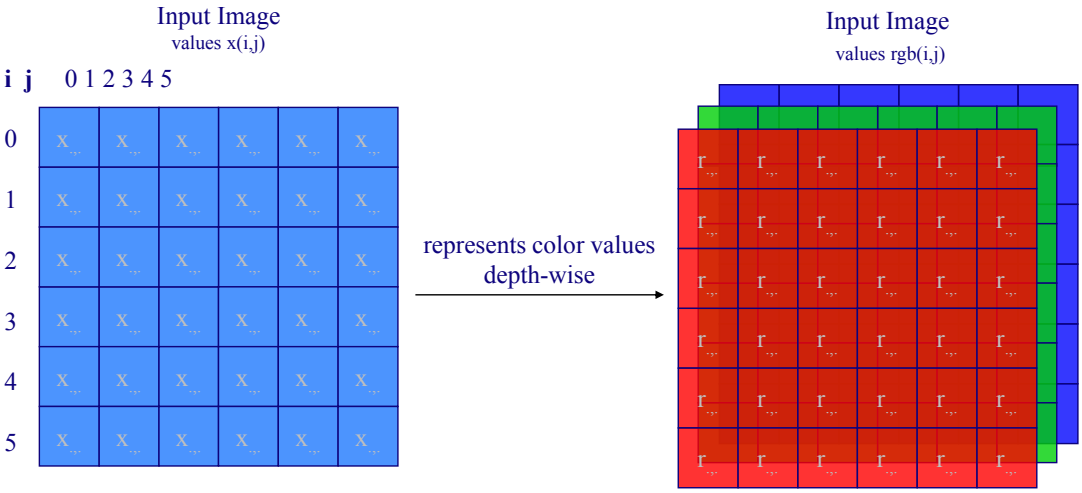

FIGURE 3.14: The values at every pixel in an input image are often RGB representations. In this case, the depth of the input array becomes 3, as every color adds depth.

The **Kernel** of a given convolutional layer is an array of learnable weights, analogous to weights described in Section 3.2.6. The kernel is smaller than the input, often 3×3, 5×5, or 7×7, but can also be larger. In 2D convolutions, the kernel spans along the entirety of the inputs depth dimension. As the data passes through the layer, the kernel produces an activation map which calculates each value seen by the kernel as in Figure 3.15. The kernel activates when a recognizable feature is seen at the current spatial region in the input array. A layer with several kernels have a differentiable output volume to its input, and the layer passes the activation maps of all kernels along the depth dimension as the output of the layer. (O'Shea and Nash, 2015)

When the input is a two dimensional array I, the kernel K is also two dimensional, with a size $m \times n$. The kernel slides across the input array, and for every position (i, j) the kernel covers a region of the input $I[i - m, j - n]$ and computes the scalar product (the weighted sum between the region covered and the kernel). The path of the kernel is restricted to positions in which the entirety of the scalar is within the input array (Bengio, Goodfellow, and Courville, 2017a).

$$s[i, j] = (I * K)[i, j] = \sum_m \sum_n I[i - m, j - n]K[m, n] \tag{3.14}$$

The kernel is considered as the defining component of a convolution. The forward and backward passes of the convolutional layer are computed by multiplying the weights of the network with the activation map "created" by the kernel.

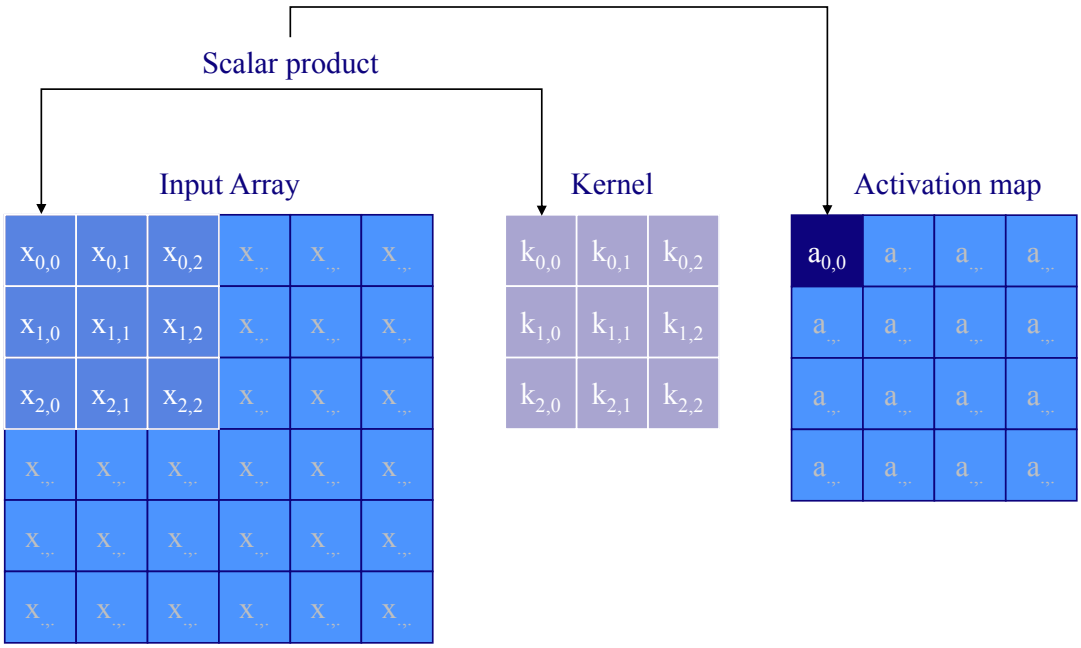

FIGURE 3.15: The kernel produces a scalar product for every region within the input array. This operation, convolutions (Equation (3.14)), essentially downscales the input array, creating an activation map of the weighted sums for every region.

The **Receptive field** for every neuron translates to the input units that affect the output of that neuron. The receptive field of a neuron is connected to the receptive fields in previous layers with their corresponding connections, as shown in Figure 3.16. As a result the computed outputs are passed through the layers.

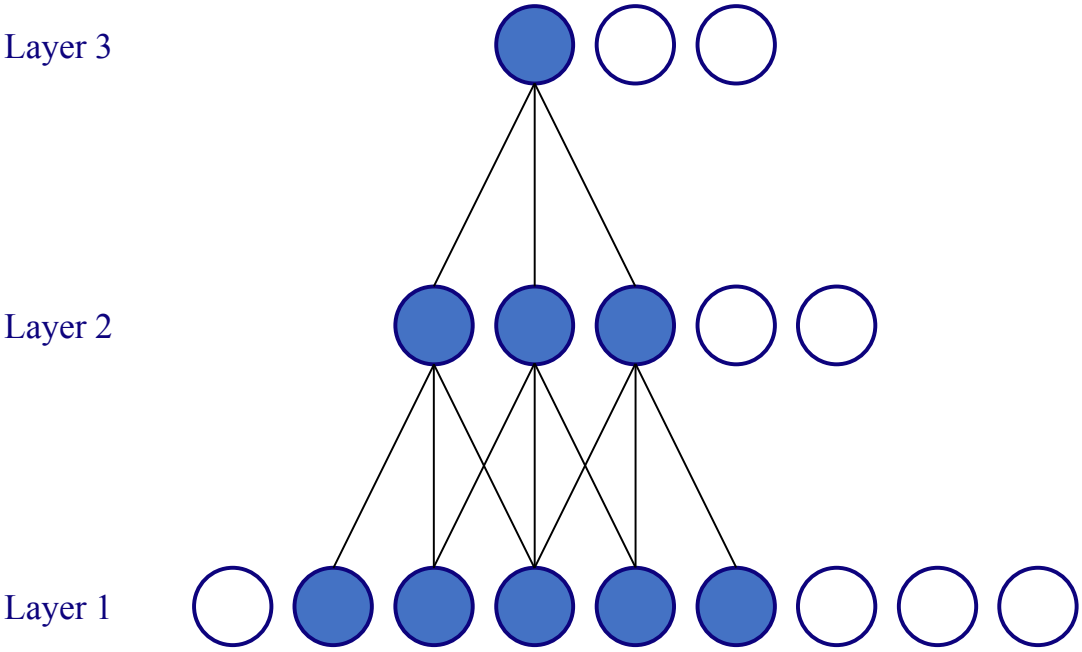

FIGURE 3.16: The receptive field of a single neuron includes the entirety of the receptive fields of connected neurons.

In convolutions the receptive field is directly connected to the kernel, and hence the weights in the network. In opposition to one layer with a large kernel, several layers with smaller kernels reduce the number of parameters required to represent the entirety of the input array. This is because the parameters are shared between layers, through the receptive fields. In Figure 3.17 every neuron of the first convolutional layer has a 3×3 view of the input vector and accordingly the neurons of the second and third layers have a 5×5 and 7×7 view of of the input vector.

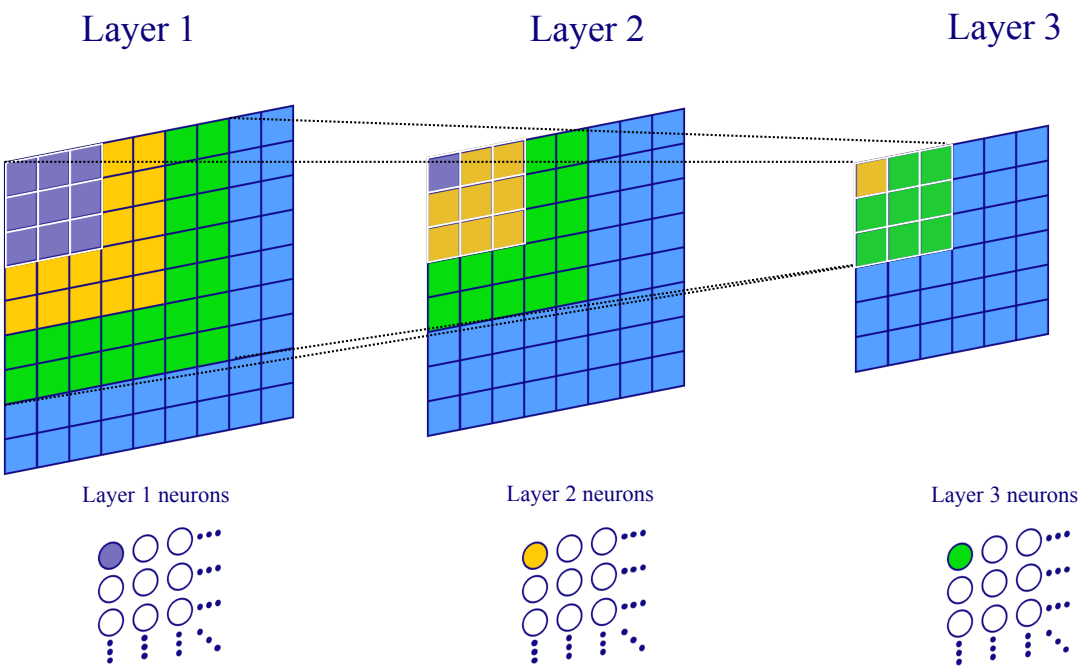

FIGURE 3.17: Receptive fields of a neurons at layers 1-3 in convolutions. Three stacked 3x3 convolutions have differing receptive fields, as the view of early layers is passed through the network as outputs.

Both **Stride** and **Dilation** are tools for further placement and enhancement of the receptive field whilst lowering the spatial dimension of the outputs in the convolutional layer. The stride of the kernel describes how many cells of the input matrix to be skipped as the kernel glides across it. A smaller stride causes overlapping of receptive fields, and a larger amount of activations, as shown in Figure 3.18. Thus, a larger stride can reduce this overlapping and reduce dimensionality (O'Shea and Nash, 2015).

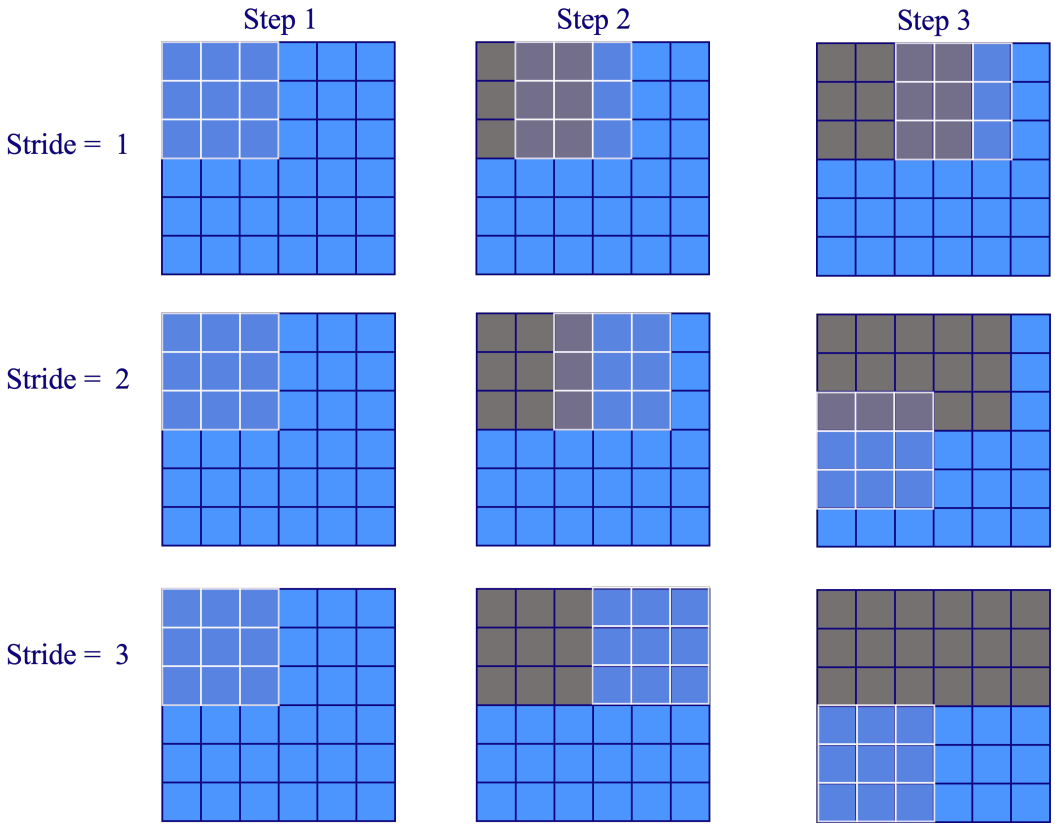

FIGURE 3.18: Path of a 3x3 kernel with strides 1-3 (first 3 steps). A larger stride covers more of the input array faster, but may miss out on some information.

Dilation refers to spacing between the cells of the kernel, as shown in Figure 3.19. Dilation increases the receptive area of the kernel whilst maintaining the amount of calculations.

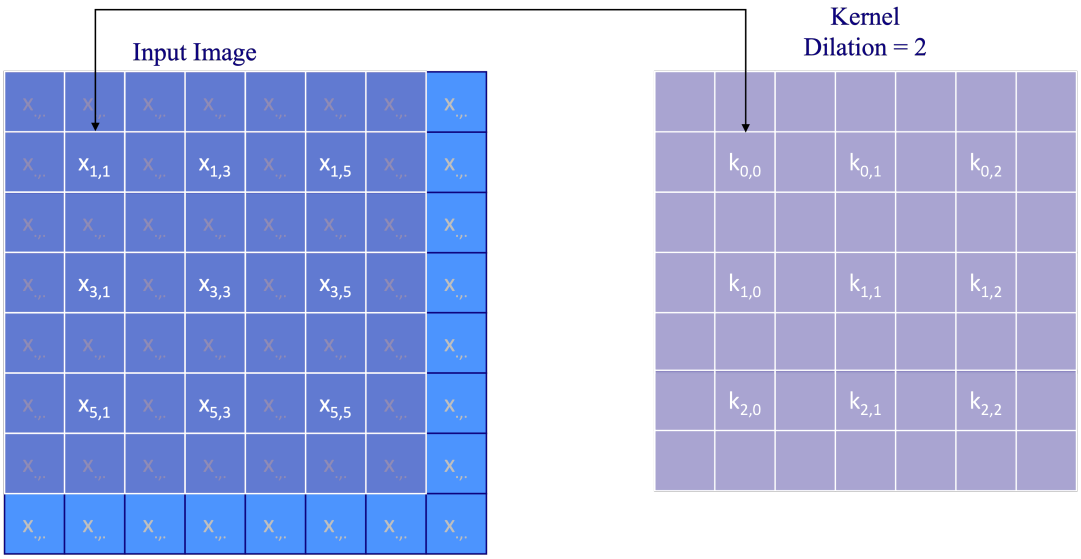

FIGURE 3.19: A Kernel with dilation of two spreads out across the input array, and processes the values highlighted in white. As a dilated kernel slides across the input array more values are processed.

Padding an input matrix along its border can control the spatial dimension of the output matrices of a convolutional layer. Padding allows for the kernel to center around the borders of the input array. Padding contradicts the restriction of the kernel staying within the borders of the input array. A correct amount of padding kan ensure alignment of receptive fields across neurons of the network, as the kernel is re-aligned with the padded-input. Several types of padding are common in modern CNNs, but in the simplest form *Zero-padding* fills a border predefined size around the input with zeros (pictured in Figure 3.20) (O'Shea and Nash, 2015).

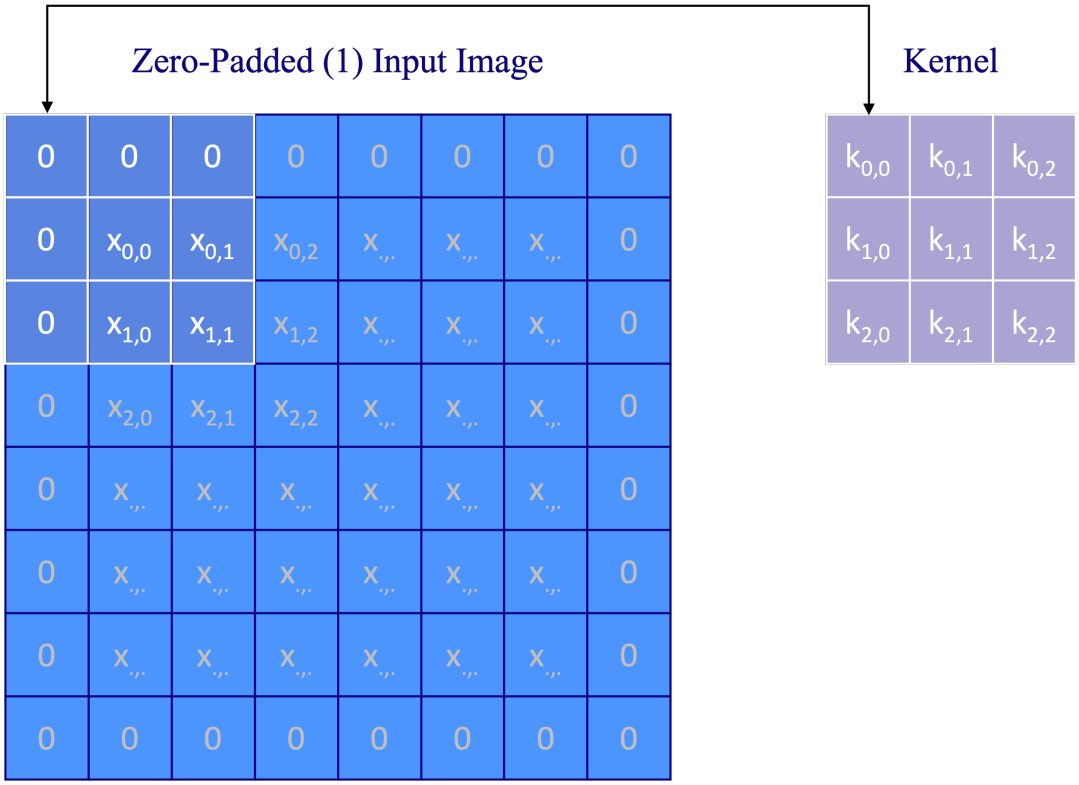

FIGURE 3.20: A Zero-Padded input image with only one pad. The zeros that create a border around the image allow the kernel to center on the edges of the original input array.

The **Output Matrix** contains features for every region which the kernel has visited whilst sliding across the input matrix (as shown in Figure 3.21). The features of the output matrix are the weighted sums of these regions in the order of when they were visited. The output of a convolution is a linear transformation of the input along it's height and width across its entire depth. The output can be deeper, equal to or shallower than the input along the depth axis. This is determined by the desired output channels of the convolution (O'Shea and Nash, 2015).

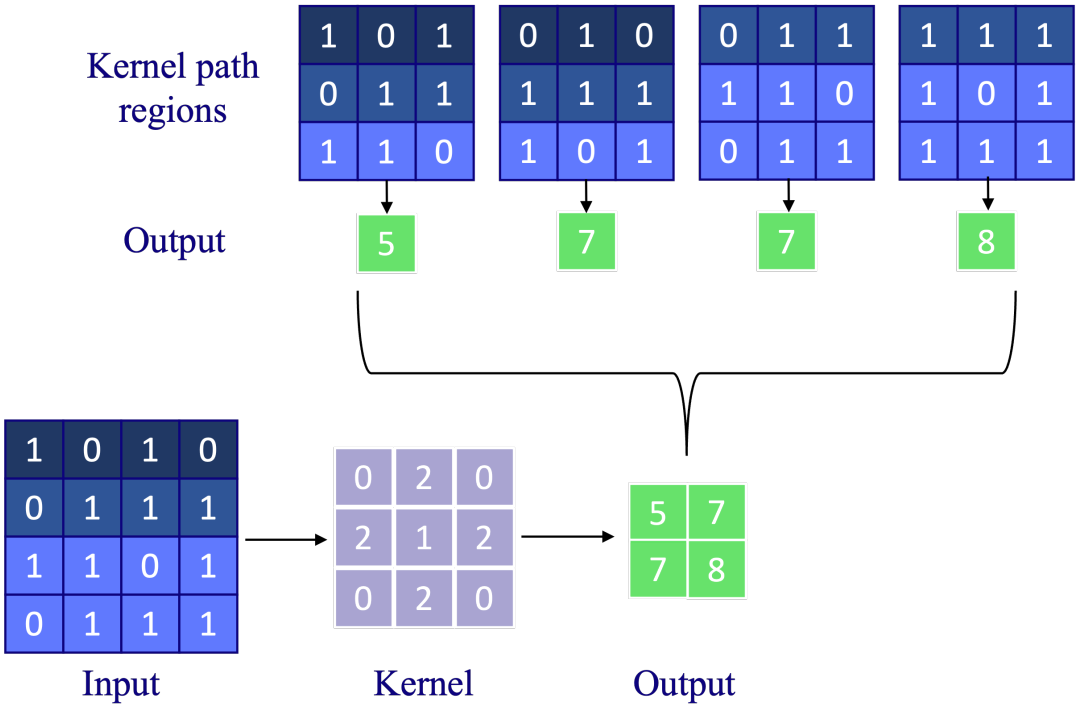

FIGURE 3.21: A 3×3 kernel (with stride 1) passes over an 4×4 input array, and generate an output for all 4 regions combined. The convolution operation in essence downscales the input, whilst retaining information from every region.

3.3.2 Pooling layer

The pooling layer of a CNN performs a down-sampling of the input along the spatial dimension. The operation of pooling reduces the number of parameters of the model, whilst also reducing computational cost and complexity. Pooling layers function by sliding (often) small filters along the spatial dimensions of the input. The amount of calculated regions within an input is decided by the size of the filter, namely the pooling operator. Pooling layers maintain the depth of the input, as it uses some function (operator) for down-scaling the spatial dimension. Most common are the MAX or AVERAGE functions, which do the following:

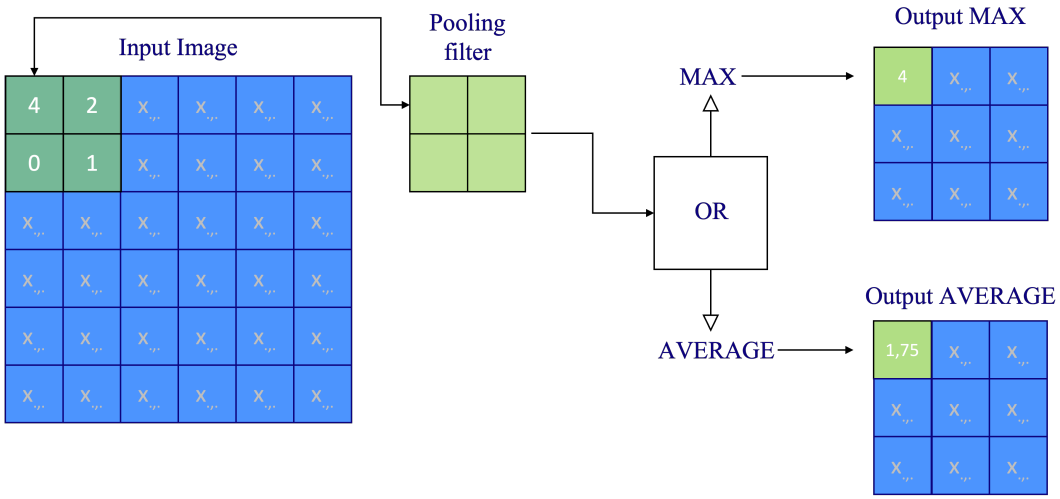

FIGURE 3.22: Pooling Operators AVERAGE and MAX are both commonly used in CNNs. The figure illustrates that the pooling filter can be either of the two. This is defined during implementation.

The **Average Pooling** operator evaluates a region within the input and returns the average value for that region.

The **Max-pooling** operator evaluates a region within the input and returns the maximum value for that region. There are many variations of pooling layers which are fully explained in "Pooling methods in deep neural networks, a review" by Gholamalinezhad and Khosravi, 2020.

3.3.3 Fully-Connected Layer

In CNNs a fully-connected layer is commonly used to directly connect neurons of convolutional- or pooling layers to the output layer (O'Shea and Nash, 2015). These layers are often tools for transferring information from the weight matrices processed through the network and to some posterior meaning or interpretation of the input (e.g. the likelihood of an image containing a cup rather than a dog), or a measurement in the form of a regression (e.g. the age of a tree based on an image of the stud).

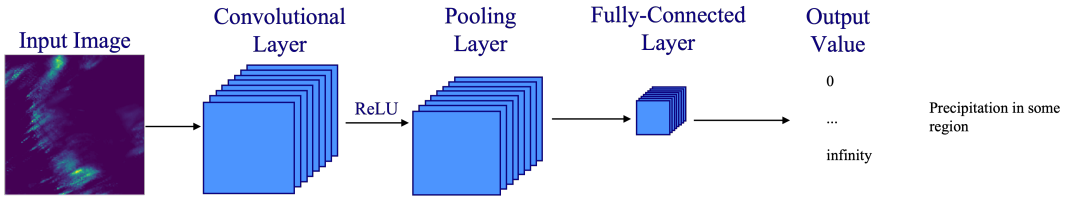

FIGURE 3.23: A simple regression CNN for precipitation in a region from radar image. The final layer before the output is a fully-connected layer which can translate images to values.

3.3.4 Transpose convolutions

Transpose convolutions, also recognized as *fractionally strided convolutions* or *deconvolutions*, are generally desired for transforming information in the opposite direction to that of regular convolutions. In essence, the process of transpose convolutions up-scale some input while maintaining connectivity, rather than down-scaling it. The input fed to transpose convolutions is often the output of regular convolutions. Operations like these are often seen in segmentation problems, where some amount of down-scaling is beneficial as an encoding tool, and up-scaling of the down-scaled output acts as a decoder. Often seen in segmentation and classification problems, this process is efficient, as it can project dense information to a higher-dimensional space after having reduced that information to an abstract representation (Dumoulin and Visin, 2016).

As discussed in the previous section on Convolutional Neural Networks, the kernel is the defining component of a regular convolution operation, which is also the case for transpose convolutions. The difference between the two methods lies most prominently in the forward and backward passes of the network (Dumoulin and Visin, 2016). Transpose convolutions are covered in "A guide to convolution arithmetic for deep learning" by Dumoulin and Visin, 2016.

3.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are applied step-wise and repeatedly to sequential data. In addition to the functionalities of a Feed-Forward Neural Network (FFNN), for every step taken some information is retained and passed back to the network in the next step. This information is kept in a *state vector* in a *hidden unit* (Bengio, Goodfellow, and Courville, 2017b, p. 333-336). This vector contains information about the history of the sequence the network has already processed, and is passed through the steps as shown in Figure 3.24.

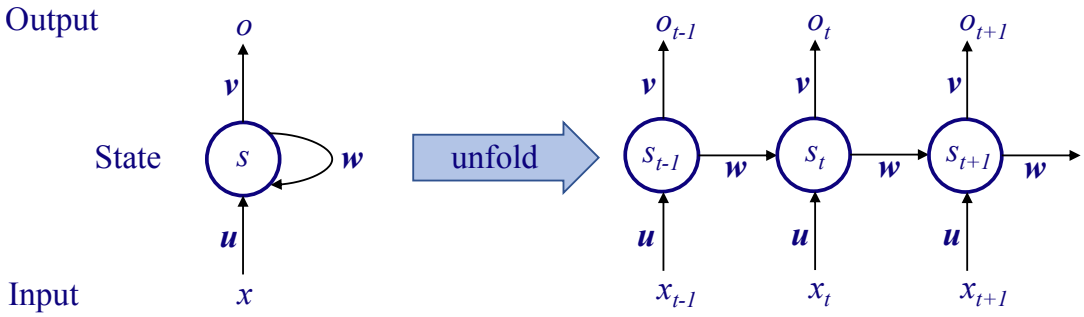

FIGURE 3.24: The RNN has a loop which returns the state in the hidden layer. The input x is passed through this layer and forms an output o. When *unfolded* this structure is a sequence of this operation where the hidden unit in the state vector is passed to the consecutive step at time t. $\boldsymbol{w}$ is the weight matrix for the hidden-hidden connection. $\boldsymbol{u}$ and $\boldsymbol{v}$ are the corresponding weigh matrices for the input-hidden and hidden-output, respectively. Figure after Bengio, Goodfellow, and Courville, 2017b, p. 333.

The outputs of hidden units (as shown in Figure 3.24) at different discrete time steps can also be considered analogous to the outputs of neurons in a multi-layered ANN as described in Section 3.1.1.

Training RNNs have been considered problematic, as gradients in back-propagation for RNNs either grow larger or shrink at every time-step (Bengio, Goodfellow, and Courville, 2017b, p. 334-338).

3.4.1 Long Short-Term Memory

The Long Short-Term Memory (LSTM) is an RNN architecture that was introduced by Hochreiter and Schmidhuber as early as 1997. LSTMs are special sequential ANN that can learn long-term dependencies through keeping long- and short-term memories. This section is largely based on the explanations in Hochreiter and Schmidhuber, 1997 and Bengio, Goodfellow, and Courville, 2017b, p. 360-361.

Figure 3.25 shows an LSTM *(memory) cell*. A cell has three *gates*, the *input*, *forget* and *output* gates. The gates have explicit tasks. The input gate tries to learn new information from the input x_t at time t. The forget gate evaluates the short-term memory in the hidden state h_{t-1}, and the long-term memory of the *cell state* c_{t-1} of the previous time step $t-1$. Finally the output gate passes updated information of the short- and long-term memories h_t and c_t, and passes h_t as its output (Hochreiter and Schmidhuber, 1997).

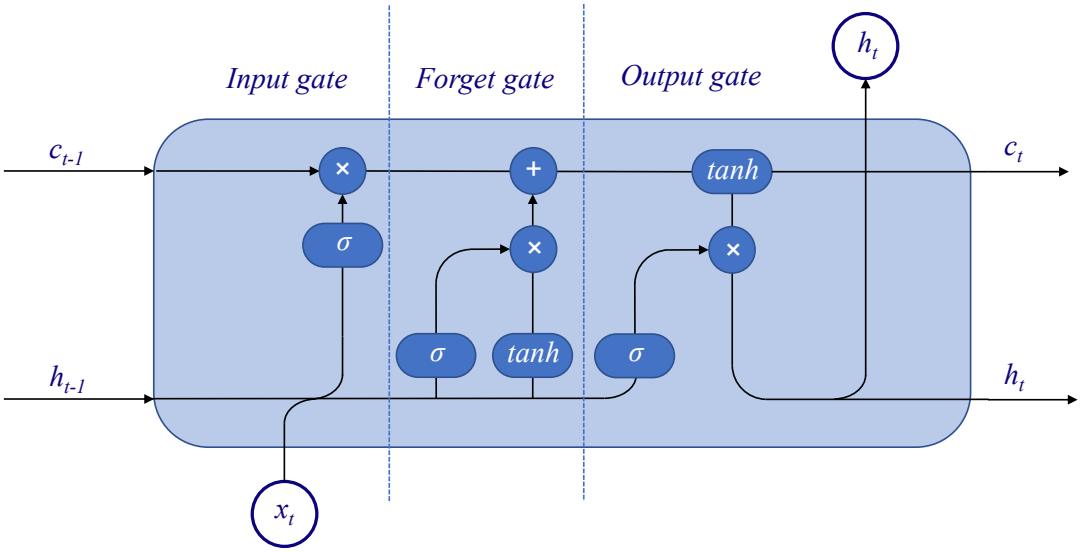

FIGURE 3.25: An LSTM Cell processes data sequentially whilst keeping its hidden state. Each line carries a vector: the long-term memory c_{t-1}, short-term memory h_{t-1} or the input x_t at for time t. The circular markers are point-wise operations (multiplication or sum), whilst the elongated markers are activations (sigmoid or tanh).

The bottom left line in Figure 3.25 shows operations made on the input

x_t and the hidden state of the previous step h_{t-1}. This information is passed upwards to top right line which shows the operations on the cell state (long-term memory). As it passes upwards it goes through the input gate G_i. It uses sigmoid activation on the networks weights for the input gate (a concatenation of the weights for the previous hidden state and the current input) $W_i[h_{t-1}, x_t]$ and bias at time t.

$$G_i = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{3.15}$$

This input gate decides which values to update. Along with this information comes the forget gate G_f, which is the point wise multiplication of the same activation of the input gate G_i and a *tanh* (hyperbolic tangent) activation of the same values.

$$G_f = \sigma(W_f[h_{t-1}, x_t] + b_f) \times \tanh(W_f[h_{t-1}, x_t] + b_f) \tag{3.16}$$

The previous cell state c_{t-1} is point-wise multiplied with the result of the input gate and then the weighted sum of the forget gate.

$$c_t = c_{t-1} \times \sigma(W_i[h_{t-1}, x_t] + b_i) + \sigma(W_f[h_{t-1}, x_t] + b_f) \times \tanh(W_f[h_{t-1}, x_t] + b_f) \tag{3.17}$$

Finally the output gate G_o uses a sigmoid activation similar to the input gate and a tanh on the cell state prior to a point-wise multiplication of the two, resulting in the new hidden state.

$$G_o = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.18}$$
$$h_t = G_f + \tanh(c_t)$$

An unrolled LSTM layer (as shown in Figure 3.26) has one or more of these cells in parallel.

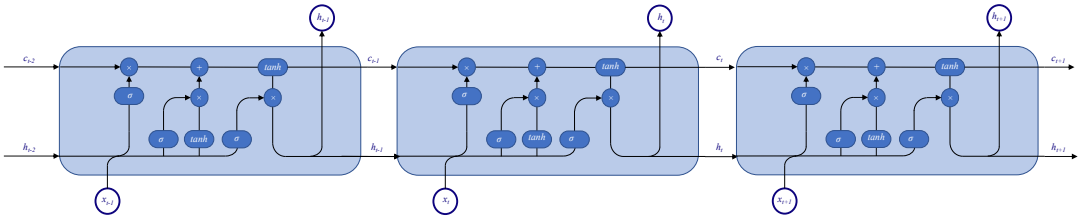

FIGURE 3.26: A chain of three LSTM Cells that passes memories sequentially.

The LSTM archetype has since then proven useful for learning patterns within distributed, real-valued and noisy data (Hochreiter and Schmidhuber, 1997). An LSTM **learns** through backpropagation through time (BPTT), which is the backpropagation discussed in Section 3.2.2 on the unfolded RNN in Section 3.4, Figure 3.24. The errors recognized through BPTT arrive at memory cell net inputs and are scaled by the output gates activations before being propagated back through previous internal states (points of memory).

Chapter 4

Working with meteorological imagery and hydropower data

The two main sources of data used in this thesis are The Norwegian Meteorological Institute and Saudefaldene. The data collected from The Norwegian Meteorological Institute are radar images, which are used for predicting precipitation. The predicted precipitation is important for predicting the inflow to Helgedalsvatnet, which Saudefaldene provides data about. The data consists of several separate time series, including the water level, recorded precipitation and the valve opening. Other data may be important and available, but to limit the complexity of the study only the above mentioned data is included.

4.1 Radar images collected through THREDDS

The THREDDS Data Service (TDS) is a web server access point for scientific data. It is used by several institutions, including The Norwegian Meteorological Institute (MET). The service provided by MET, MET Norway Thredds Service, allows users to access scientific datasets through several remote access protocols.

The data obtainable from MET Norway Thredds Service can be downloaded with a Network Common Data Form (NetCDF). These data forms are

self-describing, portable, scalable, appendable, sharable and archivable. A tool in the MET Norway Thredds Service allows for geographical grid subsetting through a NetCDF. This is important for the thesis, as the main interest is the western part of Norway, more specifically Sauda, and the area around Helgedalsvatnet. In this thesis, the data collected by this service is a set of radar images. A sequence of these radar images contain the precipitation rate of liquid water equivalents per hour. Figure 4.1 shows 8 images from the dataset that are ordered in a sequential and ordinal manner.

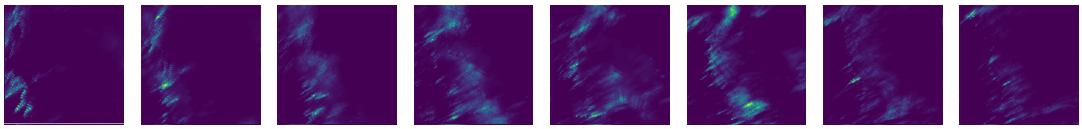

FIGURE 4.1: A sequence of 8 radar images one hour apart with varying rates of precipitation covering the area in Figure 4.2. A brighter color means more precipitation reflectivity in the image.

In order to capture where the precipitation originates from and its current course, a large enough area is needed to capture any precipitation that might pass over Helgedalsvatnet some hours prior to downfall (as depicted in Figure 4.2). The water level is affected by the downfall in Helgedalsvatnets precipitation field adjusted with the delay from downfall to inflow.

FIGURE 4.2: The entire area covered by the radar images is illustrated with the large blue bounding box, which stretches from Bergen in the north to Stavanger in the south. Map created with https://nedlasting.nve.no/gis/.

The coordinates fed to the MET Norway Thredds Service provide coverage of the geographical area in Figure 4.2, which has a center adjacent to the target area in Figure 4.3. The center of the images is $59°41'17.1''N, 6°25'35.6''E$, which is close to Helgedalsvatnet. The area received is also predefined by reach, i.e how much area around the center is desired. The subset (hereby *radar dataset*) consists of 200 by 200 (x, y) coordinates, where the images have a 1km resolution per pixel.

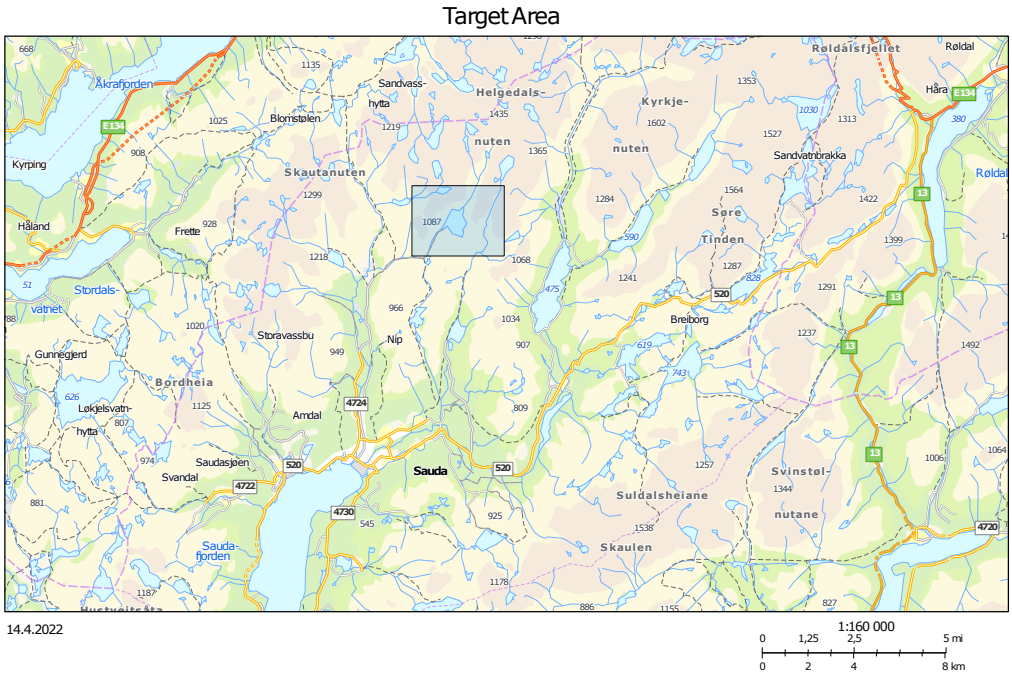

FIGURE 4.3: The target area close to center of the area (in Figure 4.2) covered by the radar images is shown with the blue bounding box, covering Helgedalsvatnet. This box covers some of the precipitation field for Helgedalsvatnet, but is only an illustration. Map created with https://nedlasting.nve.no/gis/.

The area which is most important for predicting precipitation is the precipitation field of Helgedalsvatnet, illustrated in Figure 4.3.

Every radar image has a timestamp for when the image was captured. The radar images contain captured liquid water or equivalent precipitation per (x, y) pair of coordinates at the given time. The radar images are structured at one hour intervals covering the time frame *01/01/2014 - 25/08/2021*, for a total of 67,015 images. The hourly captured images are long enough apart, time-wise, to capture the movement of recorded precipitation (weather) whilst not loosing too much information between hours. Using radar images captured at longer intervals might have prompted difficulties in future prediction as not enough information is available to capture the natural movement of clouds containing precipitation.

As rain clouds are not uniformly distributed information per image may be sparse, and many of the images do not contain any detected precipitation. The amount of detected precipitation can also vary, which introduces some uncertainty. A radar image which is considered sparse contains little to no precipitation (i.e. values ranging 0.0-1.0 or smaller) or have some small areas with dense information (as in Figure 4.4).

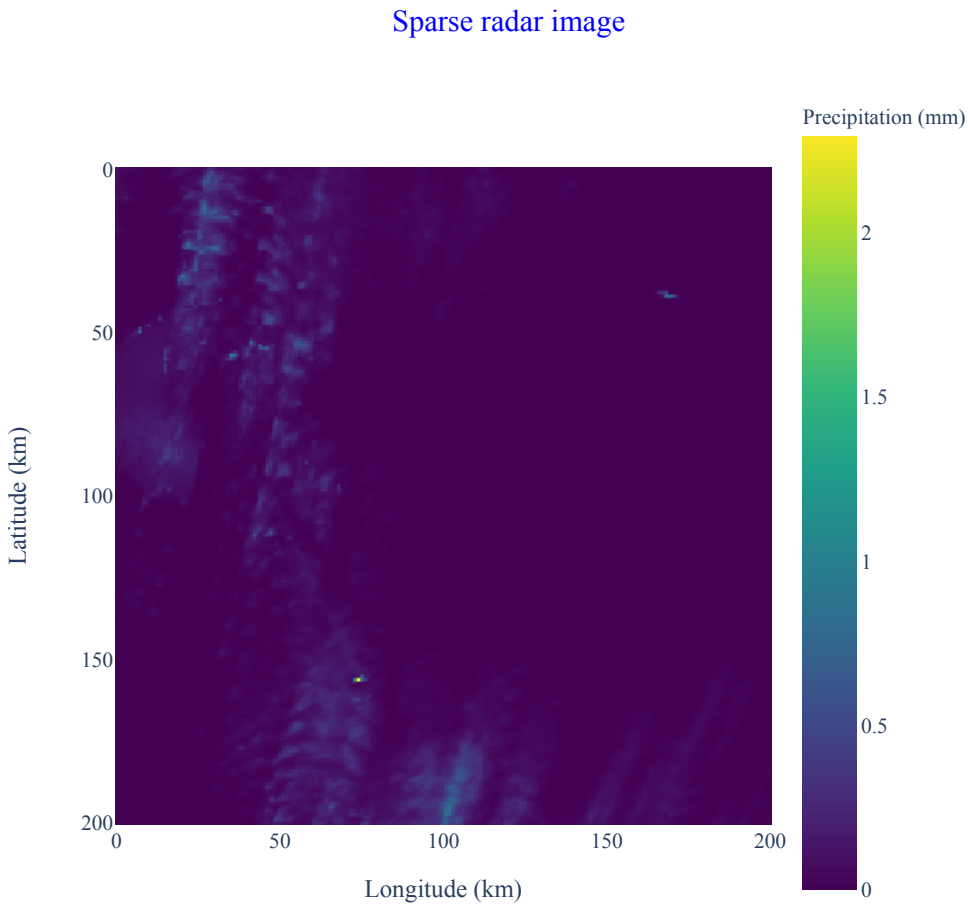

FIGURE 4.4: A sparse radar image of detected precipitation covering the area in Figure 4.2

A radar image which is considered dense has a lot of recorded precipitation. In a dense image precipitation can either cover the image rather evenly or have parts of the image with dense precipitation (as in Figure 4.5).

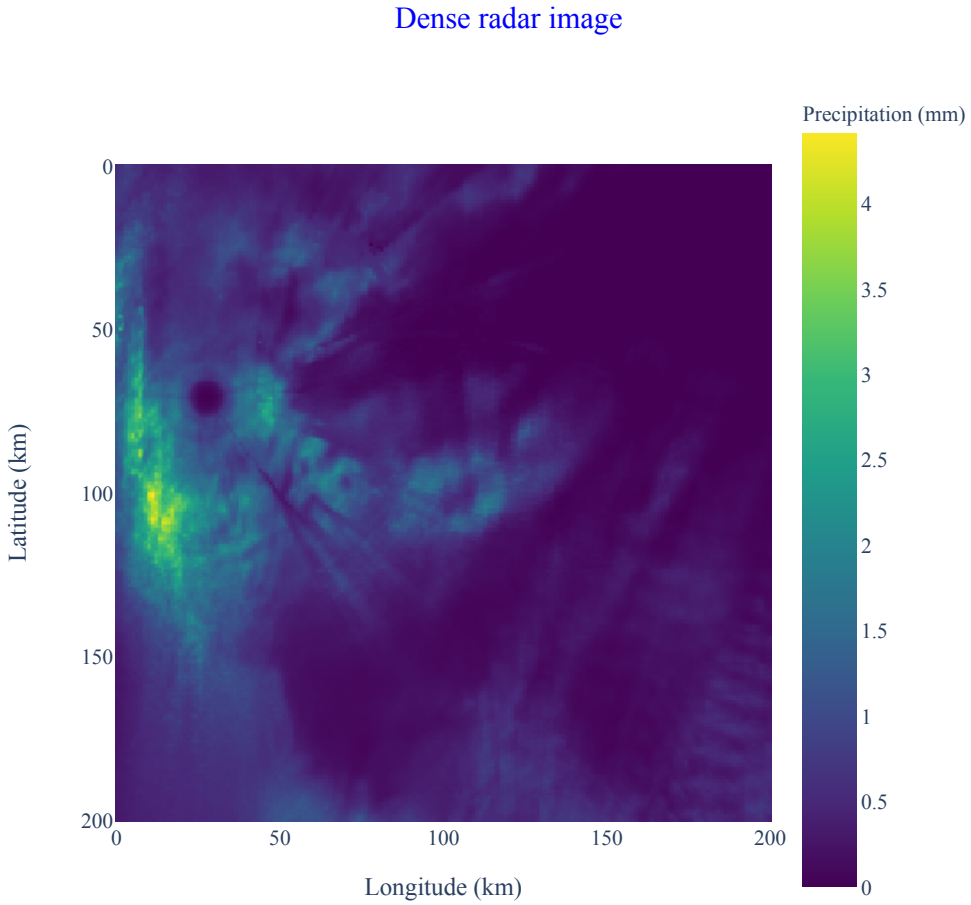

FIGURE 4.5: A dense radar image of detected precipitation covering the area in Figure 4.2

4.2 Data provided by Saudefaldene

4.2.1 Water reservoir level

Helgedalsvatnet is located at an higher altitude than the other reservoirs of Saudefaldene. Helgedalsvatnet is therefore not affected by the other reservoirs, and primarily affected by precipitation, temperature, water retention in the local area and outflow from the reservoir. The outflow is a controllable variable by the hydropower operators. This simplifies the task of predicting

a change in relative water level. The measurements are made on a daily basis, covering the time frame *01/01/2014 - 25/08/2021*, and describe the change in relative water level from some standard level. This standard water level is the Highest Regulated Water Level (HRW), and can be directly translated to the volume in the reservoir in million m^3. When the value is at it's lowest, the water level is at the Lowest Regulated Water Level, and ascends with an increase in water level.

The water level of this reservoir is the target value for the model presented in this thesis

Date of measurement	**Change in relative water level (HRW-m)**
04/08/2021	-4.74
05/08/2021	-4.94
06/08/2021	-5.14
07/08/2021	-5.34

TABLE 4.1: Examples of change in relative water level to a predefined zero (Helgedalsvatnet 04/08/2021 - 07/08/2021)

A snippet of the time series in Table 4.1 shows typical data entries. Note that not all data entries in the series are measured, as some entries are stipulated values (by Saudefaldene). The daily intervals of these data entries are transformed to hourly intervals to match the time series of the radar images from Section 4.1. This is done through linear interpolation of the measurements.

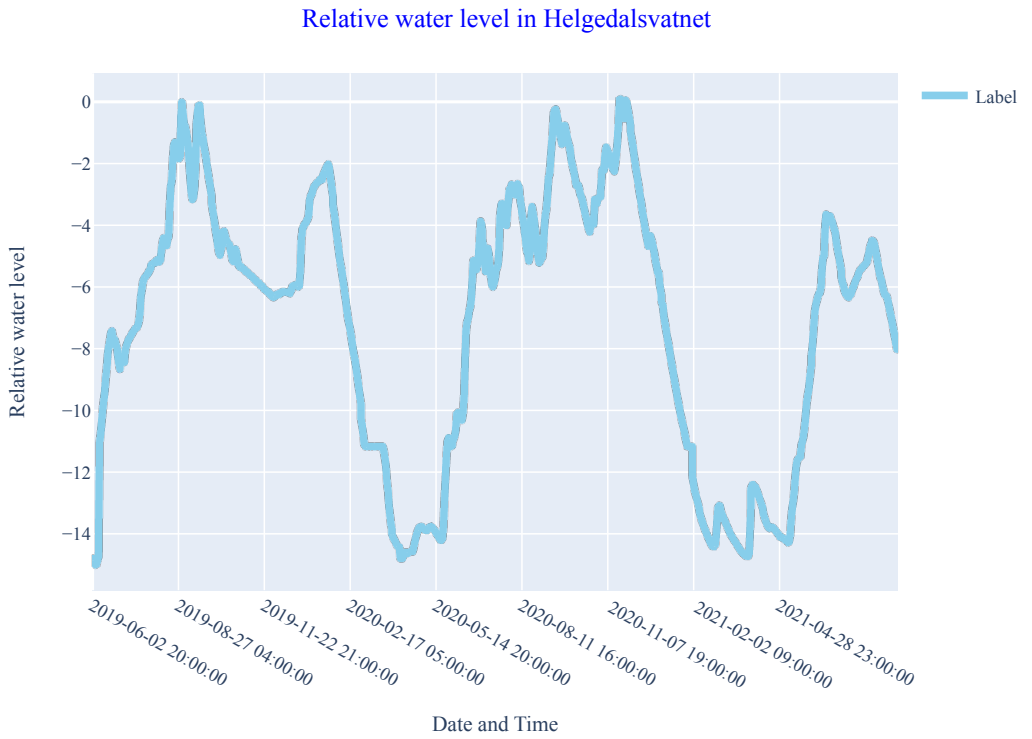

FIGURE 4.6: The relative water level in the test set of the data.

4.2.2 Precipitation

The precipitation time series delivered by Saudefaldene contains data covering the time frame *01/01/2014 - 26/08/2021*. The data entries are at 07^h and 19^h, with 12 hour intervals. The data in this time series are slightly different but consistent with the measurements that can be gathered from SeNorge.no, which is a reliable online tool for themed maps of Norway's meteorological and climactic data. The discrepancy between the datasets of precipitation is likely due to the location of the weather station, the measurement methodology and local variations in geographical topology. These minor differences are accepted as all data except the radar images discussed in Section 4.1 are sourced from Saudefaldene. The main issue with the precipitation data is the recording intervals. Similar to the water level data discussed in Section 4.2.1, the missing data points per hour has to be addressed, which will be explained in Section 4.3.

Date and time of measurement	**Precipitation (SAUDA) (mm)**
26/07/2021 07:00	0.0
26/07/2021 19:00	0.1
27/07/2021 07:00	4.5
27/07/2021 19:00	5.9
28/07/2021 07:00	7.4
28/07/2021 19:00	8.8
29/07/2021 07:00	2.6
29/07/2021 19:00	0.1

TABLE 4.2: Examples of measured precipitation at 12h intervals (SAUDA 26/07/2021 - 29/07/2021)

4.2.3 Valve opening

The data points describe the opening of the valve connected to Helgedalsvatnet. These datapoints show the valve opening in centimeters for a period of time. The values are sectioned into separate date ranges, which provides a valve opening which lasts through the entirety of the respective period.

The valve opening describes the amount of water being drawn out of the reservoir. Although the valve opening is not corresponding directly to the exact amount of outgoing water, the indication of small or large change is apparent in the data. It is reasonable to expect that the outflow changes when the valve opening changes, and therefore the assumption is made that there is a strong and consistent relationship between the valve opening and a change in the water level.

Start Date	**End Date**	**Valve Opening (cm)**
15/06/2021	10/07/2021	15
10/07/2021	02/08/2021	0
02/08/2021	04/08/2021	5
04/08/2021	01/01/2030	10

TABLE 4.3: Examples of valve opening Helgedalsvatnet (15/06/2021 - 04/08/2021)

4.3 Missing data

As the data described in Section 4.2 are structured at varying time intervals it is imperative to reorganize the separate data into a cohesive and properly structured time series. The time series is defined in one hour intervals due to the structure of the radar images. This section reviews and explains the use of two methods for handling missing data, namely linear interpolation and backwards filling. In the linear interpolation method there are some assumptions made about the natural world which can be considered reasonable as the changes in water level change smoothly and continuously, and sudden bursts of precipitation still have an accumulative delay over the geographical area which can be fit linearly within hourly ranges. Remember that these assumptions would not have been required if the data was complete with actual and cohesively recorded data.

4.3.1 Linear Interpolation

When employing linear interpolation methods the assumption is made that the variations in the quantities of the variables included are smooth and continuous. As the linear interpolations fill short time periods, such as a day or some hours, the implication that the quantities vary slowly and at the time

scale of the date entries is introduced. For example, the idea of precipitation linearly decreasing or increasing might be ambitious. Still, the amount of precipitation measured at the start of a 12 hour interval being larger than what is measured at the end of the interval describes a decline in precipitation which can be linearly represented. This representation, although not exact, describes a trend which can correlate to the target value, which is the water level of a reservoir. This target value is also linearly interpolated with similar assumptions. The water level of a given reservoir is, as discussed in Section 4.2.1, dependant on the outgoing amount of water and water inflow through water retention in the surrounding area, precipitation and possibly temperature. In regards to both precipitation and the changes of water level between two data-points with respective timestamps, the linear interpolation provides an evenly spread distribution of relative values between the data-points at hourly intervals rather than (sub-)daily.

4.3.2 Backwards Filling

The valve opening time series is adjusted backwards to align the data with the hourly structure of the radar images. It is a simple process which records gaps in the dataset, and recursively fills data within the gaps with the element recorded at the end of the gap, as shown in Table 4.4. The gaps in the dataset are due to the recording method which Saudefaldene has chosen. Since the valve openings are kept through longer periods of time it does not impose any assumptions, and only require the structural change.

Date	**Valve Opening (cm)**	**Valve Opening (cm, backwards filling)**
04/08/2021 04:00	-	5
04/08/2021 05:00	-	5
04/08/2021 06:00	-	5
04/08/2021 07:00	5	5
04/08/2021 08:00	-	10
04/08/2021 09:00	-	10
...	...	...
26/08/2021 07:00	10	10

TABLE 4.4: Example of valve opening time series before and after backwards filling. The values prior to 26/08/2021 *07:00* are filled with the end value (10) until the next value (5) is reached. The values prior to 04/08/2021 *07:00* then are filled with the end value (5) until a new value is met, etc.

4.4 The final dataset structure

The final structure of the dataset includes the timestamp, measured precipitation, valve opening, recorded water level and the corresponding radar image for that timestamp.

Timestamp (Date and time)	**Precipitation** (mm)	**Valve Opening** (cm)	**Water Level** (HRW-m)	**Radar Image** (depth, x, y)
28/07/2021 13:00	8.10	0	-4.8250	(1, 200, 200)
28/07/2021 14:00	8.21	0	-4.8175	(1, 200, 200)
28/07/2021 15:00	8.33	0	-4.8100	(1, 200, 200)
...	...	...	...	...
07/08/2021 20:00	0.04	10	-5.4429	(1, 200, 200)
07/08/2021 21:00	0.08	10	-5.4508	(1, 200, 200)

TABLE 4.5: A small section of the dataset shows the arrangement of the values.

The dataset does not necessarily directly represent the true dynamics of the natural inflow of water to the hydropower reservoir, as linear interpolation and backwards filling is used for missing data. Still, the dataset includes the information required to prove the concept of predicting the water level in a reservoir. Chapter 9 discusses how future work can address of this problem and provide better opportunities for accurate representation of these dynamics.

Errors in the data are likely, along with stipulations made by Saudefaldene. Whilst not creating issues for predictions, the errors are mentioned as not all measurements are perfect, and differences between manual and automatic measurements are common. The assumption that the relationships between the values that are included is stronger than the relevance of such minor errors is made, and consequently that the required data correctly represents the actual inflow and outflow as closely to the true data as possible, given a perfect dataset with the correct structure and no missing values.

Chapter 5

Related Work

This chapter will review some related works that use similar methods to the proposed method in this thesis. The terms *Forecasting* and *Nowcasting* are used in neural networks aimed at weather, and several efficient methods have been proposed. This thesis relates to methods that use satellite or radar images for precipitation prediction. There are few models that use such precipitation forecasts for predicting hydropower reservoir levels, which is an added problem specific to this thesis (in contrast to some of the related works). Forecasting weather, specifically precipitation, is a mature field in modern data science, statistics and meteorology. The research field contains more related works than what is included in this chapter. The chosen works are very relevant to this thesis. Note that if all the models presented in these works were targeting water reservoir levels, a fair comparison could be made to the model presented in this thesis. The models are reviewed, but the results are not used for comparison.

5.1 Skillful Twelve Hour Precipitation Forecasts using Large Context Neural Networks

"For neural models [...] each additional hour of lead time poses a substantial challenge as it requires capturing ever larger spatial

> contexts and increases the uncertainty of the prediction." (Espeholt et al., 2021, p. 1)

In August of 2021, a coalition of *Google Research* teams consisting of the Brain-, AI for Weather- and Kernel teams presented a neural network model capable of forecasting precipitation on a large scale with greater accuracy than that of top of the line physics-based models, up to 12 hours ahead. It was proposed that neural networks for weather prediction can combat the computational bottleneck of physics-based models such as NWP models and the more encompassing ensemble NWP models (Espeholt et al., 2021).

Based on this proposition the team presented MetNet-2, a successor to the proposed neural network in "MetNet: A Neural Weather Model for Precipitation Forecasting" by the same research team. MetNet-2 is trained to forecast precipitation across the Continental United States, an area of around 7000km $\times$ 2500km, with a resolution of 1km $\times$ 1km. The proposed model outperformed the currently used NWP models High-Resolution Rapid Refresh (HRRR) and High-Resolution Ensemble Forecast (HREF) models, which are covered in "A North American hourly assimilation and model forecast cycle: The Rapid Refresh" and "The High Resolution Ensemble Forecast (HREF) system: [..]" respectively. The performance measures are across a wide range of precipitation rates, and reveals that a neural network as such can exploit and learn advanced physical principles for weather forecasting.

Both MetNet-2 and NWP models gather empirical information about the state of the atmosphere for forecasting. The sources of this information vary. Most importantly for MetNet-2 are radar images that capture an instantaneous reflectivity of the amount of precipitation in the air, along with hourly measurements of accumulated precipitation. The radar images are also the ground truth training labels for the model, as these images represent the precipitation. A sequence of these images represents an estimate of accumulated precipitation over time.

The input imagery with added weather context capture an area of 2048km × 2048km per input feature, and is downsampled to a *target patch* of 512km × 512km through a factor of four in the spatial dimensions. Such inputs are concatenated across depth, representing a temporal dimension, and are fed to a Convolutional RNN. The model uses dilated convolutions, as discussed in Chapter 3, Section 3.3.1. The dilated convolutions are extracted in residual connections in the network, which passes information from all of the 512 × 512 positions in the target patch (Espeholt et al., 2021). This target patch is in the very center of the area, which largely resembles the idea of capturing the center of an area in the network presented in this thesis.

5.2 RainNet v1.0: a convolutional neural network for radar-based precipitation nowcasting

> "While expectations in the atmospheric sciences are high, the investigation of deep learning in radar-based precipitation nowcasting is still in its infancy, and universal solutions are not yet available." (Ayzel, Scheffer, and Heistermann, 2020, p. 2)

A study by the University of Potsdam proposed RainNet, "a deep convolutional neural network for radar-based precipitation nowcasting". The study group, consisting of members from the Institute for Environmental Sciences and Geography and Department of Computer Science, proposed a network inspired by the U-net and SegNet families of neural networks. Ayzel, Scheffer, and Heistermann trained RainNet to predict precipitation intensities on a continuous basis at 5-minute lead times. The model uses radar images which covers Germany at 900km × 900km in the spatial dimension. The model outperforms NWP models for this area and competes with other proposed neural network models (Ayzel, Scheffer, and Heistermann, 2020).

The model follows an encoder-decoder architecture that downscales the samples in the spatial dimension prior to upscaling any learned patterns. Skip connections in the network allow for retaining volumes and learned patterns through semantic connectivity between layers. To fit the prerequisites of a U-Net architecture, input in the spatial dimension must be a multiple of 2^{n+1} as described in "U-net: Convolutional networks for biomedical image segmentation". Hence the input radar images are mirror padded, reflecting the borders of the image for a result of 928×928 cells. The inputs are also concatenated in the temporal dimension with four images of radar precipitation reflectivity at 15, 10, 5 and 0 minutes prior to prediction for an output 5 minutes ahead (Ayzel, Scheffer, and Heistermann, 2020). RainNet uses a fully convolutional architecture for predictions, in contrast to both MetNet-2 (as proposed by Espeholt et al., 2021) and the model presented in this thesis.

5.3 Daily reservoir inflow forecasting using artificial neural networks with stopped training approach

> "In the hydrological forecasting context, recent experiments have reported that ANNs may offer a promising alternative for rainfall–runoff modeling." (Coulibaly, Anctil, and Bobée, 2000, p. 2)

Coulibaly, Anctil, and Bobée proposed one of the earliest ANNs for water reservoir inflow prediction. The ANN in the paper is a multi-layered FFNN which uses a multivariate hydrological time series. The group concluded that such a neural network could offer an alternative to statistical models in dynamic and adaptive forecasting (Coulibaly, Anctil, and Bobée, 2000).

The network proposed in the paper uses a technique known as *early stopping*, which stops the training process prior to convergence to avoid overfitting.

The network is designed to receive as input 14 variables including temporal differences and developments to separate input nodes. These variables include the predictor value y_{t-1} water inflow at the previous time step, along with max, min and mean temperatures, precipitation $p_t, \ldots, p_{t-4}$ and snowmelt $s_t, \ldots, s_{t-4}$. The values are relayed to the hidden layer and the network provides a prediction at y_t. The proposed network uses an iterative forecasting method, which uses one output node for multi-step forecasting, in opposition to a direct method (such as in this thesis), which predicts n-nodes for n-step prediction. The direct method relies on past information and predictions to perform a single-step forecast in iterations to complete the same n-step prediction.

The data for the experiment is from the Chute-du-Diable watershed in northern Quebec, Canada. It contains a large water reservoir with 32 years of data of daily natural inflow through precipitation, estimated values of melted snow, and daily measured temperatures. The model is trained and validated with 29 years of data and tested using the last two years. The group focuses on the spring period. The findings show that the neural network models tested (with and without early stopping) predict worse forecasts in one day lead time than conventional forecasting methods. The neural networks outperform the conventional forecasting methods at 2-7 day lead times. The group also found that the neural network with early stopping outperforms the other models with longer than one day lead times. The group conclusively states that a FFNN is a promising tool for inflow prediction and suggests the use of a RNN in future work (Coulibaly, Anctil, and Bobée, 2000).

5.4 Convcast: An embedded convolutional LSTM based architecture for precipitation nowcasting using satellite data

> "Recently, convolutional LSTM has been shown to be successful in solving various complex spatiotemporal based problems." (Kumar et al., 2020, p. 1)

Kumar et al., 2020, suggests a convolutional LSTM coined Convcast for precipitation nowcasting through satellite images. The network is trained on sampled of 10 of NASA's IMERG data, with 30 minute intervals, and predicts the following precipitation data. This data is used iteratively for a forecast of up to 150 minutes lead time. The study shows that nowcasting precipitation with satellite data is a viable option.

The IMERG algorithm unifies multi-satellite precipitation data. This data describes the Global Precipitation Measurement constellation. As the group seeks to predict a short-term forecast, the resolution is only 30 minute intervals of precipitation in mm/h since March of 2014. The networks predictions are the 11th sequence of ten such intervals. As such $\hat{y} = X_1 1$ predicted from $X_0, X_1, \ldots, X_1 0$. For every sequence X_n the consecutive sequence is the label $y_n = X_{n+1}$. The network is built with multiple ConvLSTM layers, each containing cells for spatial and temporal learning. The final layer is a 3D convolutional layer, which outputs the predicted precipitation.

The group compares a simple LSTM model to Convcast and finds that the spatial capabilities in convolutions greatly affect the quality of predictions. Convcast outperforms a simple LSTM and other baseline methods. Conclusively it is stated that Convcast is not accurate enough for nowcasting high precipitation due to few dense samples, containing high precipitation (Kumar et al., 2020).

Chapter 6

Methodology

This chapter explain the structure and architecture of the ANN which has been designed to predict changes in water levels in Helgedalsvatnet. The ANN learns through the time series containing radar images of precipitation rate, recorded precipitation, water level and valve opening described in Chapter 4.

The neural network architecture is coined **MetZoom** as the conceptual structure zooms in towards an area for every time step of meterological data being processed.

MetZoom is a deep, sequential and robust CNN/LSTM hybrid with variable input and output lengths. The idea behind MetZoom was sparked by the proposed MetNet-2 by Google discussed in Chapter 5, Section 5.1. The process of developing MetZoom was inspired by the idea of viewing a larger area than required for prediction of local precipitation, and using separate modules for learning the temporal importance of precipitation and the movement of clouds. The initial design of a pure CNN architecture with separate modules readily connects to a LSTM of the CNNs output. The LSTM - RNN is essentially an outer layer of processing which for every batch *uses* the output of the CNN. The combined hybrid network efficiently copes with the task of not only producing radar and precipitation predictions, but also the inflow to Helgedalsvatnet.

6.1 General remarks

This section will specify the desired output of the proposed neural network and offer a brief overview of the general architecture and components of the network. The proposed architecture processes a sequence of meterological data and output a relative water level up to 12 hours later.

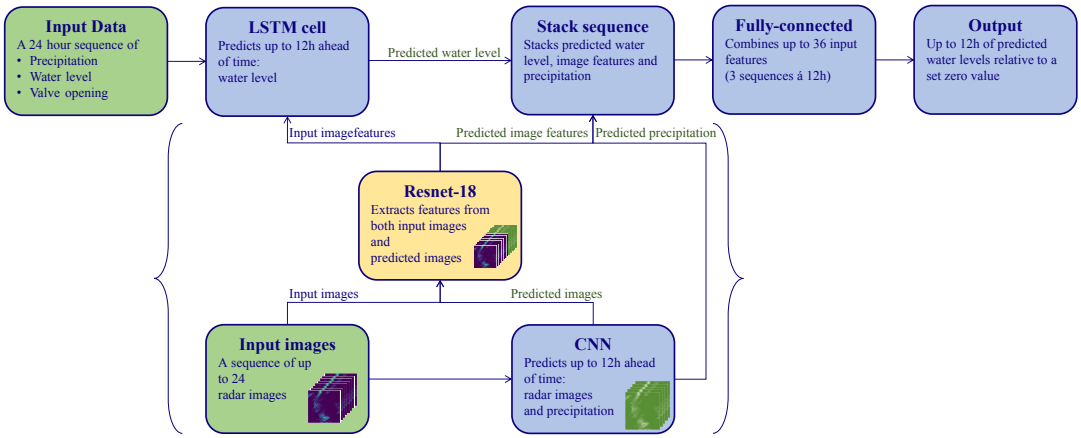

FIGURE 6.1: The general architecture of MetZoom: Separate input data is sent to an LSTM cell and a CNN, and the combined sequence of predictions are sent to a fully-connected layer which produces the expected output.

MetZoom receives, for every sample, an input sequence covering 24 hours of radar images, precipitation, valve opening and relative water levels. A predefined sequence of the samples radar images are processed by a CNN which predicts the future movement and amount of precipitation. The future movement of precipitation is represented by a prediction of radar images of a desired output length. The future precipitation are predicted end-to-end through the radar images. These two predictions are passed along the network as future variables. A LSTM processes the rest of the data to predict 12 hours of changes in the relative water level. This sequence is stacked with the output sequences of the CNN, and passed through a fully-connected part of the network to alter the prediction of the LSTM through the predictions of the CNN. The aim is to either increase or decrease the predicted relative

water levels with the radar and precipitation predictions. Information from both the images in the entirety of the input sequence and the predicted radar images is gathered efficiently through feature extraction by ResNet-18.

6.2 MetZoom: Convolutional Neural Network

The conceptual idea of the MetZoom CNN is depicted in Figure 6.2. The CNN receives a new input image for every hour that passes. The first radar image of the input sequence of the CNN is fed to the first *stage*. The introduction of every new image is passed through a *catch-up block*. Both of these components are explained in further detail in Section 6.2.2. MetZoom is structured as to gain confidence of the movement of liquid water equivalent reflected by radar as more of the input sequence is processed. This is done through recognizing and learning typical recurring patterns for the weather in the region.

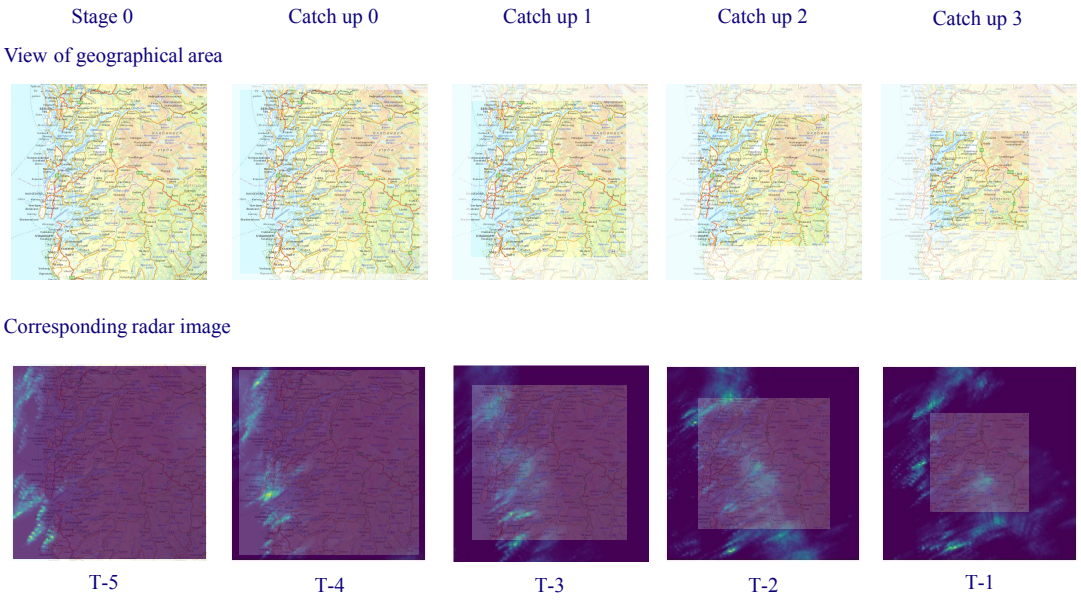

FIGURE 6.2: The first stage (0) of MetZooms CNN part views the entire area. For every catch-up block a new image at time T-n hours prior to prediction is introduced, and the CNN further zooms closer to the target area prior to predictions at $T \ldots T + 12$.

The CNN is initialized with an amount of layers equivalent to the length of the desired input sequence. The number of parameters thus grows with the length of the input sequence. The initialization allows for separate blocks (or modules) to be created per image of the input sequence length. All of these blocks have learnable parameters that are passed through the network. MetZooms CNN therefore learns to evaluate the importance of every image at different time-steps separately.

6.2.1 CNN Input: A Sequence of radar images

The input data of the CNN is a time series of radar images R of liquid water equivalent measuring the rate of precipitation per hour in millimeters. These radar images are ordered by one hour intervals with a total predefined length N_r drawn from every sample of 24 radar images which the network processes. The images are always a set of the last images in a sample. Every sample has 24 images, but the sequence R has the images from the sample at indices $23 - N_r, \ldots, 23$.

Every cell of the input images contain a floating point number representing the precipitation rate at hour h for that area. The total hourly precipitation rate for the area covered by the (whole) image at hour h is denoted by $p_{mm/h}$ or simplified as p_h.

The sequence of images is fed to the network at separate layers of the network, sequentially and ordinally, as illustrated in Figure 6.3.

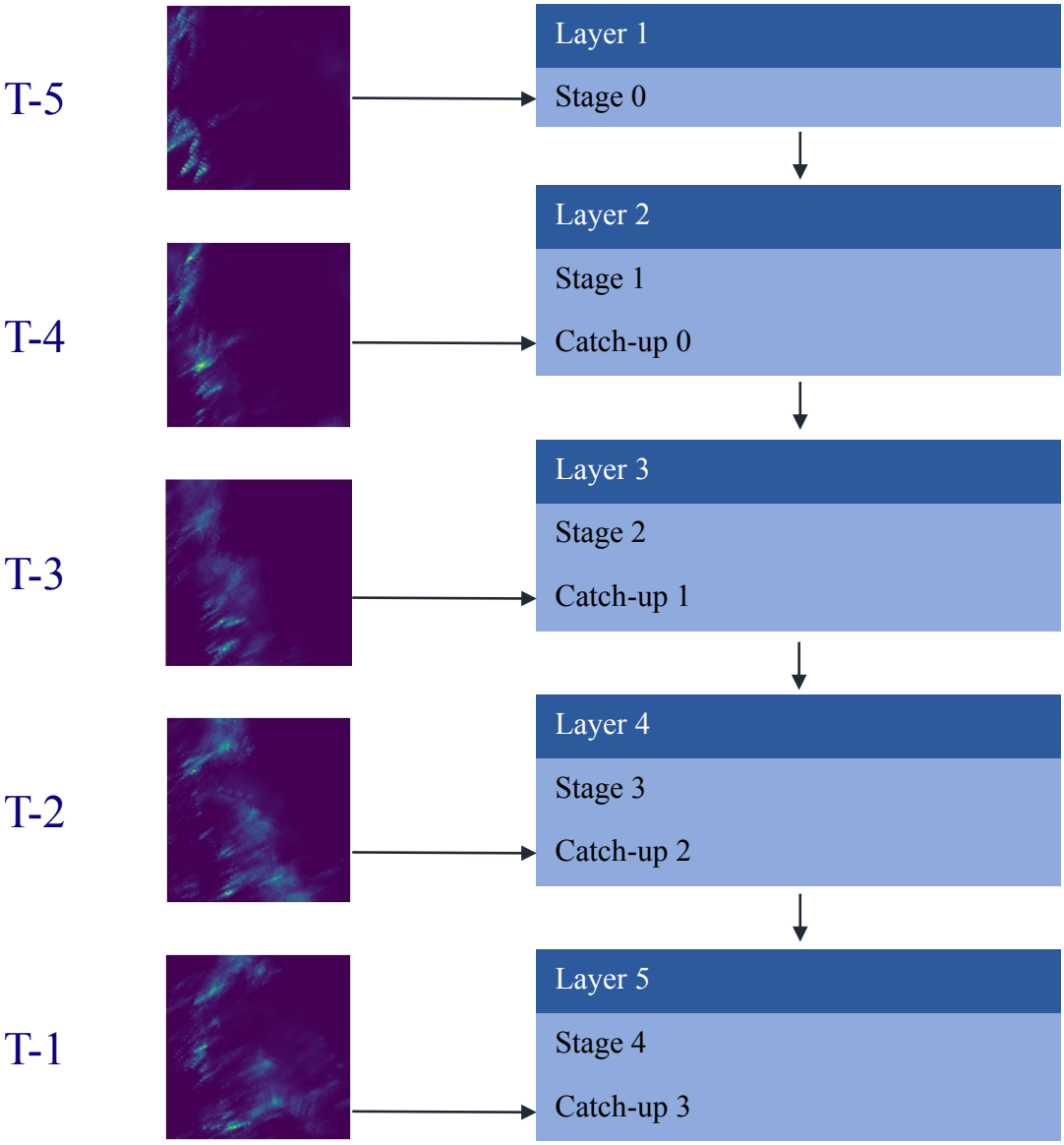

FIGURE 6.3: The first stage (0) of MetZooms CNN receives the first image of the sequence. For every catch-up block a new image is introduced and the information is combined with previous information of the stack.

6.2.2 CNN Architecture: Stages and Catch-up blocks

The layers of MetZoom are built sequentially based on the length N_r of the input radar image sequence covering a time-frame $T - N_r, T - (n-1), \ldots, T - 1$ where $T - i$ specifies the hours prior to prediction. The network bases its predictions of future radar images and expected precipitation solely on these images with their corresponding labels that are the true radar images and

the recorded precipitation in Sauda, as discussed in Chapter 4. The layers (hereby $L_0, L_1, \ldots, L_{N_r}$) of MetZoom contain two different, albeit related, operations; *stages* and *catch-up blocks*.

Stages are analogous to regular convolution layers in traditional CNNs. The stages of the network always process the output from the previous layer. The stages $S_0, S_1, \ldots, S_{(N_r-1)}$ are all structurally identical; These stages, with decreasing kernel sizes and increasing number of channels (until $S_{\lfloor(\frac{N_r}{2}+1)\rfloor}$) employ a 2D convolution, a Leaky ReLU activation function and a 2D Batch Normalization. S_{N_r}, at the next-to-last convolutional layer, changes from a Leaky ReLU to a regular ReLU activation to avoid negative values in the output, as precipitation cannot be negative. Throughout the stages, there is no padding in the convolutions, which encompasses the *zooming* effect shown in Figure 6.2.

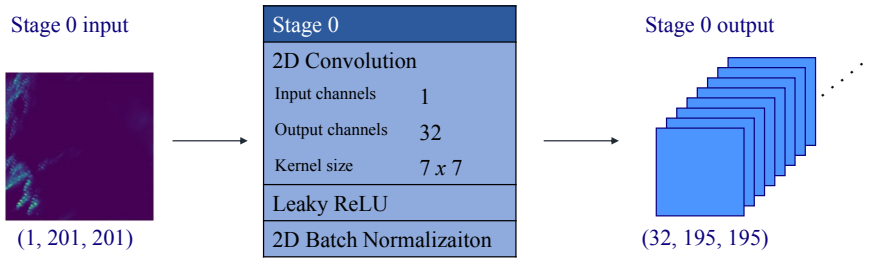

FIGURE 6.4: The first stage (0) of MetZooms CNN part. It receives as input the first radar image of the input sequence.

Catch-up blocks provide MetZoom with uniqueness, as every such block has a separate set of learnable parameters, and are situated in layers $L_1 ... L_{N_r}$. This provides the network with the possibility of learning the importance of the movement of precipitation within different sets of radar images. A catch-up block at L_k (k being an arbitrary number between $1, \ldots, N_r$) contains a sequential module which copies stages $S_0, S_1, \ldots, S_k$ and finishes with a Sigmoid activation as seen in Figure 6.5. The Sigmoid activation provides an output representing the networks confidence in the precipitations future movement.

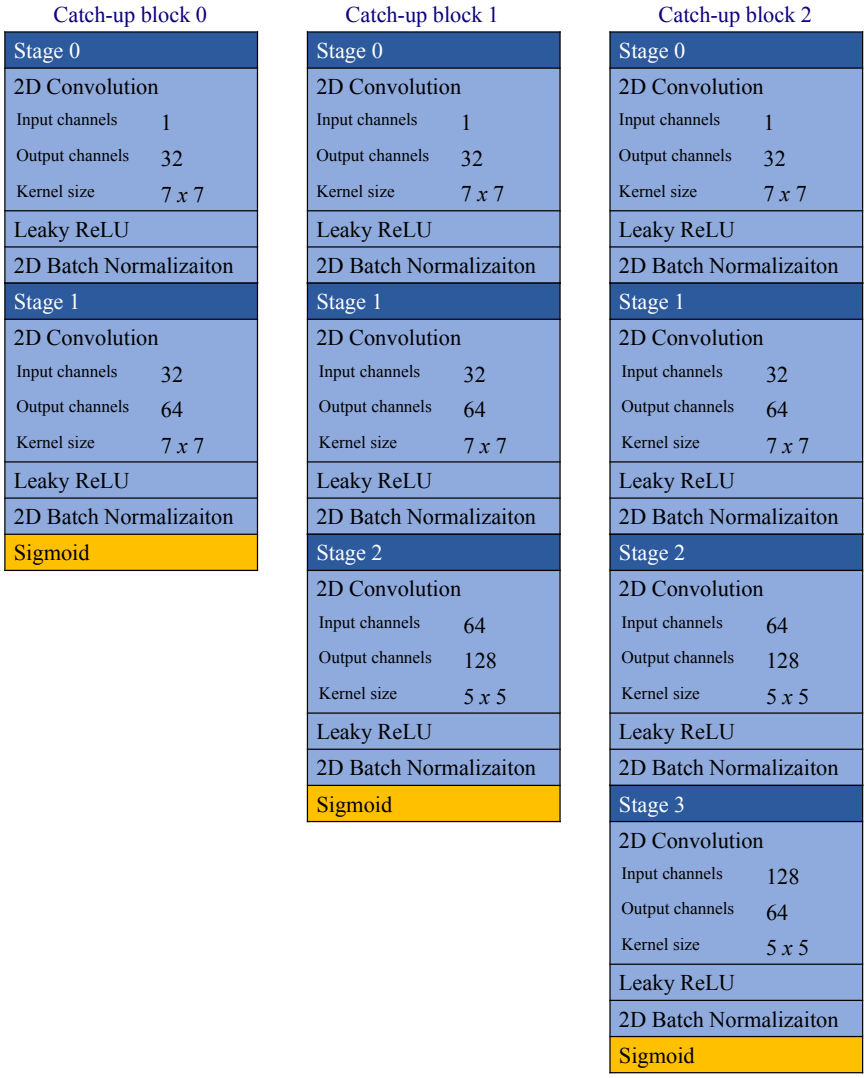

FIGURE 6.5: Thee catch-up blocks at layers 2, 3 and 4 following the first layer containing only stage 0. For every layer there is a new stage, and a catch-up block that includes that stage and all prior stages. E.g. the catch-up block at layer 2 has all stages 0, 1 and 2, where stage 2 is introduced at the same layer as the catch-up block.

As the catch-up blocks process the images $R_1, \ldots, R_{N_r}$, and being their own network modules apart from the stages, the network can learn which images in the sequence are most important for a future prediction – rather than assuming that the images are linearly connected and equally important. The two most important effects of a catch-up block is processing the next radar image of the input sequence and providing a confidence of the precipitations aerial movement since the last time-step. As the catch-up block

finishes with a Sigmoid activation function, it essentially produces an estimation of the likelihood of rain either moving, decreasing or neither in the next time-step. This estimation is element-wise multiplied with the outputs of the corresponding stage across its entire depth. The product is passed to the next stage in the consecutive layer and the corresponding catch-up block processes the next image in the sequence.

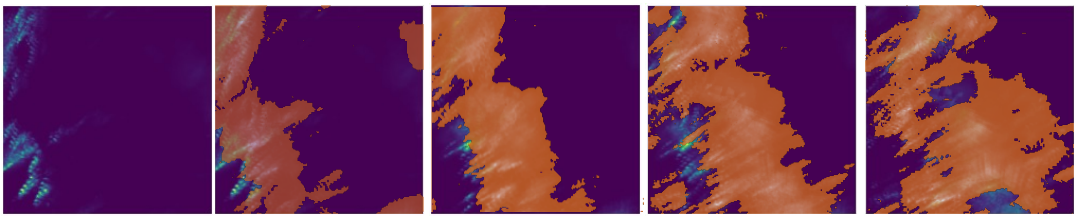

FIGURE 6.6: The image introduced at stage 0 is processed along with the output of the catch-up block in the next layer, which shows the confidence (in orange) of where the precipitation will move in the next hour. This process continues through the catch-up blocks of each layer.

Through the catch-up blocks MetZooms CNN aims to learn not only the movement of precipitation captured by radar images, but also the impact some of this movement has on regions of the next time-step with dense local information. Figure 6.6 shows the information that the catch-up blocks intend to capture, which is essentially the movement of precipitation. The implementation of adding input images to corresponding layers when zooming into the area covered geographically leads to a continuous flow of information reminiscent of residual connections. The catch-up blocks remind the network of the task at hand, predicting where the precipitation will accumulate and how certain that precipitation is.

6.2.3 CNN Output: Prediction of radar images and precipitation

MetZooms CNN outputs a sequence of future radar images $\hat{R}$ of length $N_{\hat{r}}$ along with corresponding amounts of expected precipitation $\hat{P}$ of length $N_{\hat{p}} = N_{\hat{r}}$. Although the images are collected prior to the fully connected layers of the network, both predicted sequences are measured using loss functions and included in the backward pass and optimization step. The sequences of both $\hat{R}$ and $\hat{P}$ cover the time-frame $T, \ldots, T + N_{\hat{r}}$ following the input sequence ordinally ($N_{\hat{r}}$ representing the length of the output sequence). Figure 6.7 is an illustration of the entire architecture of the CNN in MetZoom (in the case of $N_r = 5$ input images and $N_{\hat{r}} = 3$ output images). The final layers output channels correspond with the desired output sequence length $N_{\hat{r}}$.

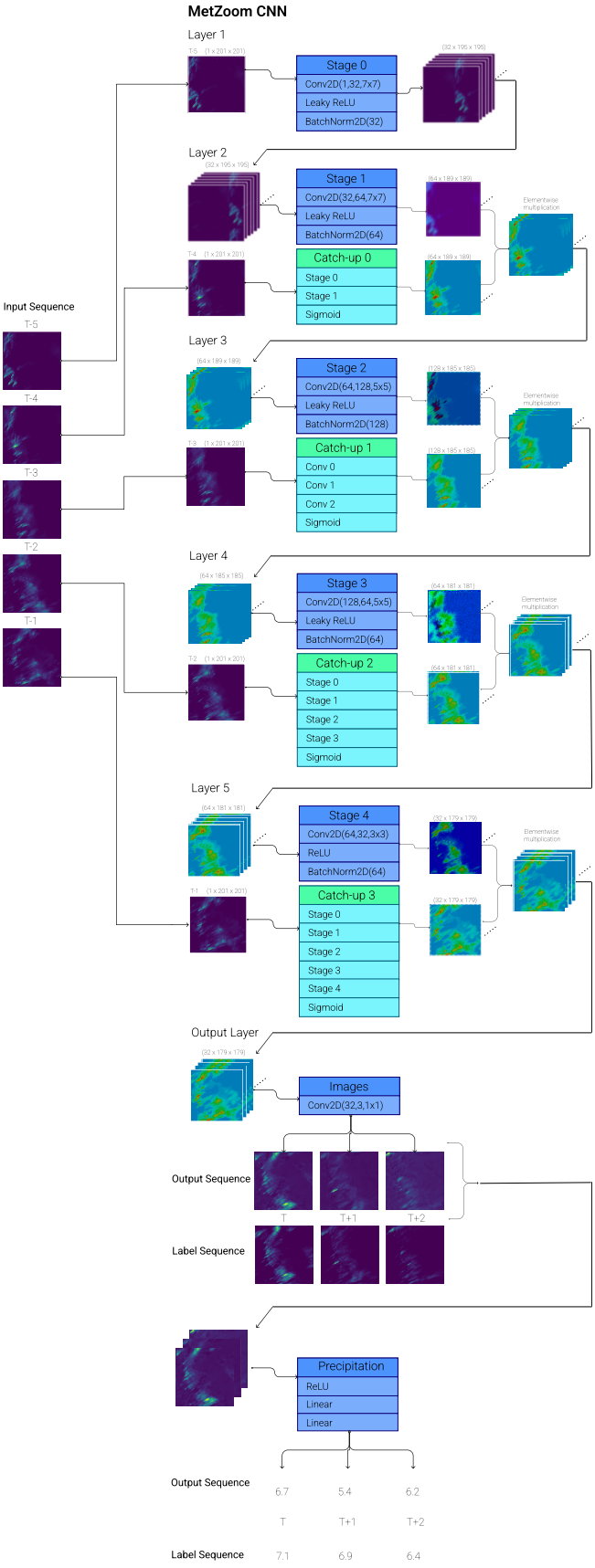

FIGURE 6.7: The architecture of MetZooms CNN with $N_r = 5$ input images and $N_f = 3$ output images. The images are sequentially ordered, every image one hour apart from the previous image.

6.2.4 CNN performance: Loss and convergence

The loss between sequences R and $\hat{R}$ are calculated separately from that of P and $\hat{P}$, however, they are included in the same step. Rather than focusing only on the expected precipitation, precise prediction of radar images can provide information in other areas than that specific to the measuring station for precipitation which provides P. The advantage of this separation is the improvement of radar image prediction for use in the next step of the network - the feature extraction from the images. Although the precipitation may be the more important value, the focus on predicting more accurate radar images allows for purposeful feature extraction of *future* radar images along with the images in the sample.

The loss on the radar image prediction is calculated through MSE loss, as explained in Chapter 3, Section 3.2.1. This is largely due to a desire to amplify the importance of both heavy precipitation in dense radar images, as well as the existing precipitation in sparse images. The MSE loss allows for capturing both of these instances, because of the detection of outliers.

The loss on the precipitation is calculated using L1 loss, as explained in Chapter 3, Section 3.2.1. The precipitation is linearly interpolated, as described in Chapter 4, Section 4.3.1, and hence L1 loss is more fitting as when linearly interpolating there are likely no outliers, and a smooth increase or decrease. The reasoning behind this choice is capturing the increasing or decreasing precipitation rate in the predictions, and furthermore capturing the same trend in the water level predictions.

The losses are then combined into a total loss which can be reviewed to inspect whether the CNN improves on both tasks.

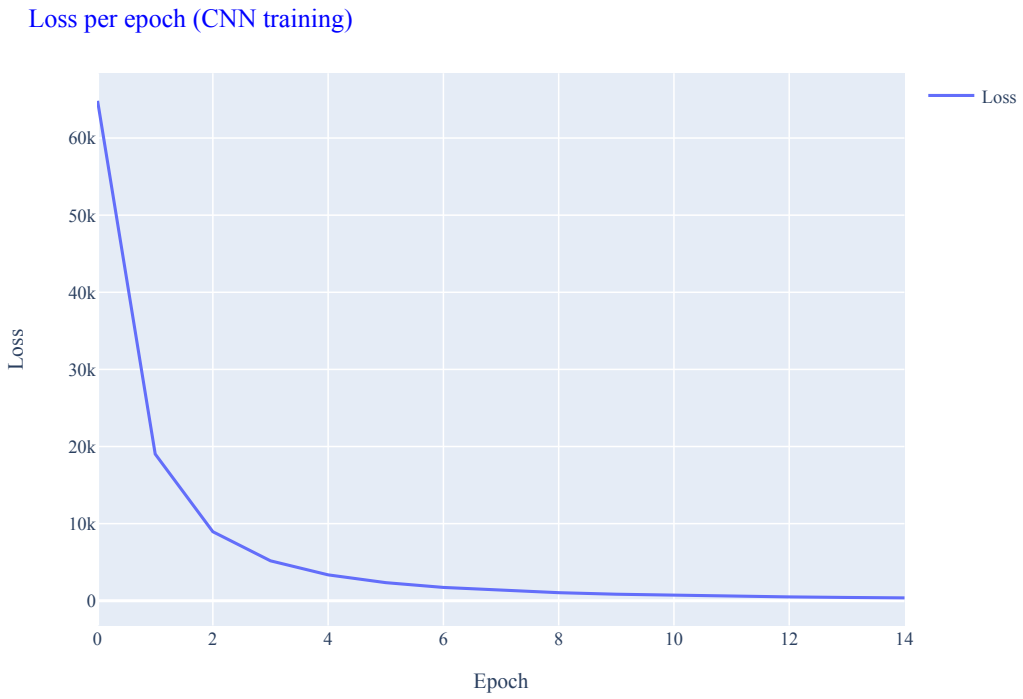

FIGURE 6.8: The loss of MetZooms CNN converges around 12 epochs when predicting 12 hours ahead of time. The loss of precipitation is almost lost in the large image loss in early epochs.

Figure 6.8 visualizes the loss function of the CNN in MetZoom. The loss on the y-axis is high due to the the pixel-wise loss on radar images.

6.3 MetZoom: Long Short-Term Memory RNN

The RNN which uses the outputs of the CNN in MetZoom is an LSTM. As MetZoom processes sequences in every sample - every sequence is recognized as both impactful in the short-term aspect of predicting immediate inflow, and included in the long-term aspect, as the data covers such a long time period. The LSTM learns the long-term dependencies of the past inputs and their consequences, and the short-term impacts of the immediate inputs (as explained in Section 3.4.1).

6.3.1 LSTM Architecture: Using the ResNet-18 extracted features of radar images

In this section a brief explanation is offered on the use of ResNet (as introduced by He et al., 2016) in this architecture, and how it is used as a tool for feature extraction from images, as shown in Figure 6.9. The specific choice is an instance of ResNet recognized as ResNet-18, where 18 describes the number of layers in the model. The ResNet used in the architecture is neither pre-trained nor directly trained through the network architecture. The training related to the inclusion of ResNet is only that of extracting the correct features from the provided sequence of radar images. The instance of ResNet is used as a component in the LSTM.

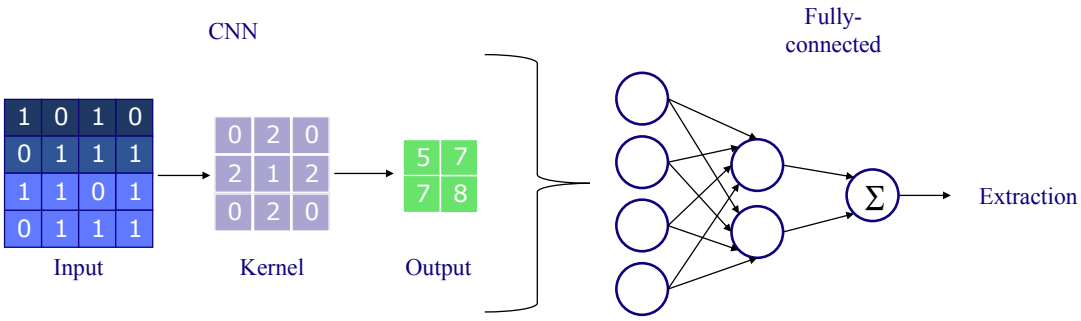

FIGURE 6.9: A simplified illustration of how ResNet 18 is used in the RNN. An extracted feature is typically drawn from a CNN and passed through a Fully-connected feed-forward neural network. The end result can be considered a feature representing some value of the input image.

As explained in Section 6.2.1 a sequence of radar images R are fed to the CNN layer of the network. Through ResNet-18 these images are transformed to a sequence of features alongside the prediction of future radar images made by the CNN, $\hat{R}$. These features are an abstract representation of precipitation in the target area shown in Figure 4.3 in Chapter 4.

The features extracted are those of ResNet-18s representation of the provided images at its penultimate layer (18). This layer is altered slightly such that ResNet-18 provides a vector containing one floating point number for

each image provided. These floating point numbers abstractly represent the general amount of precipitation in the region that the CNN zooms in to for every time step both prior to prediction and in the future.

These values are used for *multilinear subspace learning*, a form of dimensionality reduction which can be used on data that has been vectorized. Essentially, this process is a mapping from a higher dimensional space to a lower dimensional vector with retention of organizational structure. The $24 + N_{\hat{r}}$ radar images are reduced to two one-dimensional vectors V and $\hat{V}$ by ResNet-18, as illustrated in Figure 6.10.

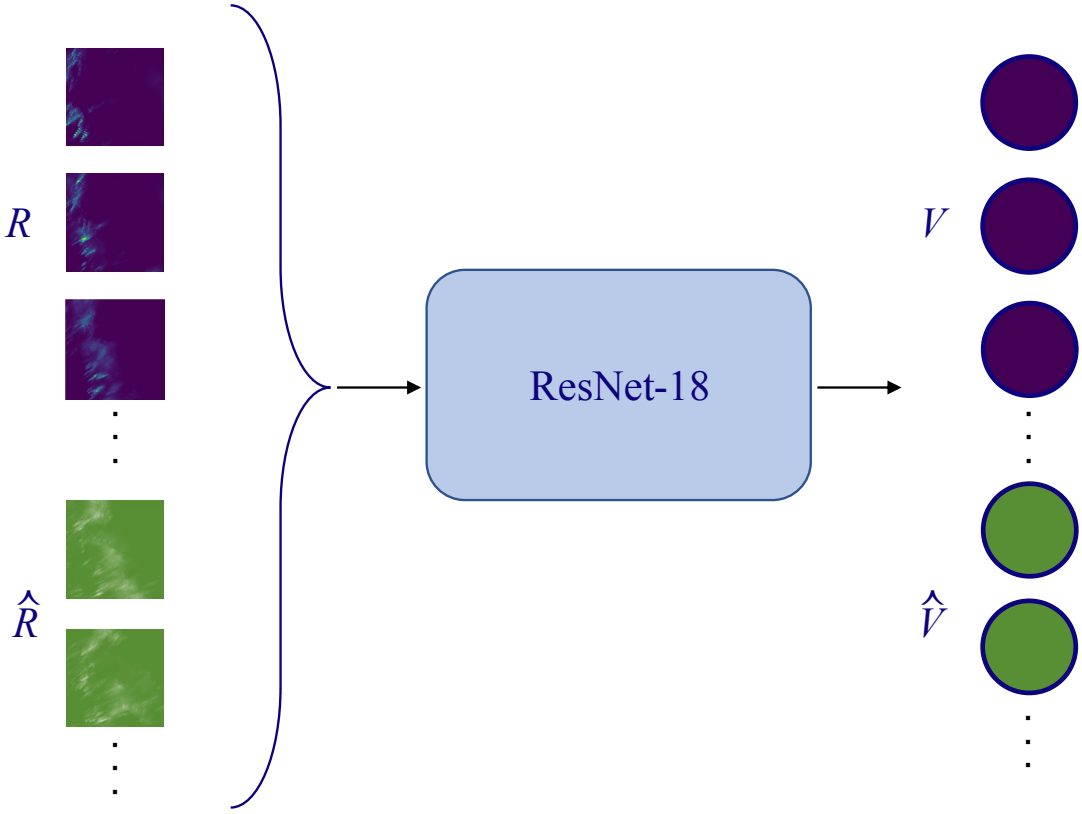

FIGURE 6.10: A simplified illustration of how ResNet 18 transforms images to floating point numbers.

In a case where the network receives N_r radar images and predicts $N_{\hat{r}}$ future radar images, these sequences are processed simultaneously as a combined sequence V of features $V_0, V_1, \ldots, V_{N_{\hat{r}}}$ in ResNet-18. The features in the sequence representing the input sequence R and the images prior to them in

the sample. The features in the sequence representing the output sequence $\hat{R}$, $V_{N_r+1}, V_{N_r+2}, \ldots, V_{N_{\hat{r}}}$, are stored in a sequence $\hat{V}$ which is passed to the output layer along with $\hat{P}$.

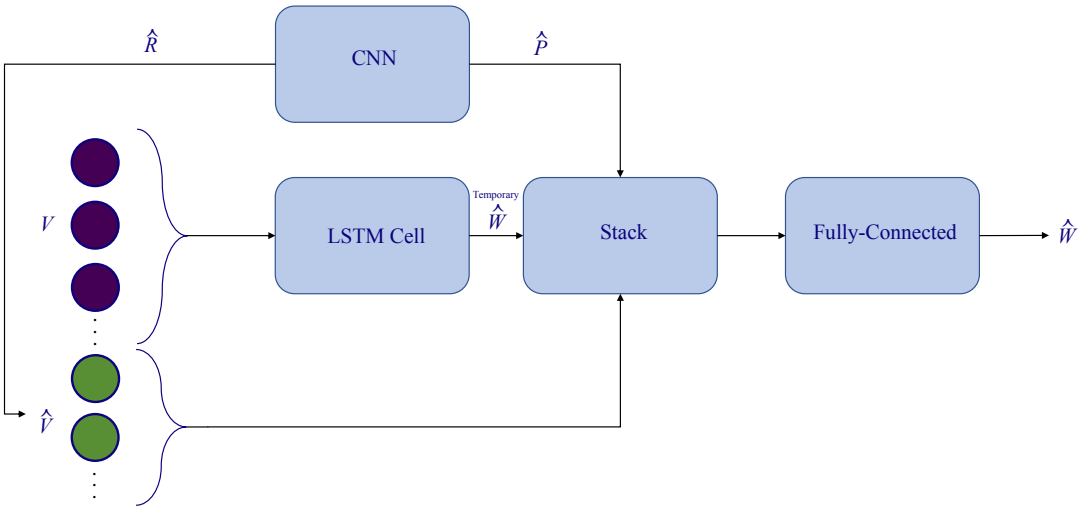

FIGURE 6.11: A simplified illustration of how the extracted features are used as input in the LSTM and fully-connected layers. The output of the CNN $\hat{R}$ is transformed to $\hat{V}$. $\hat{W}$ is predicted by the LSTM, and then altered in the fully connected layers by $\hat{V}$ and $\hat{P}$.

As radar images cannot be fed directly to an LSTM without it being a direct CNN-LSTM (meaning it reads images and not multi-type data), the feature extraction allows the representations V and $\hat{V}$ to be fed alongside the other data (as depicted in Figure 6.11). The design of MetZoom uses a separate (but connected) CNN to predict future images, which a CNN-LSTM would not do, and therefore the separation of the input data is needed.

6.3.2 LSTM input: 24h time series

The LSTM receives as inputs a sequence of 24 hours, including all of the data of the final dataset as shown in Table 4.5. This sequence includes the precipitation P, valve opening O, water level W and radar images R. Prior to processing the sequence in LSTM cells, the network feeds a subset of the sequence R of N_r radar images to the CNN, and receives an output of $\hat{R}$ radar

images along with predicted precipitation $\hat{P}$. The images in R and $\hat{R}$ are transformed to features V and $\hat{V}$. The features in V replace the images R in the original input sequence and are passed through the LSTM layer. Hence the input of the LSTM layer is exactly a sequence of 24 hours prior to the point of prediction, including the extracted features from the radar images.

6.3.3 LSTM output: Up to 12h water level prediction

The LSTM outputs a sequence of $N_{\hat{w}}$ hours of predicted relative water levels $\hat{W}$ ahead of the "current" time, up to 12 hours. This time-frame corresponds to the time-frame covered by the predicted radar sequence $\hat{R}$.

The sequence $\hat{W}$ is then stacked with $\hat{V}$ and $\hat{P}$ (as seen in Figure 6.11), recognized as the *future sequence*, all of length $N = N_{\hat{w}} = N_{\hat{r}}$. This stacked three-dimensional vector is passed through two fully connected layers, such that the predicted radar images $\hat{R}$ (through their extracted features $\hat{V}$) and the predicted precipitation $\hat{P}$ are allowed to affect the LSTM predicted water level $\hat{W}$. This results in an adjusted prediction which is considered the final output of MetZoom.

6.3.4 LSTM performance: Loss and convergence

The loss in predicted water level $\hat{W}$ is calculated with L1 loss, as explained in Chapter 3, Section 3.2.1. The reasoning is similar to that of the predicted precipitation loss. The loss is represented as the difference between the predicted actual change in water level relative to the predefined zero value.

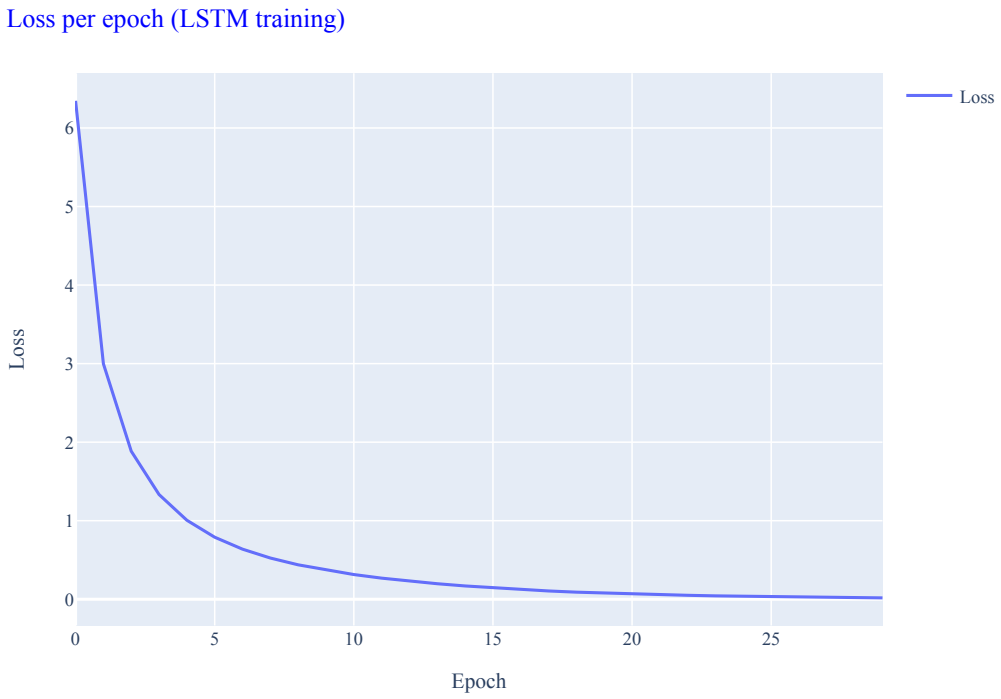

FIGURE 6.12: The loss of MetZooms LSTM converges around 25 epochs when predicting 12 hours ahead of time.

Figure 6.12 shows that the network gradually learns to predict the change in water level within a few centimeters.

Chapter 7

Results

This chapter will review the results that have been achieved, and compare these results to some other forecasting models. In accordance with the architecture of MetZoom and the nature of the sequential prediction system of images, precipitation and relative water level, the results will be reviewed in this order. The quality of the predicted radar images, and accordingly the accuracy of precipitation predictions will be discussed. Finally the most important (i.e the real goal) prediction, the change in relative water level in Helgedalsvatnet will be reviewed.

The assumptions made about the data in Chapter 4 and the complexity of the problem mentioned in the introduction of the same chapter is considered. In Chapter 9 this complexity as well as increasing complexity in possible future works will be reflected on.

The predictions made by MetZoom and other models in Section 7.4 are in the time period *2019-06-02 20:00:00* to *2021-08-26 11:00:00*. This is the entirety of the test data. As this is a very long sequence, it is very difficult to see how accurately the model predicts the changes in relative water level. Therefore a snippet of this sequence with sufficient variation is chosen to illustrate the predictions. The snippet is the time period *2020-06-18 04:00:00* to *2020-07-23 11:00:00*, approximately at the center part of the full time frame.

7.1 Predicting Radar Images

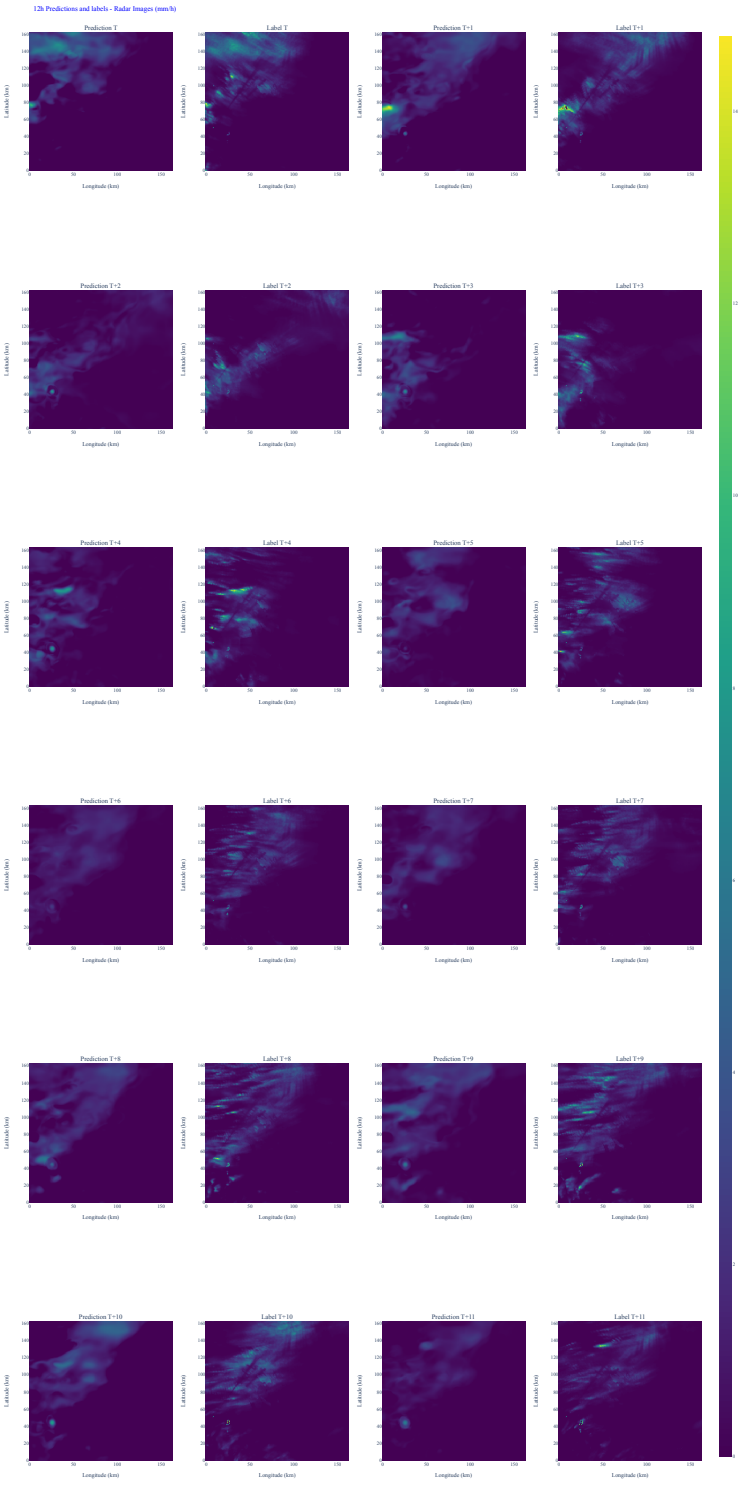

FIGURE 7.1: An example of a 12h prediction of radar images along with corresponding label images. Every row has 4 images, where the 1st and 3rd images are predictions, and the 2nd and 4th are their respective labels.

The CNN in MetZoom predicts radar images up to 12h ahead of time. There is a general lag in predictions, which might have some impact on the prediction of water level.

Figure 7.1 shows a prediction of 12h of radar images along with their corresponding labels, where the mentioned lag is slightly apparent in the west to east direction. The images are a bi-product of the predictions of MetZoom. The aim behind these predictions is learning *where* the precipitation will pass, and extract information on what precipitation will fall in to Helgedalsvatnets precipitation field. This is done through the feature extraction described in Chapter 6, Section 6.3.1.

The quality of the radar image predictions shows that MetZoom does not fully capture the precipitation reflected by the radar images, and provide a smoother distribution than the actual labels. This results in MetZoom not capturing heavy local precipitation, but does capture the general movements and amounts of precipitation.

7.2 Predicting Precipitation Through Radar Images

The CNN in MetZoom predicts precipitation up to 12h ahead of time through fully-connected layers, directly from the corresponding predicted radar images. The labels of the precipitation data are the actual recorded precipitation rates from the dataset described in Chapter 4. MetZoom fails to capture heavy precipitation in the predicted radar images. The full timeline of precipitation predictions in Figure 7.2 show that the general precipitation predictions are lower than the label precipitation rates. This is due to the smoothness of the predicted radar images from Section 7.1.

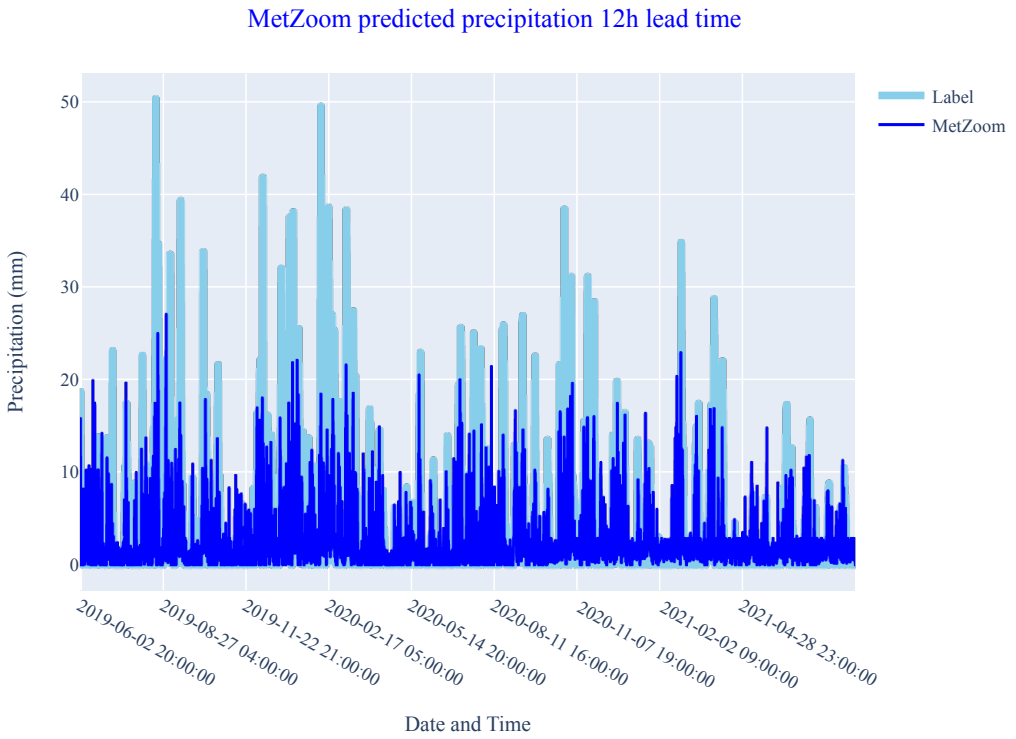

FIGURE 7.2: The result of precipitation predictions when MetZoom predicts 12 hours ahead of time. The label precipitation is often higher, meaning that MetZoom doesn't capture heavy precipitation.

A better review of the accuracy of precipitation predictions is shown in Figure 7.3. It is clear that the predictions are more noisy than the label data, and generally inaccurate with either heavy or zero precipitation.

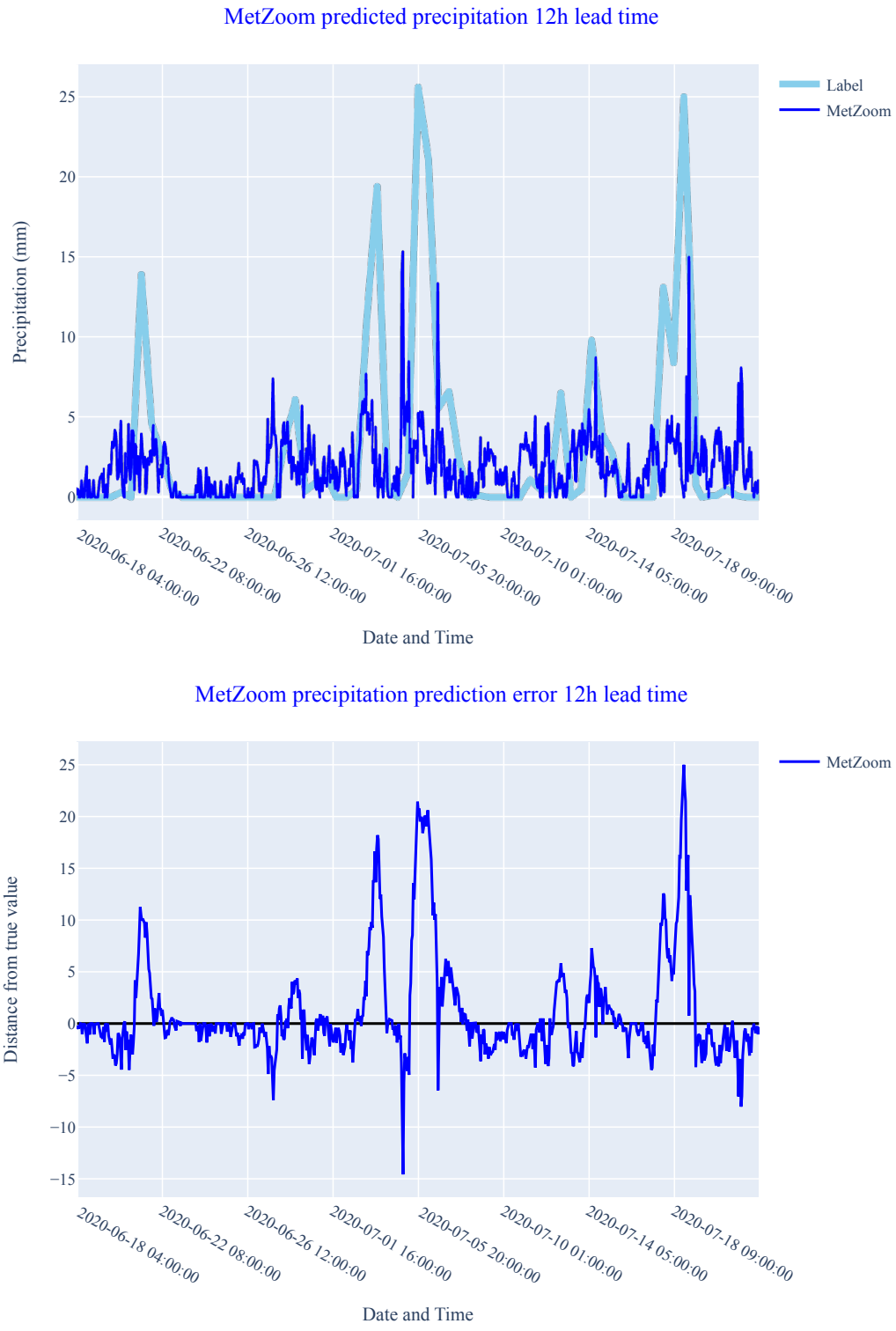

FIGURE 7.3: The result of precipitation predictions. The time series and predictions cover the central snippet of the test data. The errors show the distance between the predicted precipitation and the actual precipitation.

To measure the accuracy of precipitation predictions at different lead times,

Figure 7.4 shows the Standard Error per lead hour. The results show that the predictions have an error range between 4 and 6 millimeters for all lead times. MetZoom performs only slightly better at predicting precipitation at shorter lead times than at longer lead times.

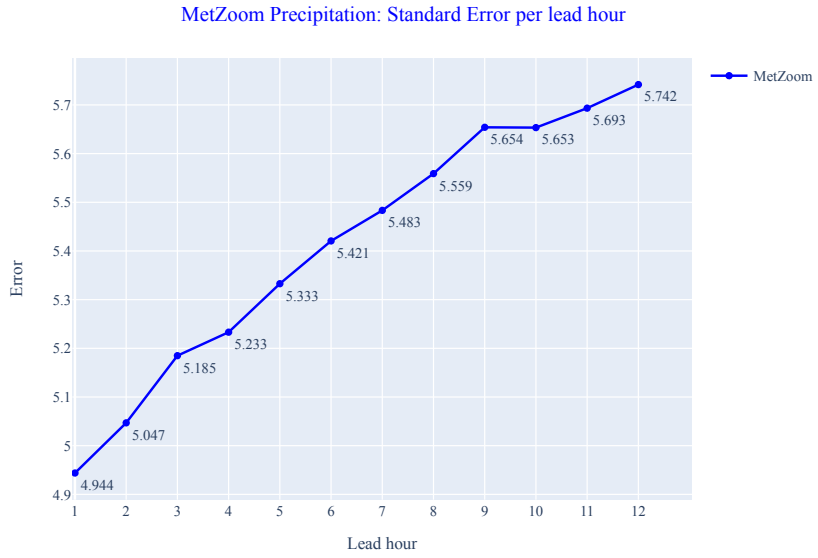

FIGURE 7.4: The Standard Error in millimeters of precipitation predictions per lead hour. The errors at different lead times is quite similar.

7.3 Predicting Relative Water Level (Helgedalsvatnet)

MetZoom predicts the relative water level of Helgedalsvatnet with up to 12 hour lead time. The labels of every time-step of relative water level in the test set is shown in Figure 7.5. This is the main focus of the thesis, and the results of predicted radar images and precipitation in Sections 7.1 and 7.2 are provided to demonstrate the quality of the inner functionality of the model.

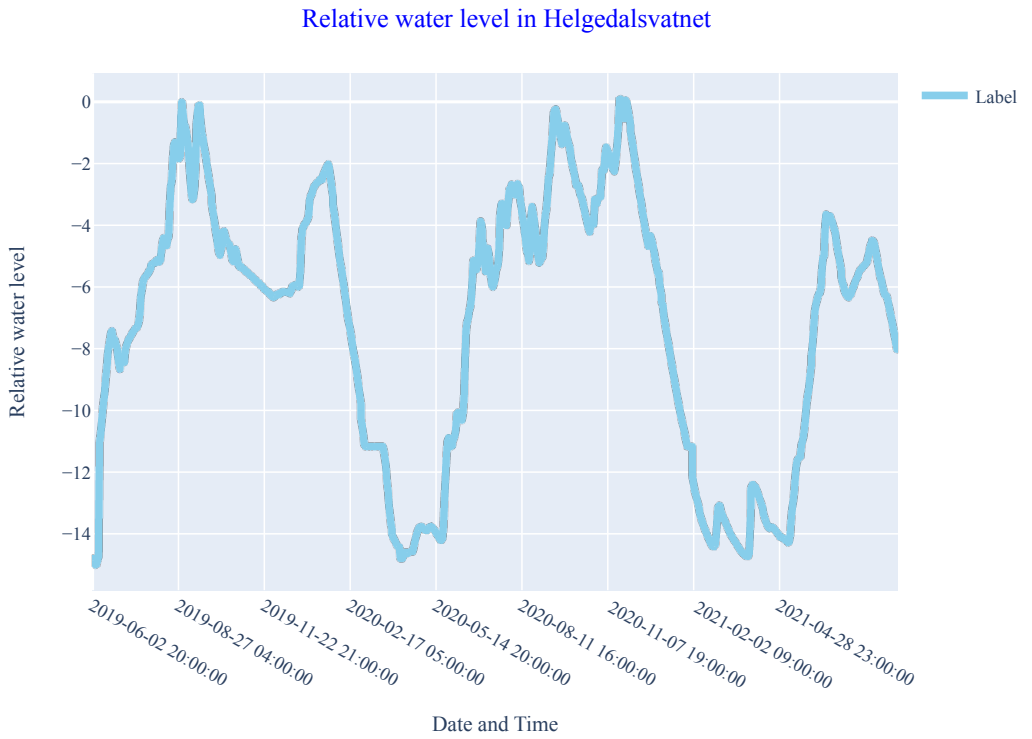

FIGURE 7.5: The relative water level in meters in the test set of the data.

The predictions are measured with three different metrics, as shown in Table 7.1. The predictions are measured in step-wise sequences of 100 hours. This provides a better estimation of how good the predictions are in different situations, such as rapid increase, slow decrease or no change in water level. The three metrics used is the Mean Squared Error (MSE), Mean Absolute Error (MAE) and Standard Error (SE) for every lead hour.

Lead hour	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
MSE	0.0005	0.0010	0.0009	0.0010	0.0016	0.0021	0.0023	0.0029	0.0036	0.0044	0.0053	0.0065
MAE	0.0134	0.0263	0.0227	0.0179	0.0278	0.0321	0.0261	0.0298	0.0358	0.0396	0.0435	0.0516
SE	0.0179	0.0221	0.0237	0.0284	0.0355	0.0406	0.0433	0.0490	0.0548	0.0611	0.0673	0.0744

TABLE 7.1: MetZoom error metrics in meters per lead hour.

Figure 7.6 shows the predicted relative water level at 12 hour lead time in the shortened snippet of the test data. The results show that MetZoom predicts an added level of noise and some over-estimates in change at the

peaks. The top image shows the prediction and label, whilst the bottom image shows the error $(y - \hat{y})$ for the same time frame.

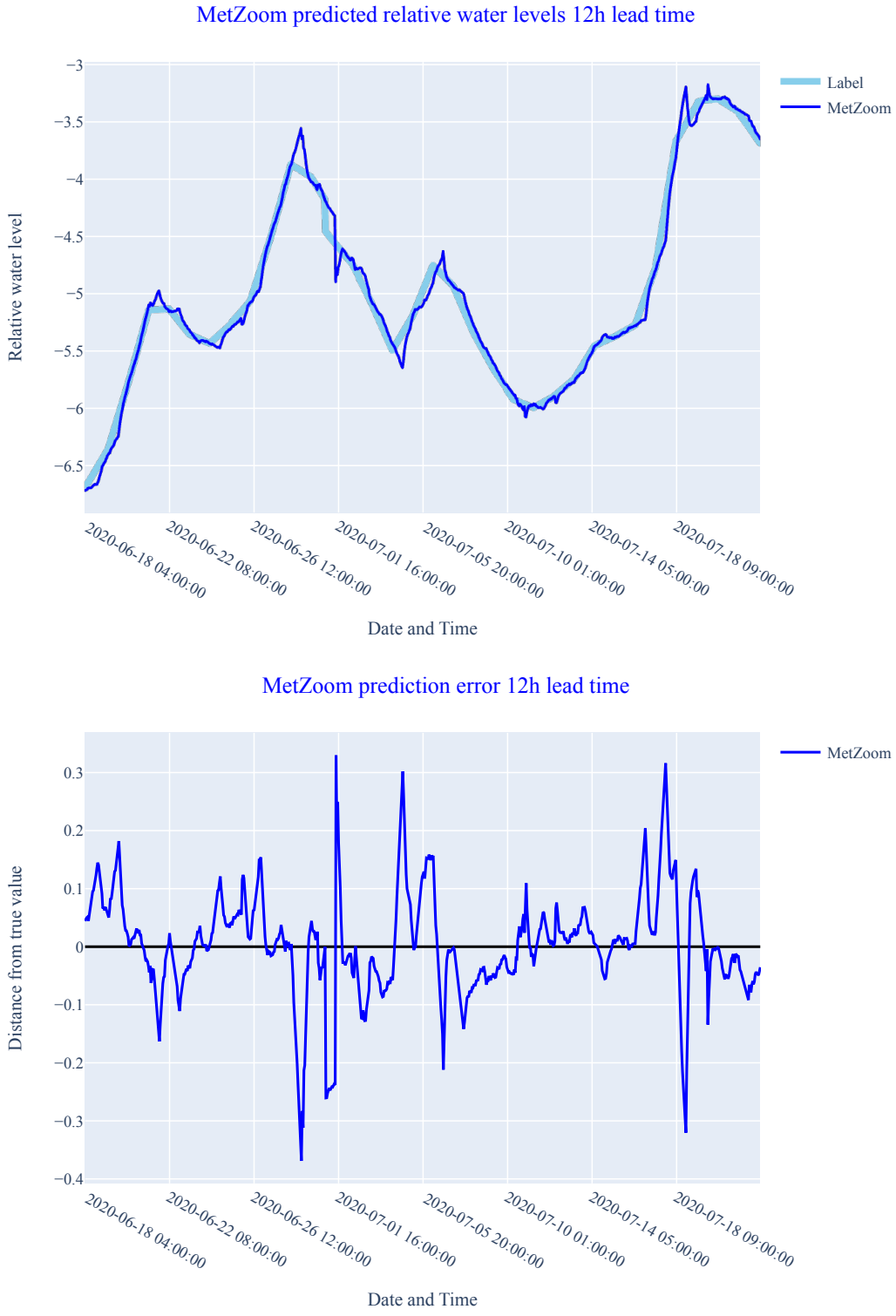

FIGURE 7.6: The result of water level predictions in meters when MetZoom predicts 12 hours ahead of time. The label water level is smoother than the predictions.

7.4 Comparing predictions to other models

The same input data and metrics are used to review the performance of MetZoom compared with other models. The models chosen for comparison is a naive baseline model, an ARIMA model and a more simple LSTM which excludes the radar images.

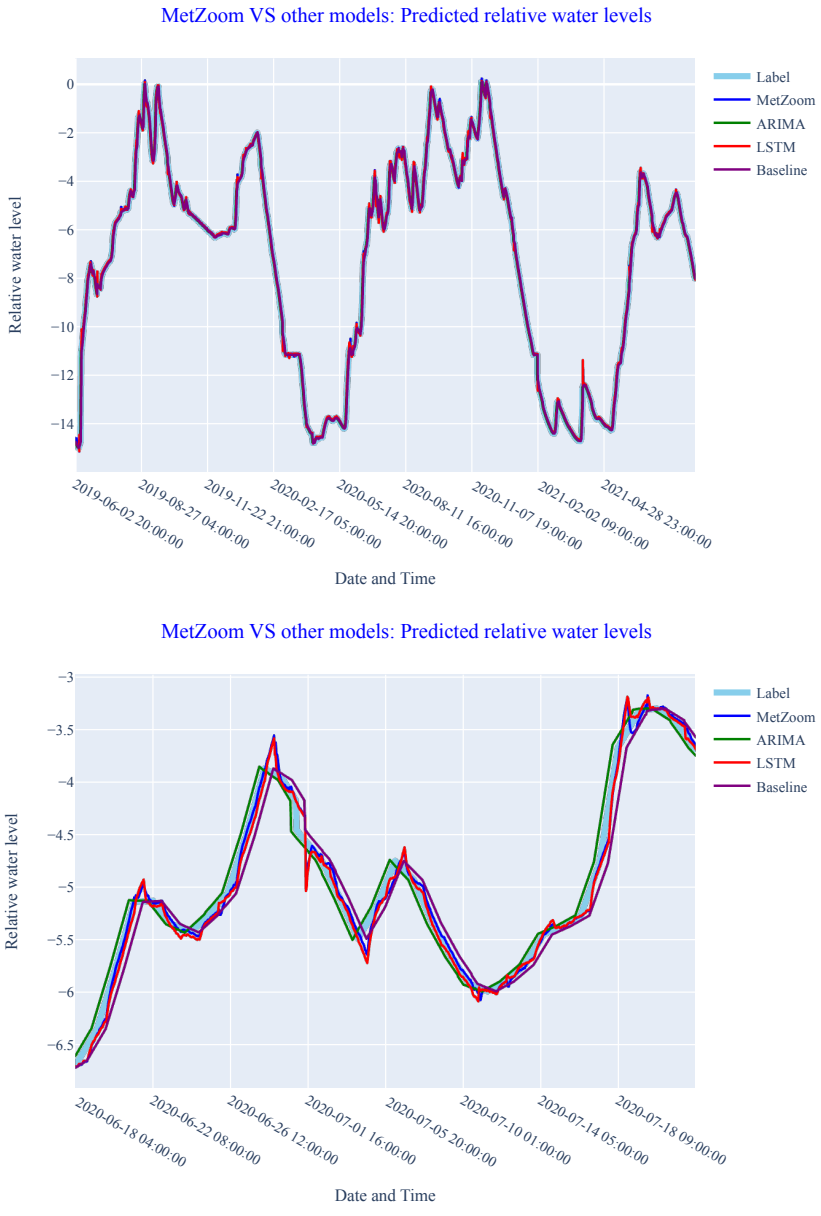

FIGURE 7.7: The different models predictions of relative water levels in meters for both the entirety of the test data and the center snippet.

The first section compares all the tested models with the previously mentioned metrics. The following sections discuss direct comparisons of MetZoom to the respective models.

The models are compared based on the full test set. Only the center snippet is used to illustrate the performances, as seen in Figure 7.7.

7.4.1 MetZoom VS. other models

All of the models closely follow the label line in Figure 7.7. It is important to note that the water level is measured in meters. Therefore small deviations in predictions becomes significant with regards to performance. The same metrics are used for all the models, with the same sequences of 100 hours.

The results in Figure 7.8 show that all models are accurate in the shorter lead times. The naive baseline model outperforms the other models with a lead time within 2 hours. This is likely due to the linear interpolation of the data along with many periods with no change. The ARIMA model outperforms all of the other models with lead time shorter than 7 hours. At lead times greater than 7 hours MetZoom outperforms the other models with a slight margin. The LSTM model competes with the other models, but does not dominate within a specific range of lead times.

FIGURE 7.8: The different models are compared with MSE, MAE and SE in meters on the entire test set at different lead times.

7.4.2 MetZoom VS. Naive Baseline

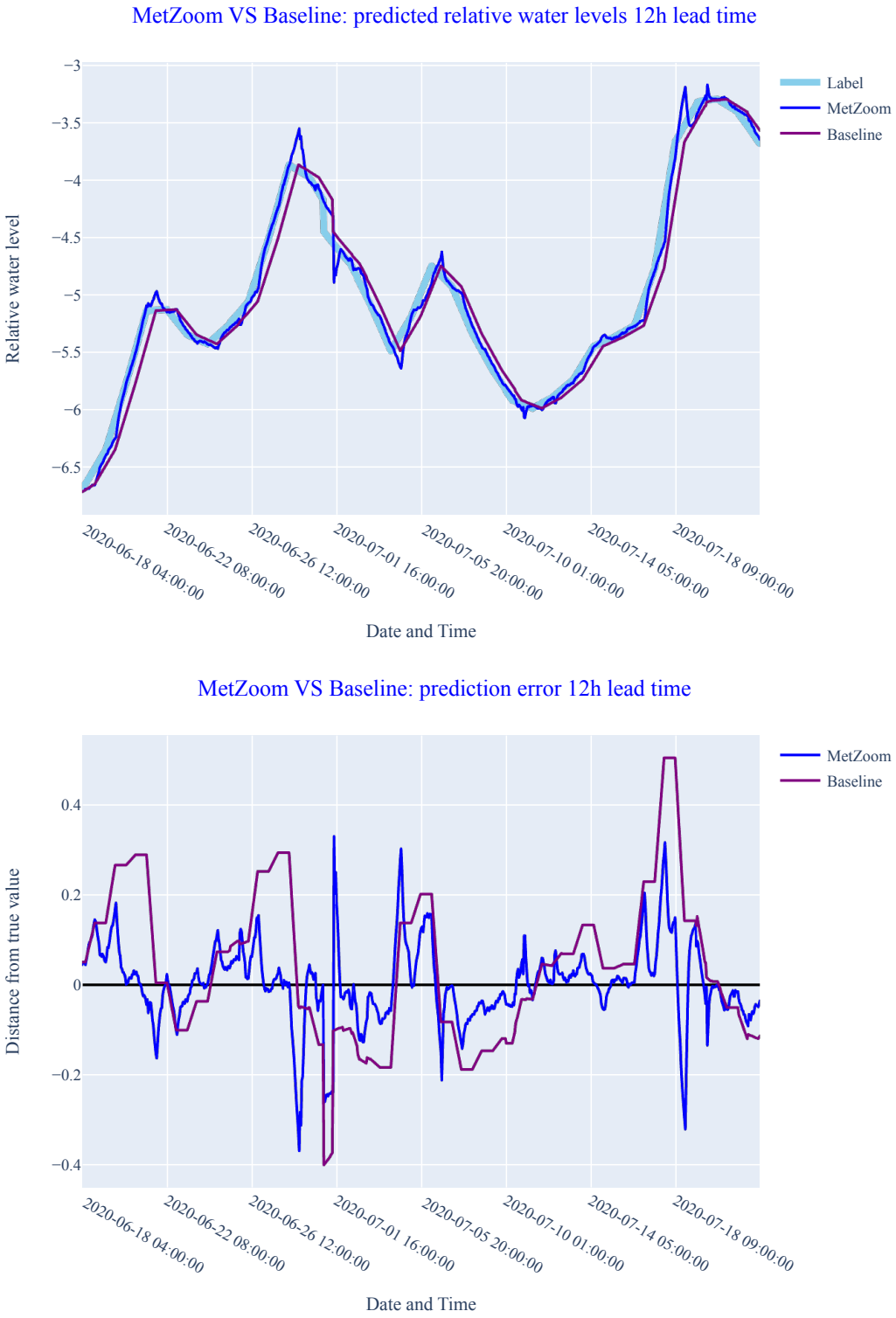

FIGURE 7.9: The top graph shows the predictions by MetZoom and the naive baseline model. The bottom graph shows the errors in the same time frame.

The naive baseline model predicts the water level to remain constant for the period of the forecast. For example, when the naive baseline model predicts changes in relative water levels 12 hours ahead, it simply reproduces the value at the current time 12 times. As changes in water level are quite slow, this model is accurate at short time frames (1-3 hours), but is quite inaccurate with longer lead times.

The naive baseline model outperforms MetZoom when forecasting in the near future (1-3 hours ahead), but from this point on performs worse than MetZoom. Figure 7.9 shows the predictions and errors of both models with 12h lead time for the center snippet.

The results show that the naive baseline models predictions are more similar to the smooth label in shape and form, but has a constant delay at 12h lead time.

7.4.3 MetZoom VS. ARIMA

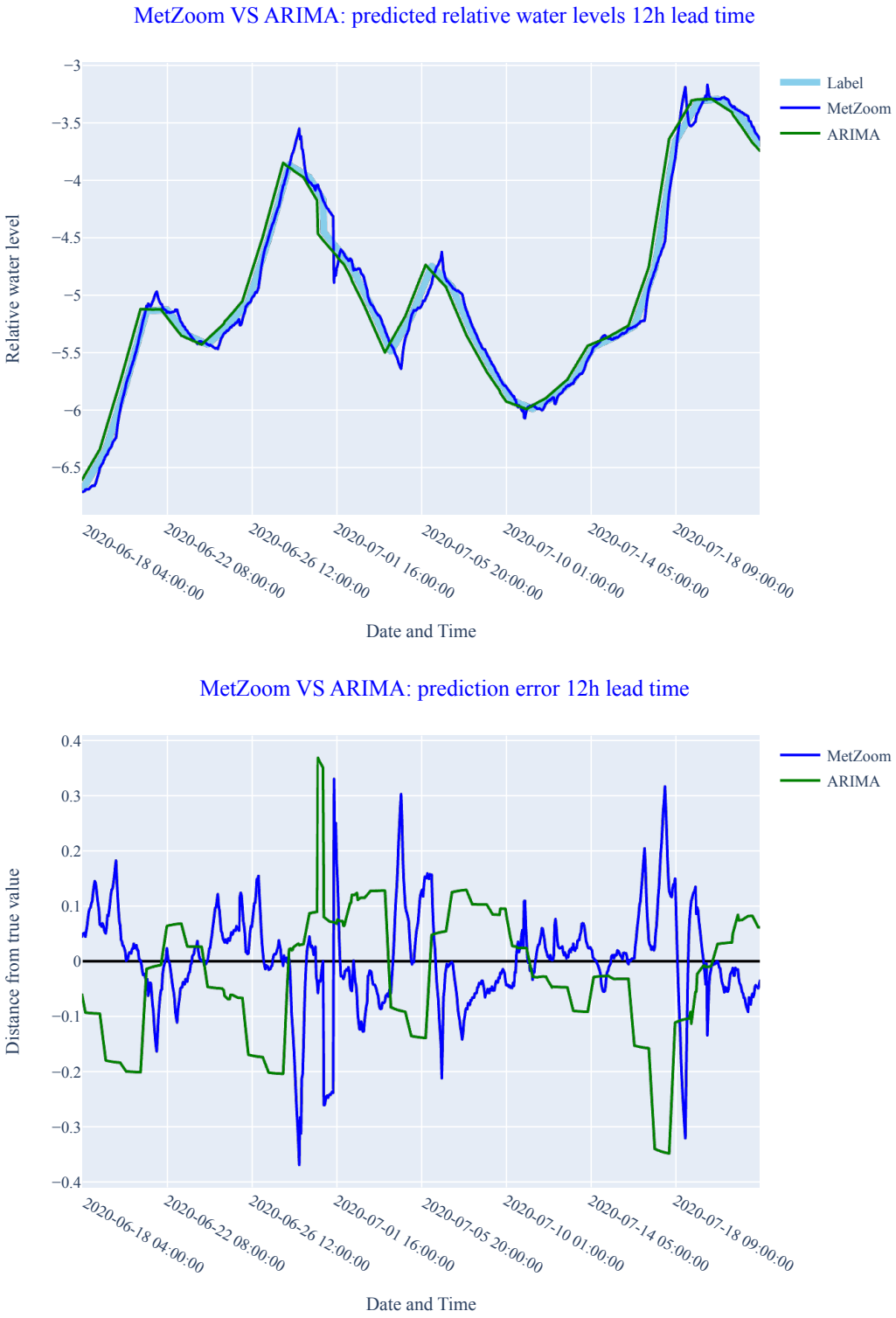

FIGURE 7.10: The top graph shows the predictions by MetZoom and the ARIMA model. The bottom graph shows the errors in the same time frame.

The ARIMA model, as explained in Chapter 2, Section 2.1.2, predicts a much smoother change in water level than MetZoom. It is more accurate than MetZoom with lead times shorter than 7 hours. When forecasting 12 hours ahead the implementation of the ARIMA model generally predicts smaller changes than MetZoom.

The shortcomings of the implementation of the ARIMA model is discussed in Chapter 8. It is believed that with more time and hyperparameter tuning, a better ARIMA model could have been implemented.

7.4.4 MetZoom VS. LSTM

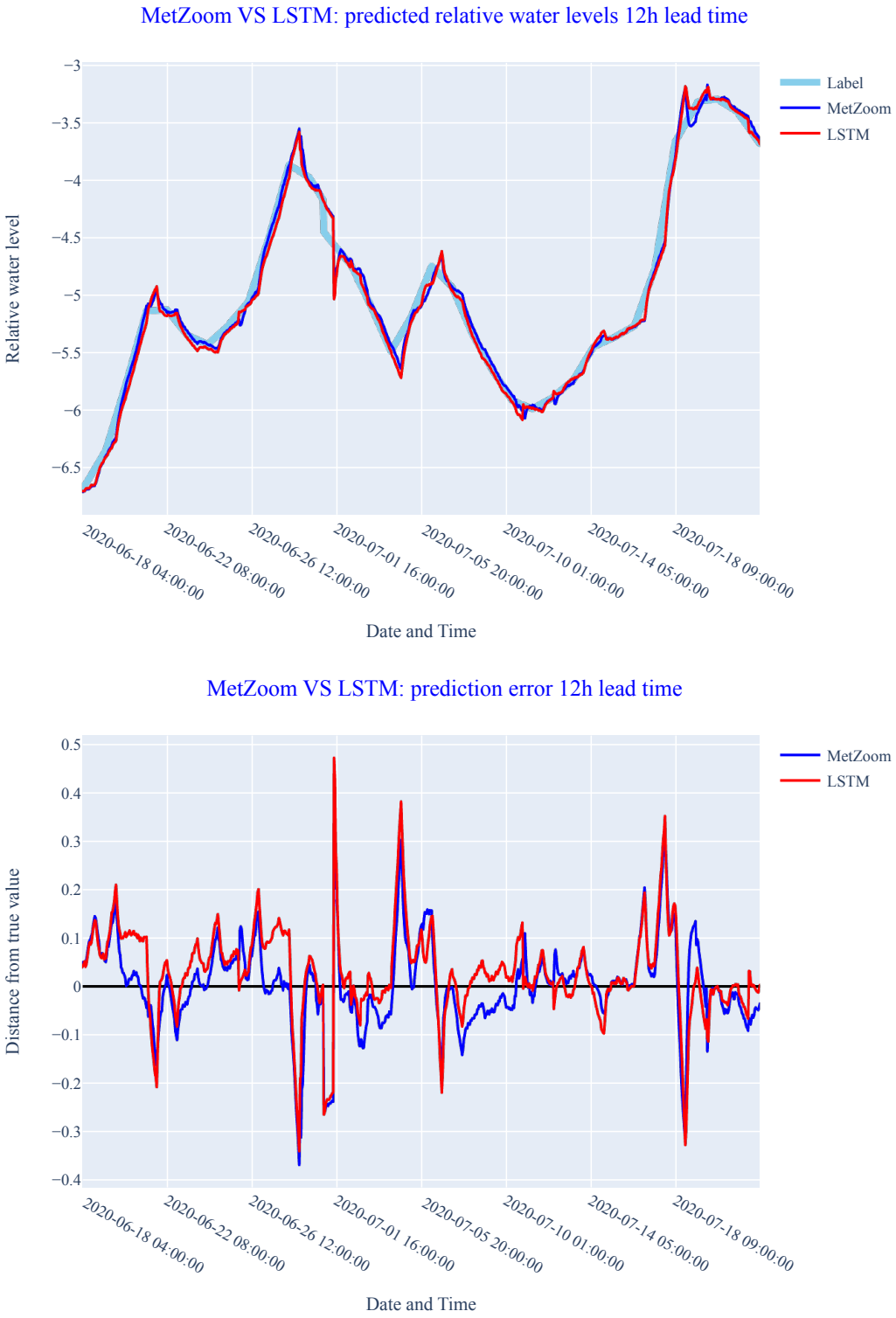

FIGURE 7.11: The top graph shows the predictions by MetZoom and the LSTM model. The bottom graph shows the errors in the same time frame.

The LSTM model used for comparison, as explained in Chapter 3, Section 3.4.1, is a more simple LSTM than the one used in MetZoom. This network omits the radar images, and hence features extracted from these, from both input and predictions. Hence this comparison creates an idea of the efficiency and added value of the radar images.

The results in Figure 7.11 show the predictions and errors of both models in in the first center snippet, when forecasting 12 hours ahead of time. Generally the LSTM seems to over-predict the changes in turning points and therefore does not follow the actual water level as closely as MetZoom. The results here, and in the metrics in Figure 7.8 show that in comparison to a LSTM the radar images provide added information that helps the model slightly.

Chapter 8

Evaluation

Some of the setbacks experienced while working on the thesis came from the most promising ideas, and were essential for learning what was required to achieve acceptable results. This chapter will reflect on some of the challenges, thoughts and ideas that were central through this process.

The final architecture and model, although inspired by others, is created from scratch with the purpose of the thesis.

8.1 Reviewing the data

8.1.1 Error margin

When predicting changes in water level only through added precipitation as a measure of inflow and valve opening as a measure of outflow many natural causes are omitted. For added accuracy on the outflow one could account for natural evaporation through temperatures, and for inflow include snow melting and inflow from water retention in the surrounding area. The water retention in the surrounding area is partially learned though the data that is included. With respect to inflow, these added variables are learned inherently from the dataset as freezing and melting conditions happen on a yearly basis. As the network is trained on the entirety of the time series, including seasonal changes, but without exact information on snow melting and

temperatures it introduces an error margin the network most likely cannot surpass. One of the largest errors is possibly due to over-focusing on precipitation and not focusing more on snow melting, as even days and weeks without any precipitation can have massive inflow. The RNN has to adapt for these unforeseen changes, and assume that melting will continue at a set rate, which is rarely the case in nature. In retrospect it would have been a good idea to focus the network on seasons where the snow melting is not a high factor, such as the late summer to early fall. In this case MetZoom could have performed better, and possibly improve its competitiveness in comparison to the other models.

8.1.2 Sparse radar images

One of the major disadvantages when predicting precipitation levels with neural networks is the natural absence of it. Throughout the time series, there are more hours (i.e radar images) with little-to-no recognizable precipitation. When batch-training this leads to the network predicting lower levels of precipitation due to loss becoming small when simply predicting no precipitation at all. To overcome this challenge a large portion of time was spent on different loss functions, methods for thresholding input and output images to focus on high precipitation and masking or normalizing the radar images. Eventually the idea of element-wise multiplying the amounts of precipitation with a confidence level of the precipitations movement led to focusing more on *where* the precipitation will fall after some time, rather than the amount of precipitation.

8.1.3 Interpolation on missing timestamps

Due to the different time series discussed in Chapter 4, some interpolation was necessary. When interpolating such an amount of data, it is important to

recognize the loss of true information representation as discussed in Chapter 2. Although in some cases interpolation can resemble the true flow of natural data, actual weather data is less smooth than interpolated data. The chaotic nature of weather becomes smoothed and interferes with the precision of the predictions according to the *actual true values* in a complete and saturated dataset.

8.2 Objectives and achievements

8.2.1 Motivation and tasks

The goal for the thesis was to investigate whether a neural network could accurately predict the relative water level in Helgedalsvatnet using a time series of relevant information. The time series includes previous water levels, valve opening, precipitation and radar images. The radar images, which cover an area with several precipitation measurement stations, reflect the rate of precipitation. These values are quite transferrable to precipitation volume, and so this became a *preliminary* for the greater goal of predicting the water level. Getting great results of precipitation predictions from the radar images was difficult, as the predictions of radar images were smoothed out and did not capture heavy precipitation. A goal was set to get predictions that were followed the trend of the true data, such that this trend could transfer to the relative water level predictions.

When this was achieved the work was focused on predicting the changes in relative water level. After deciding the final architecture, and getting some results it became apparent that it is difficult to measure the exact performance of a model when the data is linearly interpolated and the natural differences and occurrences are skewed towards a more linear level. Still, it is reasonable to expect that the constant and linear contributions will dominate on this time

scale, regardless of how accurate and frequent measurements are available. As the data was structured in hourly intervals the changes are very slow and hence a naive baseline model which simply predicts no change is quite accurate. Therefore only a small improvement on such predictions was accepted. Predicting the slope (i.e the time-derivative) of the changes in water level is not a trivial task, and there is room for more work in this area. Further improvement on the performance could have been possible with a better understanding of which natural mechanisms and sources of error dominate the data. Technical improvements could have been made either by more development and training or a more extensive hyperparameter search with the existing model, but was stagnated by the necessity to compare the existing results to other models, as done in Chapter 7.

One of the main goals was reviewing whether the inclusion of radar images could benefit in predicting changes in water levels, and at this point it is difficult to conclude on the actual benefit of the radar image addition.

8.2.2 Reviewing the comparison to other models

There were many difficulties when implementing the ARIMA model. Understanding the nature of the model took a lot of time, and the initial results were very poor. Significant time was spent on improving the ARIMA model, and without any prior knowledge or experience with such models there is definitely room for improvements.

The LSTM that was created for comparison excludes the radar images from the data it uses, and is a measure of how much the radar images add to the performance. The LSTM went through several stages of optimization and is as thoroughly trained as MetZoom, although a much simpler network.

Although the data is easily regressed due to the linear interpolation, and that is likely the cause of the baseline models accuracy, it would have been an

achievement if the results of MetZoom was far better than that of the baseline model and others in comparison to it. To achieve such results, some future work is needed.

8.3 Reviewing conceptual choices

8.3.1 Using 3D Convolutions

One of the primary ideas revolved around 3D convolutions as explained in 3. In the case of sequence prediction through 3d convolutions, passing a block of images (as opposed to one image with several input channels) opens the possibility for improvement in the networks geo-spatial awareness through time. 3D Convolutions have been acknowledged as powerful tools for learning volumetric data. The idea of applying such operations to a sequence of input images seemed promising, as the aim is to capture the movement of precipitation over time. An example is seen in Figure 8.1

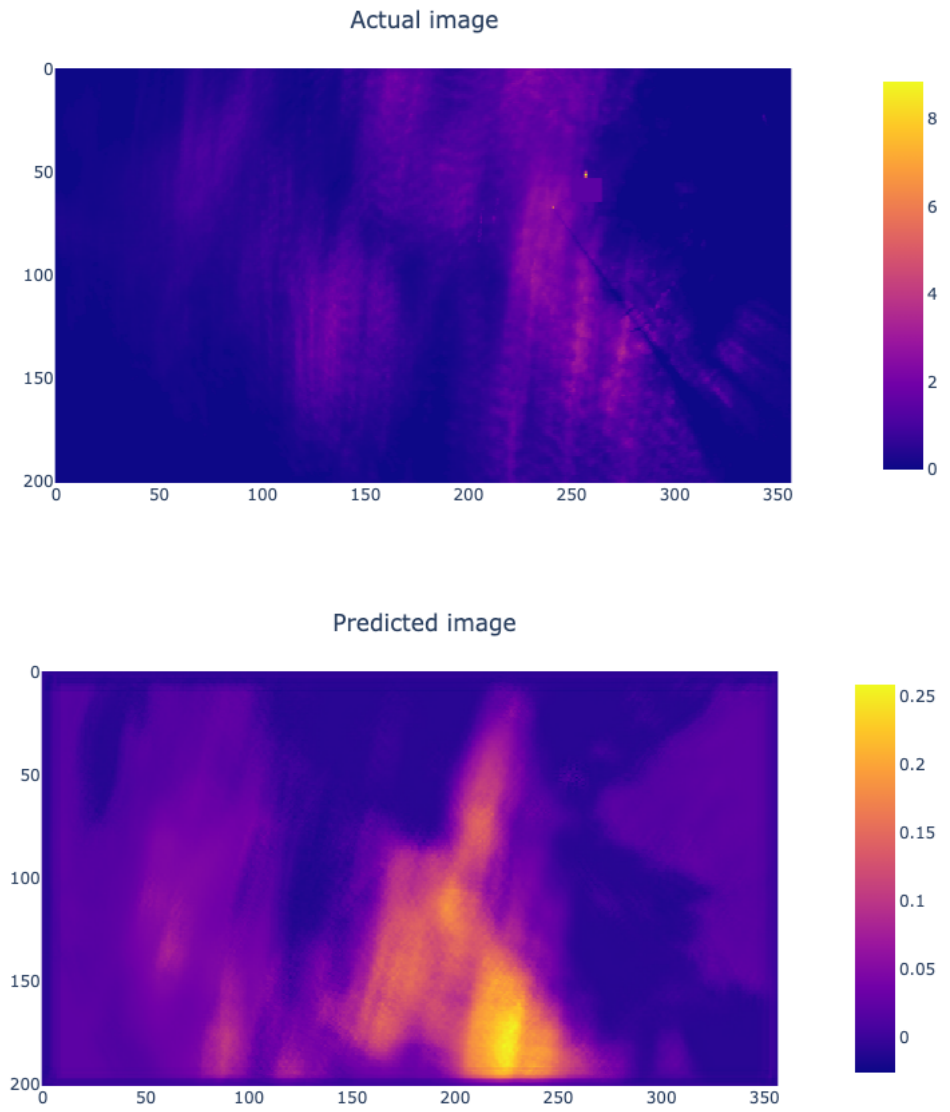

FIGURE 8.1: The actual label and the predicted image one hour into the future by the 3D convolution architecture. The units are similar to the radar images in Chapter 4

8.3.2 U-Net architecture

U-Net, introduced by Ronneberger, Fischer, and Brox 2015, has proven efficient on image segmentation with high localization accuracy, and was therefore promising in regards to capturing local precipitation in the radar images. While evaluating U-Net for the task of precipitation forecasting the radar images were the only relevant source of information, and the initial idea was to

use such an architecture to predict one future block of radar images via 3D convolution up- and down-scaling with residual connections.

In the simplest form, the testing revolved around an input block containing a sequence of radar images, and outputting a single radar image corresponding to the radar image following the sequence (i.e. 1 label image). Although a simple idea and somewhat ambitious, it seemed feasible to transform U-Net from a segmentation network to a many-to-one 3D residual U-Net.

The network, shaped like U-Net, down-scaled a batch of images (as a 3D block) while sending residual connections to the up-scaling part of the network. The final layer of the network was a transpose convolution (as described Section 3.3.4) which for for testing purposes only had 1 output channel depth-wise, representing 1 output image.

Through extensive over-fitting the network managed to replicate the whereabouts of precipitation, but still did not quite capture the amount.

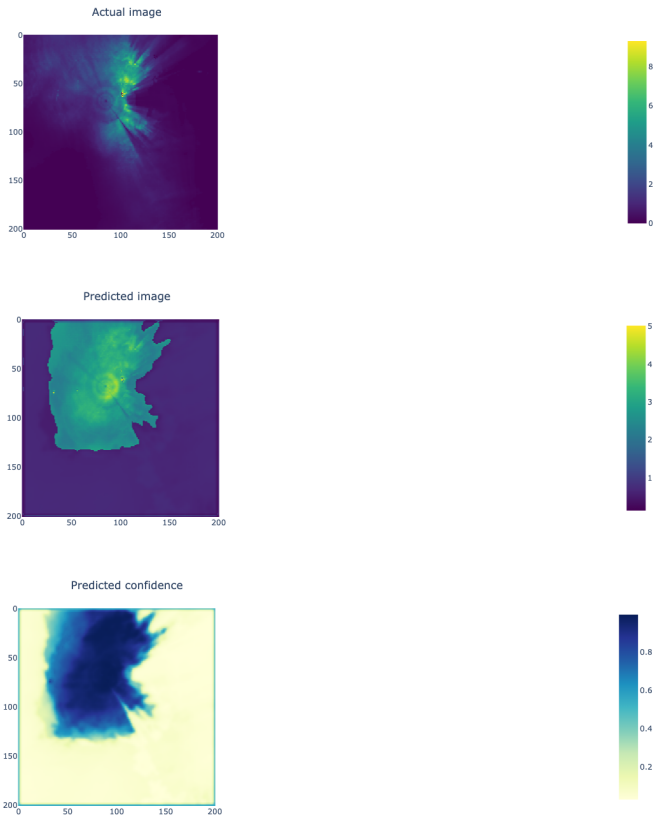

FIGURE 8.2: The actual label, the predicted image, and the predicted confidence by the U-net architecture.

A final addition was made to the architecture, which was a second output channel passed through a sigmoid activation function, which represented the confidence of the whereabouts of precipitation - allowing the original channel to solely represent amounts. The "confidence image" (i.e. probability of precipitation) could then be used as a mask, allowing the network only to calculate errors where the confidence was above some set threshold, the result is seen in Figure 8.2.

The combination of confidence prediction and value regression was promising. This realization lead to creating a new architecture which potentially could manage both tasks.

8.3.3 The conception of MetZoom

The mentioned combination of a confidence in the movement and amounts of precipitation led to the conception of MetZoom, the name of the final architecture in the thesis. When removing the 3D convolutions a new method of representing the temporal difference between the input radar images was required. The solution became adding images at every layer, at predefined blocks. These blocks were meant to represent the movement of the precipitation, whilst including some information on amounts. The amounts of precipitation are carried through every layer.

Throughout testing this theory, and getting better results on predicting future radar images and precipitation rates, the idea of zooming into an important area was also sparked. This idea, although implicit through convolutional operations, was inspired by Google's MetNet-2 (Espeholt et al., 2021).

MetZooms CNNs efficiency and accuracy within shorter time periods was deemed good enough to be used as a tool and its output accurate enough for combination with a potential LSTM architecture. MetZoom is robust and adaptable. It's variability in input and output sequence length provided possibility for testing different output lengths and an LSTMs accuracy on predicted relative water level for Helgedalsvatnet.

Originally the implementation used the output of MetZoom as input for the LSTMs block, but this caused some questions as predictions and true data was intertwined for a second future prediction. As such the LSTMs was altered such that the input of the entirety of the network was a sequence of length 24 including all of the data. The radar images of the input sequence are processed through the CNN which predicts a future set of images.

To properly draw information from the images a method for feature extraction was required, and hence ResNet-18 (He et al., 2016) was introduced.

All of the images are processed alongside each other, which results in a cohesive sequence which is then split in sections.

The resulting method uses the input images as a means of predicting the relative water level in Helgedalsvatnet, and the future (predicted) images as a mean to adjust this water level in two fully-connected layers. At this point the study revolved mainly around training and testing the network with different sequence lengths and kernel sizes.

Chapter 9

Conclusion

This chapter will draw a conclusion on the study, and the achieved results, as well as suggestions for future work.

9.1 Conclusive remarks

The work done in this thesis was aimed at answering whether a neural network could be used for predicting inflow to a hydropower reservoir, and if the inclusion of precipitation reflecting radar images could benefit such a network. The results show that the answer to both of these research questions is *yes*. To what extent is another question. A neural network such as MetZoom can predict the changes in relative water levels, representing the inflow/outflow relationship, but at a level of accuracy that is not groundbreaking in comparison to other more naive and simple models. If a more complete and saturated dataset was available, with actual measured data for every timestamp, this comparison could be different in the favour of MetZoom, as it is more precise on changes in water levels at longer lead times than the compared models. With an ANN with an architecture similar to MetZoom one has the possibility to extend the model to include even more observable variables, such as temperature, snow cover and even the topology of the geographical area. These extensions could further skew the advantage towards ANNs.

Although the current architecture of MetZoom may not be the perfect approach, it shows that ANNs can be used to incorporate many different sources of data, extending a models capabilities.

9.2 What's next?

There are many interesting factors in prediction of inflow. One of the main interests and ideas which was reviewed for inclusion early on was factoring in the geographical topology of the surrounding area, to further capture snow melting and water retention in the surrounding area. Including a 3D model or added information on the topology could provide better predictions at longer lead times, as this information adds to the long term changes of water levels. A variable which was available in the data provided but omitted was the temperature. Adding this variable to the equation could further enhance the prediction of snow melting and water retention in the surrounding area. Note that for this to work long term data on the snow cover for the area is needed, as well as a possibly very long memory model.

In this thesis, only one reservoir was used for prediction. Inclusion of other reservoirs and learning the relationship between the connections of these reservoirs could be a possible research direction. With interconnected tunnels and natural waterways between reservoirs it could be interesting to learn how the inflow/outflow relationship of one reservoir affects other connected reservoirs, whilst still predicting the changes in water levels for the included reservoirs.

The results show that there is more smoothness in the traditional ARIMA model, and a regular LSTM performs only slightly worse than MetZoom. Further studying the accuracy of precipitation downfall captured through radar could improve a model which incorporates radar images. The naive baseline model also performed fairly well, and as mentioned is likely due to

how the data was structured and interpolated. A future work could research how to structure a time series without smoothing the data as much.

Furthermore there are not many works on prediction of inflow through radar images. Through the results of this work it is apparent that these images can benefit a network, and there are most likely better ways to employ these images in a network than what has been done in this thesis. The results of radar image predictions also show a smoothed version of the corresponding labels. This is likely due to loss of information concerning the amount of precipitation when the images are fed sequentially and masked with a sigmoid activation to capture the confidence of movement. A research direction relevant to the work in this thesis could involve efficiently capturing both the amount and the movement of precipitation at a more precise level. As the results of precipitation predictions show, the smoothed predicted radar images impact the extracted precipitation, which confirms the models inaccuracy with heavy precipitation.

A thorough research on how to extract correct and better information from radar images could be very interesting, as they have proven to be a possible tool for predicting water inflow.

Bibliography

Ayzel, Georgy, Tobias Scheffer, and Maik Heistermann (2020). "RainNet v1. 0: a convolutional neural network for radar-based precipitation nowcasting". In: *Geoscientific Model Development* 13.6, pp. 2631–2644.

Bengio, Yoshua, Ian Goodfellow, and Aaron Courville (2017a). *Deep learning*. Vol. 1. MIT press Cambridge, MA, USA.

— (2017b). *Deep learning*. Vol. 1. MIT press Cambridge, MA, USA.

Benjamin, Stanley G et al. (2016). "A North American hourly assimilation and model forecast cycle: The Rapid Refresh". In: *Monthly Weather Review* 144.4, pp. 1669–1694.

Buizza, Roberto (2002). "Chaos and weather prediction January 2000". In: *European Centre for Medium-Range Weather Meteorological Training Course Lecture Series ECMWF*.

Coulibaly, Paulin, François Anctil, and Bernard Bobée (2000). "Daily reservoir inflow forecasting using artificial neural networks with stopped training approach". In: *Journal of Hydrology* 230.3-4, pp. 244–257.

Dumoulin, Vincent and Francesco Visin (2016). "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285*.

Espeholt, Lasse et al. (2021). "Skillful Twelve Hour Precipitation Forecasts using Large Context Neural Networks". In: *arXiv preprint arXiv:2111.07470*.

Gholamalinezhad, Hossein and Hossein Khosravi (2020). "Pooling methods in deep neural networks, a review". In: *arXiv preprint arXiv:2009.07485*.

Gu, Jiuxiang et al. (2018). "Recent advances in convolutional neural networks". In: *Pattern Recognition* 77, pp. 354–377.

Haykin, Simon and N Network (1999). "A comprehensive foundation". In: *Neural networks* 2.1999, pp. 41–44.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hyndman, Rob J and George Athanasopoulos (2018). *Forecasting: principles and practice*. OTexts.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR, pp. 448–456.

Janocha, Katarzyna and Wojciech Marian Czarnecki (2017). "On loss functions for deep neural networks [..]" In: *arXiv preprint arXiv:1702.05659*.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Krenker, Andrej, Janez Bešter, and Andrej Kos (2011). "Introduction to the artificial neural networks". In: *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech*, pp. 1–18.

Kumar, Ashutosh et al. (2020). "Convcast: An embedded convolutional LSTM based architecture for precipitation nowcasting using satellite data". In: *Plos one* 15.3, e0230114.

Maass, Wolfgang (1997). "Networks of spiking neurons: The third generation of neural network models". In: *Neural Networks* 10.9, pp. 1659–1671. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7. URL: https://www.sciencedirect.com/science/article/pii/S0893608097000117.

Netrapalli, Praneeth (2019). "Stochastic gradient descent and its variants in machine learning". In: *Journal of the Indian Institute of Science* 99.2, pp. 201–213.

O'Shea, Keiron and Ryan Nash (2015). "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458*.

Roberts, Brett et al. (2019). "The High Resolution Ensemble Forecast (HREF) system: [..]" In.

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

Samuel, A. L. (1959). "Some studies in machine learning using the game of checkers". In: *IBM Journal of Research and Development* 44.1.2, pp. 206–226. DOI: 10.1147/rd.441.0206.

Sharma, Sagar, Simone Sharma, and Anidhya Athaiya (2017). "Activation functions in neural networks". In: *towards data science* 6.12, pp. 310–316.

Sønderby, Casper Kaae et al. (2020). "MetNet: A Neural Weather Model for Precipitation Forecasting". In: *Submission to journal*. URL: https://arxiv.org/abs/2003.12140.

Werbos, P.J. (1990). "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10, pp. 1550–1560. DOI: 10.1109/5.58337.

Zhang, Aston et al. (2021). "Dive into deep learning". In: *arXiv preprint arXiv:2106.11342*.