KTH ROYAL INSTITUTE
OF TECHNOLOGY

DEGREE PROJECT IN ENGINEERING PHYSICS

SECOND CYCLE, 30 CREDITS

Precipitation Nowcasting using Deep Neural Networks

VALTER FALLENIUS

Stockholm, Sweden 2022

TRITA – SCI-GRU 2022:034

Abstract

Deep neural networks (DNNs) based on satellite and radar data have shown promising results for precipitation nowcasting, beating physical models and optical flow for time horizons up to 8 hours. "MetNet", developed by Google AI, is a 225 million parameter DNN combining three different types of architectures that was trained on satellite and radar data over the United States. They claim to be the first machine learning model to outperform physical models at such a scale. In this work, we implemented a similar but simplified model trained on radar-only Swedish data, with the aim to perform precipitation nowcasting for up to 2 hours into the future. Furthermore, we compare the model to another, simpler model that omits the spatial aggregator of the DNN architecture which is a state-of-the-art vision transformer. Our results show that, although the adopted training dataset was too small to prevent overfitting, the model is still able to outperform the persistence benchmark for lead times longer than 30 minutes with a threshold of 0.2mm/h precipitation. Our simplified model, perhaps unsurprisingly, is outperformed by MetNet because of having too few training data samples or variances in the models' implementation. We show, nonetheless, that the adopted spatial aggregator fulfills a vital role as expected, aggregating global information into spatial and temporal contexts.

Due to the limitations imposed by the reduced size of the model, we cannot, unfortunately, draw definitive conclusions on whether a radar-only model could yield similar forecast skills as MetNet. To improve on these results, more training data is certainly needed. This would require that more robust computation resources are available, but pre-training the model on a larger dataset — or even implementing a model that takes in different geographical locations for training — can naturally lead to significant improvements in the predictions.

Sammanfattning

Djupa neurala nätverk (DNN) baserade på satellit och radar data har gett bra resultat för korta nederbördsprognoser och kan slå fysikaliska modeller och optical flow för prognoser upp till 8 timmar i framtiden. “MetNet” är ett 225 million DNN publicerat av Google som kombinerar tre olika typer av djupa arkitekturer, det är tränat på satellit och radar data över USA och är enligt dom den första maskininlärningsmodellen som presterar bättre än fysikaliska modeller. I denna uppsats har vi konstruerat en modell som liknar deras på ett nedskalat problem. Vi har färre parametrar, lägre upplöst data, endast 2 timmar prognostisering och använder bara radar data över Sverige för att träna modellen. Vi använder F1-score för att evaluera modellens prestanda och jämför prognosen mot persistens som referens. Vidare undersöker vi en mindre komplicerad modell där den tredje arkitekturen inte används för att se vilken roll vision transformern har. Våra resultat visar att datasetet vi tränat på är för litet och modellen överanpassas men modellen lyckas ändå slå persistens referensen för prognoser 30–120 minuter när en 0.2mm/h regn tröskel tillämpas. Resultaten är sämre än MetNet av Google och vi kan inte dra några slutsatser huruvida en modell med endast radar-data skulle kunna ge liknande resultat eller inte, eftersom modellen inte tränats till dess fulla potential. Vi visar och att den tredje arkitekturen, vision transformern, är en viktig del av nätverket och aggregerar global information till lokala kontexter över tid och rum. För att förbättra våra resultat skulle vi pröva att låta modellen träna på det amerikanska datasetet använt av Google och implementera en modell vars input varierar geografisk position.

Preface

I would first like to thank my two supervisors at SMHI, Heiner Körnich and Paulo Medeiros, for supporting me at each step of the way and taking the time to discuss the meteorological problem with me. I would also like to thank my supervisor at KTH, Ricardo Vinuesa, who helped me get resource allocation for training my model on the Berzelius supercomputer. Finally I would like to thank Jacob Bieker and Jack Kelly at Open Climate Fix for providing the open source code skeleton, helping me iron out bugs, discussing different aspects of the network and showing interest in my project.

Contents

Chapter 1

Introduction

In recent years, deep learning has revolutionized many industries like facial recognition [1], autonomous vehicles [2] and natural language processing [3]. Deep neural networks (DNNs) require large amounts of data with correlations (to some degree) between input and output as well as a model that can learn a representation of the input data and apply necessary transformations (or "transforms") to produce a realistic output. There are few industries with as much data as meteorology, and the patterns present in weather data are a consequence of the governing physical laws. Therefore numerical weather prediction (NWP) is a reasonable task for deep learning. The application of DNNs as a weather forecasting technique would facilitate computations on a day-to-day basis since physical models require much larger computational load [4, 5]. Better weather forecasts save lives through early warning systems [6], creates revenue [7] and climate savings in sectors like energy [8], aviation [9] and agriculture through smarter planning [10], and help to mitigate economic impacts of weather related disasters like hurricanes [11].

For physical models the computational load grows with the resolution [12], and this creates a trade-off between accuracy and the time it takes to generate a forecast. Today's NWP forecasts can take multiple hours to generate [5] and this is a problem especially for very short-range time horizons, namely nowcasting. Nowcasting is the practice of very short-range forecasts up to 8 hours into the future, if the forecast takes 2 hours this is certainly a problem. A strength of DNNs is that even if they take many days to train, this is just a one-time cost and once it is trained it would be able to generate a forecast in a few seconds [4].

Another potential strength of DNNs for weather forecasting is the fact that it is output-optimized. Our physical models' mathematical formulations are based on the current understanding of physics and might be imprecise at the given resolution. There might for example be a special shaped mountain range that under certain conditions produces some effects on the local wind fields that our physical models fail to capture. Or, another example of effects our physical model's might omit, are microphysical effects with regards to man-made aerosols that can have a strong

influence on cloud albedo and microscopic droplet formations [13]. These effects, if substantial enough, could be detected by a DNN since it is not bound by physical laws but rather it detects and replicates the underlying patterns. If there is some pattern in the data, the model will learn it. But this also leads us to a weakness of DNNs, namely its lack of physical constraints, which can lead to blurry forecasts [14, 15] since physical constraints restrict the output space. The lack of physicality, in itself, might not matter so much in the nowcasting task since we are mostly interested in accurate forecasts but in some cases we desire our models to be bound by physical constraints. One such example would be in climate research where we want to predict how the climate will change in the next 100 years, here a DNN might give powerful insights or warnings, however DNNs are in nature more or less like black boxes which limits any physical conclusions that might be drawn.

Chapter 2

Background

2.1 Seven Governing Equations

Weather forecasting has been one of the greatest challenges since its inception. With endless degrees of freedom and chaotic interactions in the atmosphere, it is safe to say that NWPs are hard. NWP has been around for over a century. First suggested at the turn of the 20th century by Abbe [16] and Bjerknes [17], they proposed that weather was a physical system governed by a set of equations of motion that could be treated as an initial value problem. At that time there were no computers, very few routine observations of the atmosphere, no guarantees of Newtonian determinism or any knowledge about the predictability of weather [12]. These major obstacles and unknowns, however, did not preclude the practice of NWP from being initiated.

NWP models use numerical discretizations to partition the atmosphere into spatial and temporal representations of different atmospheric variables. Bjerknes [17] stated for the first time in 1904 that there are seven governing equations for the evolution of the atmospheric state:

1. The three Navier-Stokes equations of momentum

2. The continuity equation (mass conservation)

3. The equation of state of the atmosphere (eg. the ideal gas law)

4. The two fundamental laws of thermodynamics regarding energy and entropy

The equations are paired with seven independent variables: temperature, pressure, density, humidity and three wind speed variables. Together, these determine the evolution of the state of the atmosphere.

2.2 Operational NWP Models Today

After Bjerkness [17] and Abbe [16] it took some time until the first attempts at predicting the weather numerically took place. The first numerical prediction was performed by Richardson in 1922 [18]. Although this attempt was not particularly successful, it was later shown in 2006 by Lynch [19] that this was due to a failure to apply smoothing techniques on the data to avoid non-physical initial conditions. Until the 1950s weather forecasting was done by meteorologists through subjective interpretations of synoptic charts. However it became clear that these empirical techniques were inefficient and with the emergence of computers the first operational NWPs were born [20]. Today at the Swedish Meteorological and Hydrological Institute (SMHI), NWP models operate with a horizontal resolution of about 2.5km and 65 vertical levels, which means the models are using about 4.6 million grid points in Sweden. The numerical models propagate the initial conditions in time using discrete time steps of about 40s, for a total simulated time of up to 66 hours into the future [5]. One can, in principle, expect more accurate results by decreasing the time steps and grid spacings used in the numerical simulations. The most obvious route to enable running models with finer time and space resolutions – thereby improving the forecasting skills[1] of such models – is thus increasing the computational power. Indeed, this is what we have seen in the last 50 years how the incredible development of faster computers has gone hand in hand with better weather forecasts [21].

Today, NWP models are run on some of the largest computers in the world [22, 23] with vast quantities of data. Most NWP models nowadays are run using ensembles of multiple forecasts. Each forecast in an ensemble is called an ensemble member. At SMHI the current operational system "MEPS" is run with 5 members. Producing forecasts for up to 66 hours into the future, these ensemble runs are started every hour and take about 1 hours to finish [5]. Here we can see an indication of just how powerful a DNN with similar forecasting skill could be: they would, in principle, be able to generate a forecast in mere seconds using present data instead of 1 hours old data. In recent years the rise of DNNs has shown promising results for NWP [24], especially for precipitation forecasting [4, 25], and have the potential to change the way operational NWP is done today. In contrast to the current physical models a deep learning model would be able to generate a forecast in seconds and this could lead to a shift where computational power is no longer the bottleneck but instead higher resolution observations might have the potential to increase the forecasting skill for short-range forecasts.

Precipitation nowcasting is today done via physical models that advect[2] precipitation fields by time stepping them in the wind speed direction. There are multiple methods for precipitation nowcasting today; some are physics-based and rely on humidity, pressure, wind speed and advection, while others, like optical flow [26], is based on non-physical assumptions, namely:

[1] This refers to any measure of performance of the forecasting weather models.
[2] Advection is the transportation of substances in a fluid.

1. The observed total rain intensity is constant in time
2. The velocity field is smooth

Since these assumptions do not always hold true for real weather, there can be variations in both rain intensity and the velocity field, and therefore optical flow does not always perform well. Indeed, especially for medium and long-range forecasting (12h+), the physical models tend to do much better than optical flow.

2.3 Weather Forecasting in Sweden

SMHI is a government agency and research facility towards weather forecasting, hydrology and climate models. Along with other institutes from Norway, Finland and Estonia they form a meteorological cooperation called MetCoOp. The MetCoOp's current operational system "MEPS" is an ensemble prediction system and has been operational since 2016. In order to estimate the uncertainty of weather forecasts, most NWP models today run multiple simulations, so-called members, of a perturbed initial state and compare different simulations to get an uncertainty estimation. To generate these members one could launch all of them at the time of the forecast and then compare the results, but this approach leads to a very unbalanced usage of computer resources. So, in order to maximize the number of members available for the forecast, a continuous approach has been used since 2020 where, instead of generating 10 members every 6 hours, 5 members are generated every hour [5].

MEPS uses forecasts from the European Centre for Medium-Range Weather Forecasts (ECMWF) as boundary conditions for all their members and perturbs them to generate different initial and boundary conditions [5]. In order to assert smoothness at the boundaries they overlap the edges and smooth them out. By using the continuous approach MEPS can improve a number of shortcomings of the previous implementation. The first improvement is seen in a decrease of the spinup time, this is the time it takes for the model to reach a balanced state.[3] Shorter spinup times thus render short-term forecasts more valuable. The second improvement is reduced otherwise excessive initial winds and precipitation, this is a consequence of reducing the spinup time [5].

2.4 Societal Impact

Better weather forecasts have the potential to save lives, mitigate economic impact of weather-related disasters and increase revenue in sectors like energy and agriculture. According to a report by the U.S. Global Change Research Program, climate

[3]Here we avoid using the words "equilibrium state" since the atmosphere is never in an equilibrium. A balanced state means that the non-equilibrium is physical and all large-scale forces are within "normal" limits.

change has in the last decades led to more frequent and intense extreme weather events [27] which drives the need for better weather forecasting techniques. In August 2014 in Malmö, Sweden, one such extreme weather event occurred when over 100 mm rain fell in the course of a few hours, which is equivalent to two normal summer months total rainfall. This led to problems such as, for instance, flooding in 2,300 basements, 3,000 damaged cars among others, and an estimated total cost of at least 600 million SEK in reparations [28]. Here, better weather forecasting would certainly allow security services to react faster, better plan their efforts and target their efforts where the most rain was bound to fall, which could have mitigated the economic and societal repercussions of such an event. The impact of better weather forecasts — and in particular machine learning solutions — can also be seen in the energy sector. For example, Molinder et al. [29] present a machine learning method to predict icing issues on wind turbines. Icing forms on wind turbines in colder weather and can decrease energy production by up to 80% [30] or even reduce the lifetime of a turbine [31]. By training directly on energy production data, with weather forecasts as input, the model's probabilistic output adds value to the forecast when compared to physical deterministic models [29]. Another high-impact machine learning model on weather data is the solar nowcasting technology by Open Climate Fix [32]. The consumers expect a constant supply of electricity and so the power companies need to have reliable power reserves — typically generators run on nonrenewable resources — that they can switch on within seconds when the weather is cloudy. These power generators can take anything from minutes to hours to initiate, which makes it necessary to have some of them running in reserve even when they are not needed. By nowcasting the weather and solar power supply, a third-party "Impact Forecast" estimated this new technology by Open Climate Fix could potentially enable a 1.3 million tons of CO2 climate impact per year in Europe by disburdening the power generators when they are not needed [33].

2.5 Climate Change and Weather Forecasting

In the last decades, global warming projections have shaken our society and, since the Paris agreement in 2015, the world leaders have committed towards limiting global warming to below 2°C. According to Vinuesa et al. [34] advances in artificial intelligence could have a substantial impact on 134 out of 169 targets of the 17 Sustainable Development Goals [35]. In this section, we discuss the impacts of climate change on the weather forecasting task and some advances in machine learning that can have a positive impact on our transition towards a more sustainable way of living. In section 2.4, we discussed the socioeconomic impacts of accurate weather forecasts and argued that they are substantial. It is therefore very important to consider the impacts of climate change on weather forecasting when discussing the socioeconomic effects of global warming.

Weather is chaotic in nature. This means that, even with a good understanding of the governing equations and an accurate description of initial state and boundary conditions, it is safe to say that there will always be uncertainties in the produced weather forecasts — the more uncertain the farther into the future one tries to perform a forecast for. The nature of the uncertainties in the various types of forecasts is, however, not always the same. Some weather systems are "easier" to predict and with less degree of uncertainty. For precipitation forecasting it is commonly the case that low-pressure systems with large-scale cloud fronts are easier to predict than small-scale convective events [36]. Convective flows in the atmosphere are short-lived, with time scales of ∼hours, in contrast with low-pressure fronts which are longer-lived, with time scales of ∼days. Convective flows are caused by water that evaporates and rises to higher altitudes where it condenses again and releases energy. This cycle of water and vapor is hard to capture through radar imaging since vapor is invisible to radars, the short time scales decrease predictability and increases the demands for higher defined atmospheric models. In Scher and Messori [37] they investigate the impact of climate change on the weather forecasting task and conclude that the predictability of the atmosphere could significantly change with a warmer climate. They show that some fields like temperature and pressure will be easier to predict, especially in the northern hemisphere, due to smaller temperature gradients between the equator and poles thus reducing the available potential energy for cyclonic development. They also conclude that precipitation forecasting is likely to become more difficult to perform due to an increase of high-pressure — and thus probably convective — systems.

2.6 Deep Neural Networks for Precipitation Forecasting

DNNs are collections of neurons organized in a deep architecture of multiple layers. The neurons receive some input stimulus, where stimulus refers to the data the neuron is stimulated by when analyzing it, e.g. a radar image, and then the neurons calculate the output through linear weighted sums and non-linear activation functions. Activation functions are used to create an easily differentiable loss function that can be used when backpropagating the weights with e.g. stochastic gradient descent (SGD). Read more about DNNs, backpropagation and activation functions in [38]. Through an iterative process of offering inputs to the network, allowing it to guess on a candidate output and telling it to what degree the guess was correct, we can tune the weights (using, e.g., SGD) to better predict the desired output. If a network has too many parameters compared with the amount of input and output data, then one runs into the issue of overfitting. Overfitting occurs when the network memorizes an input and output pair instead of learning the underlying patterns. This is why a lot of data is needed for DNNs in order to force them to learn the patterns of cause and effect present in the system rather

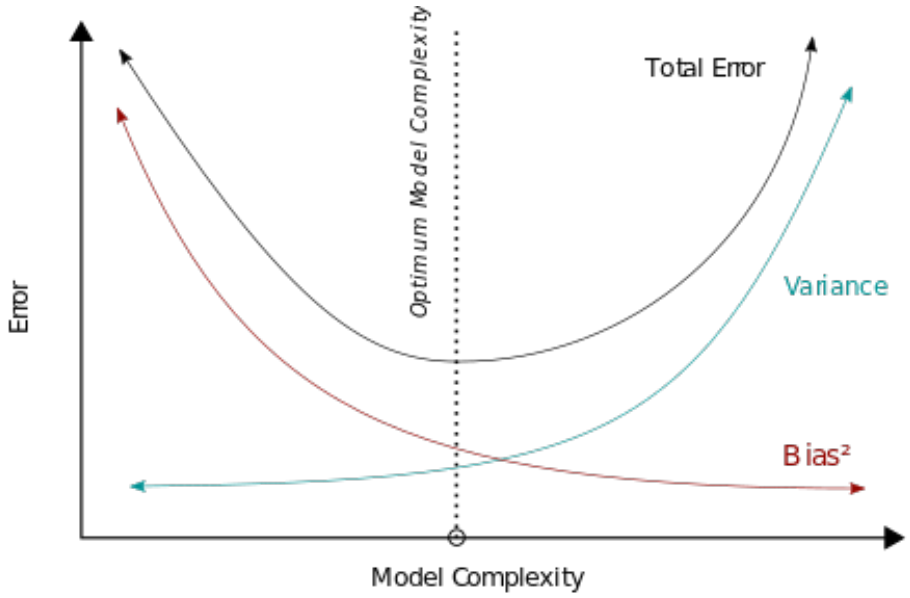

Figure 2.1. Optimal model with respect to variance and bias.

than memorizing particular events. A common way of describing it is through the analysis of the model's bias–variance tradeoff [39], where we need to optimize the network so that it does not have a high bias nor a high variance, see Figure 2.1. Figure 2.1 illustrates how bias and variance is connected to the model complexity. High bias arises when the model has too low complexity and oversimplifies the data, and high variance occurs when the model pays too much attention to the training data and fails to predict unseen data. See an illustration of a high bias and a high variance model in Figure 2.2.

There are a lot of different types of deep learning architectures that are good at solving different problems [40]. For weather forecasting, convolutional neural networks (CNNs) have shown to be promising [4, 14, 25]. CNNs are especially strong for image related tasks like detecting edges or more complex patterns which can translate into pressure gradients or cloud fronts in the weather forecasting task. Another commonly used architecture for time series are recurrent neural networks (RNNs) [41], which take in sequential data as input to measure changes in time. This method can also be employed in the weather forecasting task for its, among others, motion detection capabilities [42] to quantify cloud speeds. Recently Sønderby et al. [4] published the DNN called "MetNet" and claimed to be the first deep Learning model to beat state-of-the-art physical models in the precipitation nowcasting task at the large scale, see a plot of MetNet's performance in Figure 2.3. MetNet combines three different DNN architectures: the previously mentioned

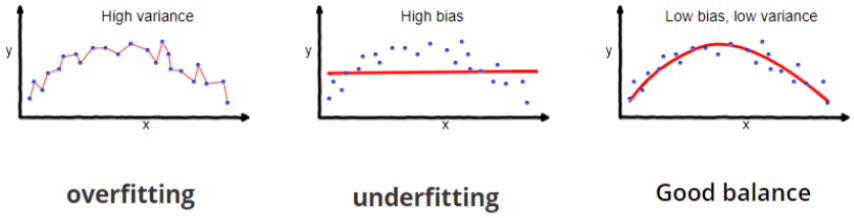

Figure 2.2. Illustration of too high variance (left), too high bias (middle) and optimal bias and variance (right). Figure extracted from article by Seema Singh [39].

CNN and RNN, along with a new type of architecture called a *vision transformer*, see Figure 2.4. As we implement a similar architecture to MetNet in this work, we present a more in-depth discussion of it in Chapter 4.

Another approach for precipitation nowcasting with DNNs are *deep generative models*. One such model, used for instance by Ravuri et al. [25], is the so-called Generative Adversarial Network (GAN). GANs use a setup consisting of two different networks that are pitched against each other: one network (the generator) tries to "fake" reality. Here reality refers to something that is feasible and realistic, e.g. if the GAN tries to fake an image of a bird then it might be regarded as feasible or realistic if a human would not be able to distinguish the generated bird from a picture of a real bird. While another network (the discriminator) that is trained to tell reality from the fake. In a "zero-sum game" the networks compete, the generator gets better at mimicking reality and the discriminator gets better at telling the real image from the fake one. Ravuri et al. suggest that GANs can perform better than other DNNs for precipitation nowcasting, as, while DNNs produce blurry forecasts [14, 15] for longer lead times, GANs are generally able to produce sharper images, as the discriminator component help identifying unrealistic features as artifacts [43]. They evaluated their model using both qualitative and quantitative measures, more about this in Section 4.4.

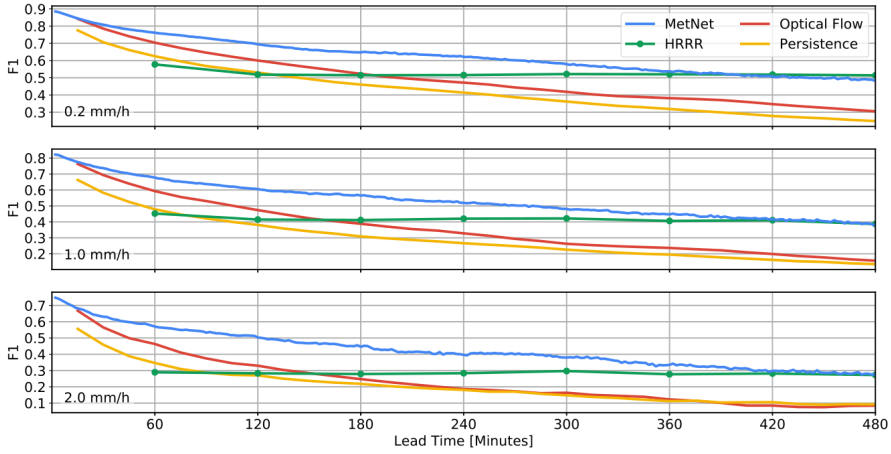

Figure 2.3. F1-score (see Section 4.4) of MetNet *vs* benchmarks optical flow, physical model High-Resolution Rapid Refresh ("HRRR") and persistence. Figure extracted from Sønderby et al. [4] with permission from the authors.

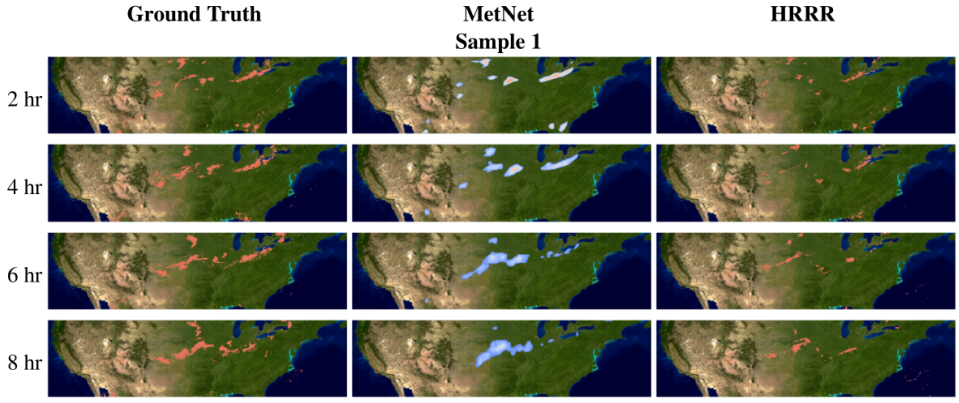

Figure 2.4. Ground truth (left), MetNet prediction (middle), physical model (right). Figure extracted from Sønderby et al. [4] with permission from the authors.

Chapter 3

Data

In this chapter we discuss the data we used as input and output, as well as how we processed it for use with our model. A challenge we faced working with the dataset was that it is very large ∼500GB and this led to issues when loading into the RAM-memory. We tackled this by only loading a few samples from memory at a time, which slows down the training but not to a significant degree. Another challenge working with large data for machine learning is to create data with unit-variance and zero mean, and to do this we had to sum all $525,000$ data samples and calculate the means of our different fields.

3.1 Precipitation data

Precipitation data was collected from 10 radar towers in Sweden that measure the reflectivity of the atmosphere in dBZ. Since normal air does not reflect the wavelength of radar very effectively but water droplets do, these can be converted into 2D composite maps of precipitation. One important shortcoming of these radar fields is that they do not measure humidity, as water vapor does not reflect radar pulses, more on this in Chapter 6. To convert dBZ into precipitation rate R $[\frac{\text{mm}}{\text{s}}]$ the following formula is used [44]:

$$R = \left[\frac{10^{\frac{\text{dBZ}}{10}}}{200}\right]^{\frac{5}{8}} \tag{3.1}$$

Precipitation maps are extracted for the entirety of Sweden every 5 minutes.[1] In this project we have used 5 years of data, accumulated between 2017–2021, which yields ∼$525,000$ precipitation maps. The radar fields have a spatial resolution of 2km with 881×454 pixels. For our purposes having the data in a square grid makes sense, assuming that spatial input west to east is equally important as north

[1] Barring events when radar towers are down for maintenance or otherwise blocked.

to south. To avoid areas where the radars are out of range the southern half of Sweden was used in a 448×448 square, see Figure 3.1.

The radar data files come with two different fields: raw and corrected. The raw data is purely based on the reflectivity of the atmosphere and the corrected data is processed[2] to better match observational data from weather stations. In order to achieve closer correspondence to the real state of the atmosphere, we have opted to use the corrected data fields.

To speed up training — and so that computations can be done in parallel without reducing the resolution of the data too much — we divide the input images into 8 channels for the network to process, one lower resolution image and one higher resolution center crop image that both are transformed using a space-to-depth transformation [3] into eight 112×112 images as illustrated in Figure 3.2.

3.2 Elevation, longitude and latitude

To make it easier for our model to learn the patterns of the weather we help it by giving it additional data fields in 112×112 image encodings. Three channels are used to represent the elevation, longitude and latitude of each pixel. The motivation for including an elevation map is that it will be able to account for effects on precipitation caused by terrain shifts, for example mountainous areas might act as a blockade for clouds to pass or they can lift air parcels to condensation levels so it rains. To reduce the noise in the data a mean-pooling filter[4] is used to smooth the elevation map. See a plot of longitude, latitude and elevation map in Figure 3.3.

Longitude and latitude data are included, mostly to reproduce the data structure used by Google in their model [4]. While Google used a model where the input area moved around, our input is stationary. Since the input of MetNet covers a much larger geographical area, longitude and latitude encodings are likely more important for their application than for ours, but to eliminate any doubt we include these two data fields in our model nonetheless. In general, latitude allows the model to account for effects regarding the Coriolis force and longitude helps the model to tweak for local time differences; this feature seems more important when different time zones are covered.

[2]The precipitation field is increased when there is less precipitation than the corresponding observed ground data. Also, clutter and noise in the radar images are removed.

[3]Space-to-depth is done by rearranging the pixels of a 2D-tensor of any size into a 3D-tensor of lower resolution images without losing any information. For example we can space-to-depth a 100×100 pixels image into a $25 \times 25 \times 4$ tensor as a linear mapping. Note here that for our 8 channels we start with $448 \times 448 \approx 200,000$ pixels but end up with $112 \times 112 \approx 100,000$ so half of the information is lost.

[4]Mean-pooling as described in this link: `https://paperswithcode.com/method/average-pooling`.

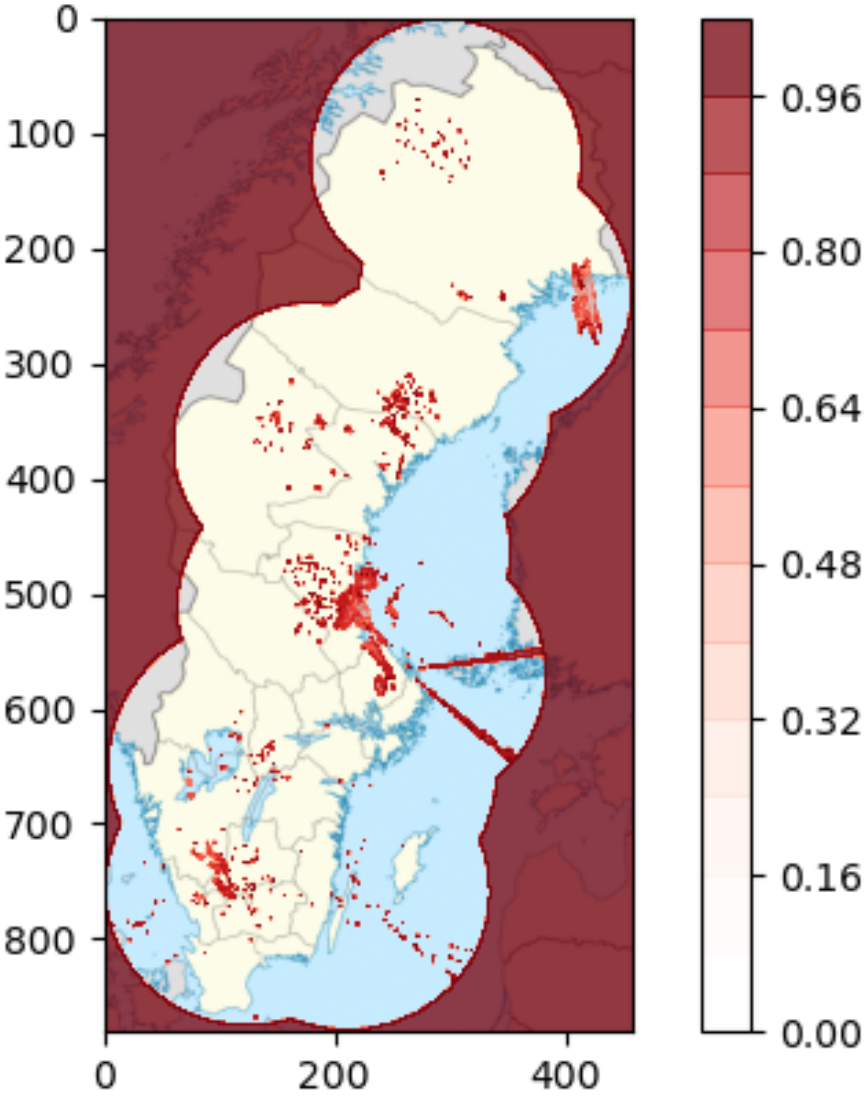

Figure 3.1. Radar data over Sweden (in dBZ) normalized to the range [0,1]. Note that, in this plot, the visible ranges of the radar and that values close to 1 means high precipitation and values of 1 mean "no data available". Before feeding the input to our network we transform all "no data available" to 0 instead.

3.3 Periodical encodings

To account for the day/night cycle and seasonal effects on precipitation four additional channels are introduced. Figure 3.4 shows the mean rain rate for the hour of the day and the month of the year. We can see that precipitation is more common during the summer months than in winter, as well as during the afternoon hours when compared with night hours. Each input radar map has a date and time that is mapped to four sinusoidal functions to encode for the natural periodicity of time, eg. 23:45 should be encoded in such a way that it is similar to 01:00. These are then placed in 4 channels of size 112×112 to replicate the structure of the other input channels. Here are the four encodings:

$$\alpha = \sin(\frac{2\pi[\text{day}]}{365}) \tag{3.2}$$

$$\beta = \cos(\frac{2\pi[\text{day}]}{365}) \tag{3.3}$$

$$\gamma = \sin(\frac{2\pi[\text{minute}]}{60 \times 24}) \tag{3.4}$$

$$\theta = \cos(\frac{2\pi[\text{minute}]}{60 \times 24}) \tag{3.5}$$

3.4 Lead time encodings

The network is a multi functional neural network in terms of which lead time it predicts, meaning we use the same model for generating a 15 minute forecast as a 60 minute forecast. To make this possible we need to encode the desired lead time and we do this in the input layer by adding other 8 input channels that we one-hot encode[5] with the lead time. So for a given lead time we have 7 channels with 112×112 zeros and one channel with 112×112 ones.

3.5 Temporal Concatenation

The network is a type of RNN and takes sequences as input. These sequences consist of the data fields discussed earlier in this chapter and are separated by 15 minutes. These make up tensors with shape $23 \times 112 \times 112$ which are then concatenated into a sequence along a new axis. Like the original model MetNet [4] the sequence consists of seven input separated temporally by 15 minutes, that is one $23 \times 112 \times 112$ tensor describing the state at T-90min, one at T-75min, and so on down to T-0min.

[5] A one-hot encoding is a way to represent categorical data. For example a set with 3 classes $\{dog, cat, bat\}$ can be represented with three different vectors $[1, 0, 0] = dog$, $[0, 1, 0] = cat$ and $[0, 0, 1] = bat$.

3.6 Preprocessing

The radar channels have values between −30 dBZ and 72 dBZ. The range of the radar towers is limited and sometimes the radar towers are under maintenance or the line of sight is blocked. This is represented in the dataset with a value of 72dBZ, but to not confuse the network to think that 72dBZ is heavy rain we replace all pixels with exactly 72 dBZ with 0dBZ. We normalize this data using the same method as the authors of MetNet [4] with two transformations as seen below. All the data was then normalized channel wise with zero-mean and unit variance.

1. $\log(\text{dBZ} + 0.01)/4$
2. Replace NaNs with 0

Since rainfall is a rare occurrence we sort out radar data when there is little rainfall, and because we want to train our model to predict where it rains, it needs to be trained on data with rain. Training the model on a data sample without any rain does not let the model learn how precipitation flows, instead it learns that usually there is no precipitation, which is an issue we have to constantly battle when dealing with tail-heavy datasets in deep learning. To sort out when there is rainfall we count the number of pixels in the prediction zone with more than 0.2mm/h precipitation and only sample lead times during training with more than 5 rainy pixels, the same threshold was used by MetNet [C. Sønderby, personal communication, 5 March, 2022]. This results in keeping about 15% of the most rainy data with respect to the prediction zone displayed in Figure 3.2. See comparison between the ground truth rain rates when we keep 100% *vs* 15% of the data in Figure 3.5. When we plot the quotients of rain rate occurrence of 100% vs 15% of the data we get Figure 3.6, here we see that all bins associated with rain increase by a factor of about 4.5.

3.7 Data partitioning

All the data fields described in this chapter are combined into pairs of input and output tensors of shape $7 \times 23 \times 112 \times 112$ and $8 \times 128 \times 28 \times 28$ respectively. We split the data into temporally non-overlapping data samples with 16 training samples, followed by 2 validation samples and 2 test samples. This yields a partitioning of 80% training data, 10% validation data and 10% test data. However, since we do a qualitative sieve over the training set, as explained in section 3.6, in the previous section we lose approximately 85% of the training samples since most samples do not contain any rain. The reason for not splitting the validation or test datasets with the same sieve is to make sure our model is able to predict "no rain" when it should.

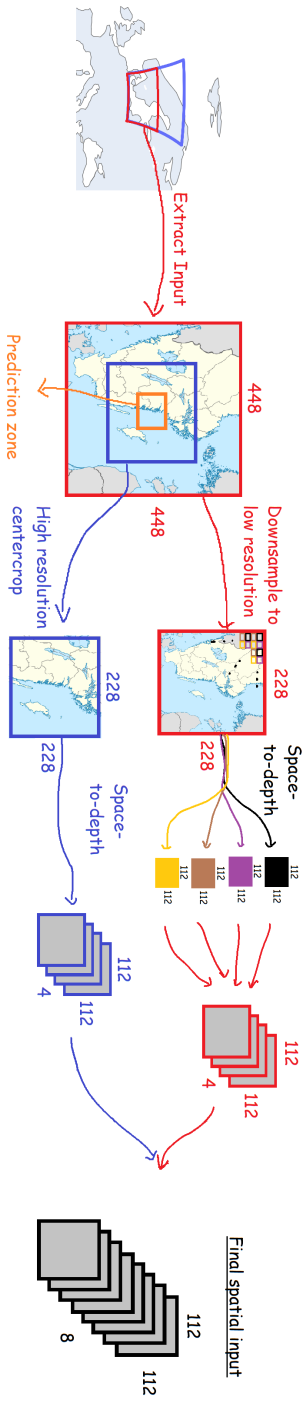

Figure 3.2. Input patch for radar data and illustration of space-to-depth transformation.

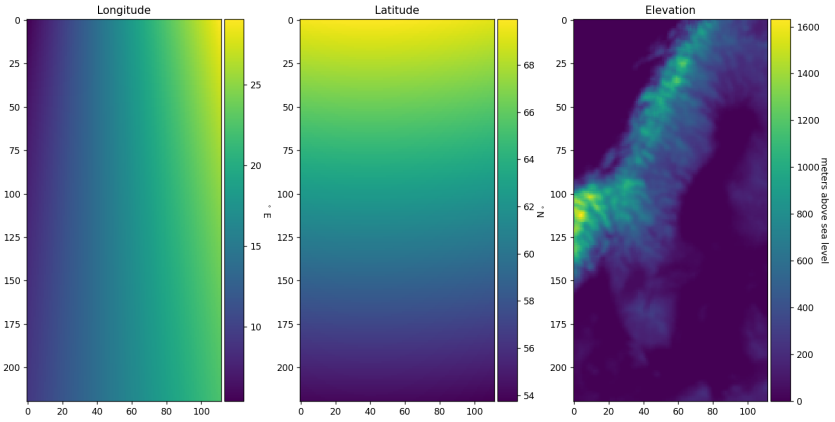

Figure 3.3. Longitude, latitude and elevation maps over the Swedish dataset

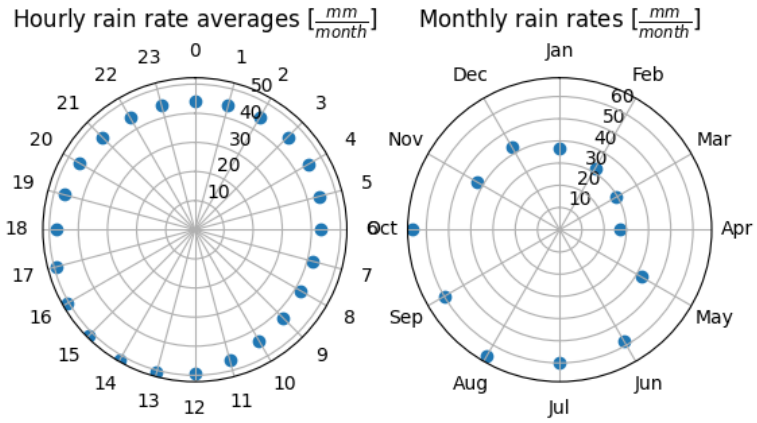

Figure 3.4. Day/night rain rate averages (left) and monthly precipitation averages (right). Averaged over the Swedish radar dataset over the same 5 years, 2017–2021, we used to train and test the model.

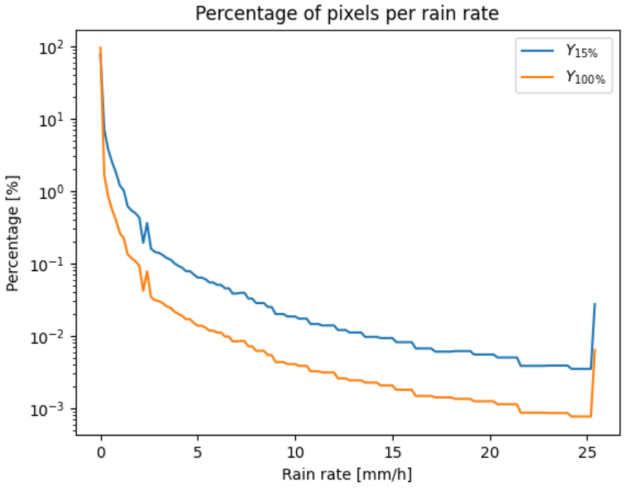

Figure 3.5. Rain distribution after sampling 15% heaviest rainfall events.

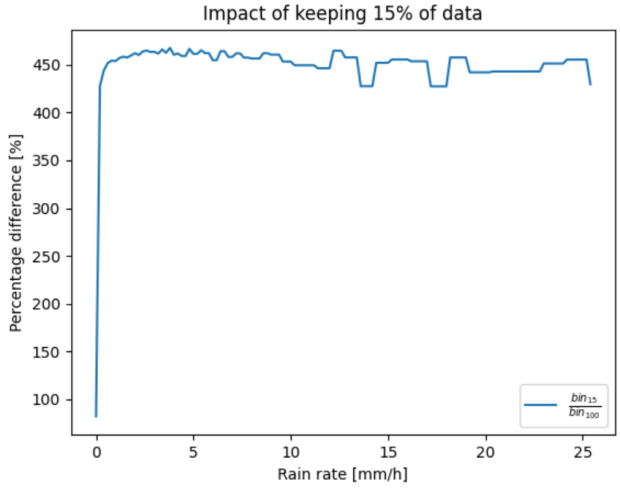

Figure 3.6. Rain distribution after sampling 15% heaviest rainfall events.

Chapter 4

Methods

In this chapter we describe the architecture of the model, what loss function we used, how we parallelized the code to speed up training, the hyperparameter tuning and finally we discuss the benchmarks that were used to evaluate the model's performance. We trained the model on the Berzelius cluster [45] in Sweden using a single NVIDIA® DGX-A100 compute node with 8 GPUs. After a total of 208 GPU-h and 250 epochs without improving validation loss the training was terminated and reinitialized with a smaller learning rate to attempt a better fine-tuning.

4.1 Model Architecture

4.1.1 Background

In this thesis the model architecture is inspired by Google Brain's network MetNet [4], and an open source implementation of MetNet by Bieker and Kelly [32] was used for the code skeleton. The model is tasked with precipitation forecasting for 120 minutes in the future at lead time increments of 15 minutes and it is trained on radar data exclusively (as opposed to MetNet that also included satellite images). The network is trained and validated on 5 years of radar composites with categorical cross entropy loss [46].

MetNet [4] is a DNN for precipitation nowcasting with 225 million parameters that combines three deep learning architectures: a CNN, a convolutional long short-term memory network (ConvLSTM) [47] and an axial attention transformer [48]. The CNN acts as a spatial downsampler by giving the data a more compact representation, the ConvLSTM takes in the downsampled temporally separated data and extracts temporal attributes in the data like cloud velocities and precipitation decay and the axial attention transformer aggregates the global information into the prediction zone. We have used a very similar network architecture but much

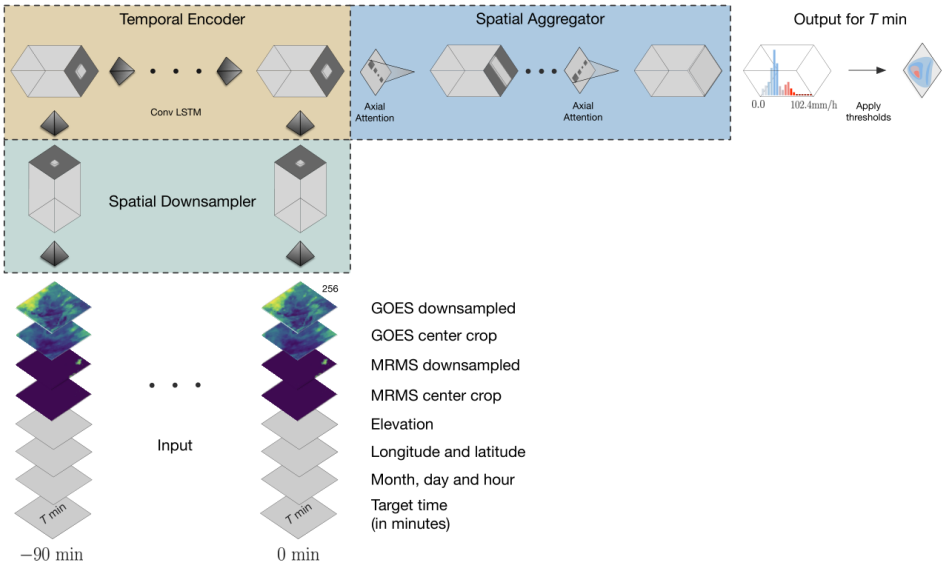

Figure 4.1. MetNet architecture and input patch. Figure extracted from Sønderby et al. [4] with permission from the authors.

smaller at around 7.3 million parameters and with the exception of the ConvLSTM where we instead have used the simpler convolutional gated recurrent unit (ConvGRU) unit that only has one memory-state.

4.1.2 Input and output

The inputs are a 448×448 pixels radar images along with other data fields, described in Chapter 3. With 2km resolution, it covers a 896km × 896km area of southern Sweden. Since our input area is stationary, the amount of data that can be extracted from one radar image is limited but also facilitates the implementation. The output area is a small 28 × 28 pixels, or 56km × 56km square in the middle of the input patch and needs to be smaller in order to have any predictability. For example a cloud situated 360km outside the target patch might move to it under the right conditions at 6 hours lead time. Assuming clouds move with a maximum velocity of ∼60km/h , the maximum time horizon is about ∼6 hours which is why the model is limited to 300 minutes lead time. However a model with 300 minute lead time proved to be too ambitious given the size of our dataset and we therefore opted to use a 120 minute lead time which means the size of our input area might be excessive but should not impede the performance of the network.

4.1.3 Spatial Downsampler

CNNs use "nearest neighbor interactions" for neuron activation schemes to learn to recognize different patterns. For image recognition they can identify edges or other simple patterns characterized by their kernel. Kernels are sliding weight matrices that we use for the convolution operations, they dictate how the neighbors are weighted with a neuron. Convolutional layers, when stacked in deep networks, can identify more complex patterns like facial expressions [1]. The first CNN predecessor was proposed by Kunihiko Fukushima [49] in 1980 and was called the *neocognitron*. The neocognitron was inspired by the work done by Hubel & Wiesel [50] on animal cells in the visual cortex and how they responded to different light stimuli. In 1989 LeCun et al. [51] established a framework that laid the foundation for the modern CNN but it was not until 2012 that CNNs revolutionized the computer vision industry.

In 2012 Krizhevsky et al. [52] created a CNN that reduced the error of the state-of-the-art models available at the time for image classification by more than 10%. The authors introduced a new approach to training a neural network using GPUs and parallelization techniques which allowed them to train the model on a significantly larger amount of data than what was previously possible. After this breakthrough CNNs have been very successful in various computer vision tasks like facial recognition [1], text recognition [53] and other image analysis tasks and is today the most widely used deep learning architecture[54].

In our model the spatial downsampler is tasked with compressing the spatial data. The input is represented by tensors of shape $23 \times 112 \times 112$ and the 23 channels consist of 8 radar channels, 4 channels for temporal encoding, and 3 channels for elevation, longitude and latitude and finally 8 channels for encoding the desired lead time. The downsampler consists of four convolutional layers, three batch normalization layers and two max pooling operation,[1] this yields a final output tensor of shape $256 \times 28 \times 28$. This is done for seven samples with 15 minute increments to act as input for the LSTM at T-90min, T-75min, ..., T-0min.

4.1.4 Temporal Encoder

The original MetNet model [4] uses a ConvLSTM network which is a type of RNN that sequentially analyzes input and uses two hidden states to store information through the sequence, a long and short-term memory. For long sequences in vanilla RNNs, information in the beginning of a sequence is lost because of the vanishing gradients problem [55]. To address this issue LSTMs add a pipeline for the long-term memory to directly propagate through to the next state and updates it with regards to the short-term memory state through three gates: the input gate, the forget gate and the output gate. The input gate decides what information from the current input and the previous short-term memory to store in the long-term

[1] We use max pooling with a 2×2 kernel, explained in this article `https://computerscience wiki.org/index.php/Max-pooling_/_Pooling`.

memory. The forget gate also takes the current input and the short-term memory to decide what information in the long-term memory to delete. Finally the output gate combines the long and short-term memories with the current input to calculate the new short-term memory state.

In a ConvLSTM the inputs are processed using convolutional operations as described in the previous section. Unlike MetNet we are using a ConvGRU that is very similar to a ConvLSTM. It has two instead of three gates and only a single hidden state. It operates without a long-term memory and addresses the vanishing gradient problem differently. A ConvGRU consists of an update gate and a reset gate. The update gate is analogous to the output gate of an LSTM: it determines what information to pass on to the next state from the previous state and this allows the GRU to pass all the information along and thus deals with the vanishing gradients problem. The reset gate is analogous to the input and forget gate of an LSTM: it decides what information from the previous state to neglect and updates the hidden state accordingly.

For this study, the data from the downsampler is a sequence of tensors with shape $256 \times 28 \times 28$: 256 channels and 28×28 spatial fields. First the tensor corresponding to T-90min is fed to the ConvGRU, then the T-75min and so on to present time T. Here the ConvGRU has generated a representation of the current weather state not only where it is cloudy, but also where all the clouds are moving, where clouds are falling down as rain and other time related differences in the input. This is then analyzed by the final piece in the network, the spatial aggregator.

4.1.5 Spatial Aggregator

The spatial aggregator in MetNet is a type of vision "transformer" network that uses 8 axial self-attention blocks [48]. Transformers are the new state-of-the-art for natural language processing tasks since the famous paper "Attention Is All You Need" from 2017 by Vaswani et al. [56]. The reader is referred to that reference for more information about the attention mechanism. In short, transformers take sequences as their input data just like RNNs but do not necessarily look at the input sequentially. Instead, transformers attend to sets and use encodings to decide where in the sequence a value lies. This makes it possible to parallelize transformers — which is impossible with RNNs.

Transformers have shown promising results in many computer vision tasks with the pioneering vision transformer ViT by Google [57] and the Swin Transformer by Liu et al. [58] excelling in object detection and semantic segmentation. The Swin Transformer architecture has also been successfully applied on video motion recognition [59] and medical image segmentation [60]. CNNs have dominated the scene for computer vision tasks for the past 10 years for their exceptional skill at capturing local semantic information [61], however, since each node only looks at input from the nearest neighbors, they are less effective for learning long-range interactions. Transformers are significantly more computationally heavy since the parameters grow with the square of input size, for images, this means the input

side length to the power of four, while CNNs parameters are constant independent of side length. This is ultimately what led to the development of the special type of transformer that uses axial self-attention.

In axial self-attention the input image is divided into strips and the transformer only attends to its corresponding row or column one at a time, instead of attending to the entire picture. This means the number of parameters can be kept to a minimum and at the same time global context can be achieved after just 2 layers of row and column attention. MetNet uses 8 axial self-attention layers swapping between column and row attention in the spatial aggregator, and we have opted to use the same amount. However it is clear that the implementation of the spatial aggregator in MetNet differs from the implementation used in the open source model we have applied. When initiating a model with the same hyperparameters as in MetNet our model only has 18.2 million parameters, thus differing from the 225 million parameter network by Google.

In our model the spatial aggregator is tasked with aggregating information from global spatial context to each pixel in the prediction zone. The previous layers analyze local phenomena like cloud boundaries, precipitation decay, elevation blockades and cloud velocities, and the axial self-attention blocks are tasked with "making sense" of the data into a probabilistic categorical prediction. After the attention layer we have used a 1×1 CNN layer to transform the data into the desired shape.

4.2 Loss

SGD [62] was used to train the model with a categorical cross entropy loss function [46]. The output layer of the model is fed through a softmax operation [63] along the categorical rain bins, which yields a probability distribution of the different rain rate bins. There are 128 rain bins with 0.2mm/h increments from 0 up to 25.4mm/h, and all rain rates greater than 25.4mm/h are grouped in the final bin. The ground truth pixels are one-hot encoded vectors corresponding to which rain rate that pixel has, and the categorical cross entropy is calculated using the following formula:

$$\text{Loss} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{4.1}$$

where n is the number of bins, $\hat{y}$ is the prediction for a pixel, and y is the ground truth. Since the ground truth is one-hot encoded, the sum in Equation 4.1 reduces down to a single term, the higher the predicted probability for this term the lower the loss will be. This loss is then used for the back propagation [64] with SGD to tune the weights of the model. Furthermore we use gradient descent with momentum as this has been shown to greatly speed up convergence [65] but needs to be tuned properly in order to not reduce the performance of the model. The gradient

for the current batch, ∇_i, is a function of the calculated gradient, ∇, and the gradient of the previous step, ∇_{i-1}, as described by the following recursive formula:

$$\nabla_i = \nabla + \alpha \nabla_{i-1} \tag{4.2}$$

where α is our momentum hyperparameter and usually is set to $\alpha \approx 0.9$. The training and validation loss are calculated differently for a given epoch. During training we introduce some randomness since we only sample a single lead time for each training sample in a batch, but for the validation loss we want to reduce the randomness as much as possible and therefore we calculate the validation loss over all training samples.

4.2.1 Parallelization and Implementation

The code skeleton for the network was provided by the open source non-profit organization Open Climate Fix [32] and was implemented using the library PyTorch [66]. Their implementation had to the best of our knowledge not been rigorously tested or trained before our implementation. A major bottleneck was found in the implementation of the axial attention layer, which was quickly addressed after a short communication with the authors.[2] This master thesis was allocated 1080 GPU-h/month at the Berzelius cluster in Sweden [45]. To be able to use the allocated resources efficiently the code skeleton had to be parallelized and we chose to use the PyTorch Lightning library [67], which is a research framework that facilitates parallelization of DNNs implemented with PyTorch. By parallelizing the code we could train on multiple GPUs at the same time which enabled us to significantly reduce the time it takes to train a network.

The parallelization scheme used is a so-called *distributed data parallel*, an implementation where each GPU receives a copy of the model. Then for each step in a training epoch the GPUs receive different input data samples and individually process them and calculate the gradients using SGD. The gradients are then accumulated and averaged over all parallelized GPUs to update the model's weights. To put it in perspective, our network trained on 8 GPUs for 36 hours, but without parallelization the same network would take upwards of 12 days to train. Additionally the Python library and framework Weights & Biases [68] was used to track gradients, GPU usage and training, as well as validation loss during training.

4.3 Hyperparameter tuning

Because of the model's complexity it takes a very long time to train and we were not able to perform an extensive parameter search for all hyperparameters. Instead most of the time was spent finding hyperparameters that "did the job". We tried

[2] Read more about this issue report at: `https://github.com/openclimatefix/metnet/issues/22`.

varying hyperparameters like learning rate, number of lead times, number of attention heads and momentum, and observed how the results on the test set changed. The learning rate that was used was $\eta = 0.01$. We tried varying the learning rate between 0.1 and 0.001 but found that 0.01 was the best performing learning rate. We also tried using 60 lead times with 5 minute increments but found that the model was not able to cope with this many lead times. So, we reduced these to 8 lead times and 15 minute increments. For momentum, due to time restraints, we only tried one value, $\alpha = 0.9$,[3] and it did the job better than having no momentum term. This was a recurring theme of the hyperparameter search: even if we wanted to optimize our model's hyperparameters, the resources allocated simply did not allow it. Since the project was allocated 1080 GPU-h/month and the networks we use take at least 250 GPU-h to train, we were limited to training 4 networks per month, thus making an extensive hyperparameter search unfeasible. We therefore relied, to some extent, on the hyperparameters reported by Google [4].

4.4 Evaluation Benchmarks

To evaluate our model's ability for predicting rainfall we have used the F1-score on a separate test dataset, see Equation 4.3. F1-score is a score that is commonly used for tail heavy datasets; it addresses the problem of only guessing on the most common occurring event. Since only a small fraction of pixels contain rain a guess of "no rain" would most of the time achieve very high accuracy but F1-score is directly proportional to the number of true positives so when there are no true positives the F1-score will be 0. There are two ways of averaging the F1-score across multiple samples: one way is to take the mean F1 score across all samples. However, if most samples contain little to no rain, then most F1-scores will likely be zero since, if there is only one pixel with rain, the model is unlikely to guess this pixel correctly and the F1-score for this sample will be zero. Additionally, the F1-score won't be defined when all pixels show no rain. When averaged across multiple samples the F1-score will then be very low. So, instead, the number of true positives (TP), false positives (FP) and false negatives (FN) are calculated through all samples and then the formula in Equation 4.3 is used to calculate the F1-score. We have opted to use this F1-score averaging technique since our dataset is very tail-heavy towards no rain.

$$\mathrm{F1} = \frac{TP}{TP + 0.5(FP + FN)} \tag{4.3}$$

Due to time constraints most of the time was spent developing the model and training it. Because of this we have opted for using the simplest possible benchmarks to compare our model with: one is persistence and another is a more vanilla RNN similar to MetNet except that it omits the spatial aggregator. In persistence

[3] This value was not optimized because of reasons explained in this chapter, however $\alpha = 0.9$ is a common value used for SGD.

we take the rain image at $T = 0$ minutes, let it act as a prediction guess and evaluate it against the prediction of the model. By evaluating our model against a vanilla RNN we can get powerful insights on the value of the spatial aggregator and its functions.

During training a random lead time is generated for each training sample and paired with the corresponding ground truth in order to reserve memory and to replicate the method used by Google [4]. The training loss is therefore a little bit random depending on which lead times were sampled. However during validation it is important to always evaluate the same dataset in order to reduce stochasticity in the validation loss, therefore the validation loss is evaluated on all lead times and all validation samples. Finally once a validation minimum is reached then the evaluation is done on a separate test set that the model has not yet seen.

Another way of evaluating DNNs forecasting skill is through qualitative metrics. Ravuri et al. [25] evaluated their model using two different methods. One method was to let a group of independent expert meteorologists compare the GAN and other state-of-the-art models with the ground truth. At the end they all came together to discuss their evaluation and concluded that the article's deep generative model was preferred for "best fit and rates overall". The article claims that their model makes more skilful predictions when compared with the other models by the meteorologists in subjective qualitative metrics. However, the quantitative metrics are similar for all models. Critically the article concludes that qualitative measurements of forecasting models are just as important as quantitative measurements, highlighting the need for new qualitative metrics when evaluating forecasting models.

Chapter 5

Results

In this chapter we present the convergence during training in terms of training and validation loss. Then we present the performance of the model and compare with the previously discussed benchmarks; the more vanilla model and persistence.

5.1 Validation Loss

The validation loss during training is shown in Figure 5.1 and 5.2 as a function of the number of global steps.[1] During training the model learns to predict the precipitation of the target patch better and better and this can be seen in Figure 5.1 as the validation loss steadily decreases. After $70,000$ global steps the model's convergence stagnates and the network is then reinitialized with a smaller learning rate to fine tune the weights. In figure 5.2 we see how the network starts to overfit the data as the validation loss increases. The validation loss is calculated for each epoch and if it reaches a new minimum the network is saved. As discussed in Section 4.2 we have a categorical cross entropy loss function over the 28×28 pixel prediction zone and we calculate the average validation loss as:

$$loss = \frac{-\sum_{n=0}^{N_{samples}} \sum_{i=0}^{28} \sum_{j=0}^{28} y_{i,j}^{n} \cdot \log \hat{y}_{i,j}^{n}}{N_{samples}} \tag{5.1}$$

where $y_{i,j}^{n}$ is the one-hot encoded ground truth, $\hat{y}_{i,j}^{n}$ is the prediction and $N_{samples} \approx 3,300$ is the number of validation samples. Note that the terms are a dot-product between the one-hot encoded ground truth and the network's estimated probability of each rain class.

[1] The number of global steps refers to the number of optimizer steps taken, or gradient steps. Since each batch is 8 training samples and there are at every given time 8 different GPUs processing a batch, effective batch size is 64 training samples. With $4,400$ training samples we get $\frac{4,400}{64} \approx 69$ global steps per epoch.

As can be seen in Figure 5.1 the validation loss sometimes spikes, which is likely due to the tail-heavy nature of our dataset as well as underestimating "no rain" probabilities. We assess this in detail in Chapter 6. Figure 5.2 shows that the validation loss steadily increases and starts to overfit after reinitializing the network with a smaller learning rate — meaning that the variance is getting too high. This will also be discussed in detail in Chapter 6. The training loss is shown in Figure 5.3 and Figure 5.4. It decreases even though the validation loss increases, which is a clear sign of overfitting.

5.2 F1 Performance

We measure forecast performance with F1-score and compare it against the persistence benchmark. The network with the lowest validation score was evaluated by its F1-score when a threshold is applied at different precipitation levels. Since the output of the network is a probability distribution of different precipitation rates, the probabilities above a certain threshold were summed together. For light rainfall > 0.2mm/h we obtain the result on the test dataset shown in Figure 5.5 by summing the probabilities of all categorical bins corresponding to the applied threshold. Figure 5.5 shows the F1-score on the test dataset as a function of the 8 different lead times. The network shows better performance than the baseline. However, performance is unfortunately still poor. When compared with persistence it fares a little bit better after 30 minutes but does not outperform persistence, like MetNet does, for all lead times.

When we visualize some samples of heavy rainfall events we can see that the model manages to capture the general behavior of the clouds. See Figure 5.6 for an example where the network succeeds to predict the future state and Figure 5.7 where the network struggles. It seems that the network is able to predict low-pressure rain fronts accurately where the motion of the clouds is less complex. But when the cloud coverage is sparse the model's predictions are much worse.

We measured the performance on higher precipitation thresholds > 0.6mm/h and > 1 mm/h, see Figure 5.8 and 5.9. Here we summed all bins corresponding to the thresholds, analogously to what was done in the case of the 0.2mm/h threshold. The F1-score for the model drops significantly for higher thresholds than 0.2mm/h.

5.3 Vanilla Model

We tried training a more vanilla network, using only the spatial downsampler and temporal encoder, to observe how it would fare against the more complex model. The results show that the simpler model struggles with differentiating between 15 and 120 minute lead times, offering more or less the same prediction for all lead times. As a result the F1-score is very low for all lead times, as can be seen in Figure 5.11. The model is less flexible when comparing the prediction between

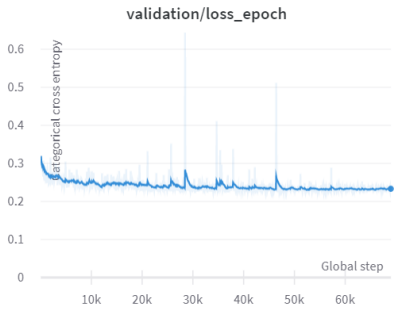

Figure 5.1. Smoothed validation loss during $1,000$ epochs or $\sim$70 thousand global steps with a learning rate $\eta = 0.01$.

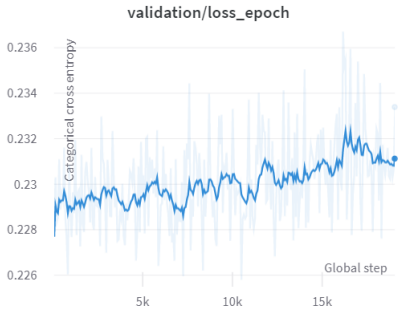

Figure 5.2. Continuation of validation loss for another $\sim$20 thousand global steps with a lower learning rate $\eta = 0.001$. Overfitting is observed.

Figure 5.3. Training loss during $1,000$ epochs or $\sim$70 thousand global steps with a learning rate $\eta = 0.01$.

Figure 5.4. Continuation of training loss for another $\sim$20 thousand global steps with a lower learning rate $\eta = 0.001$.

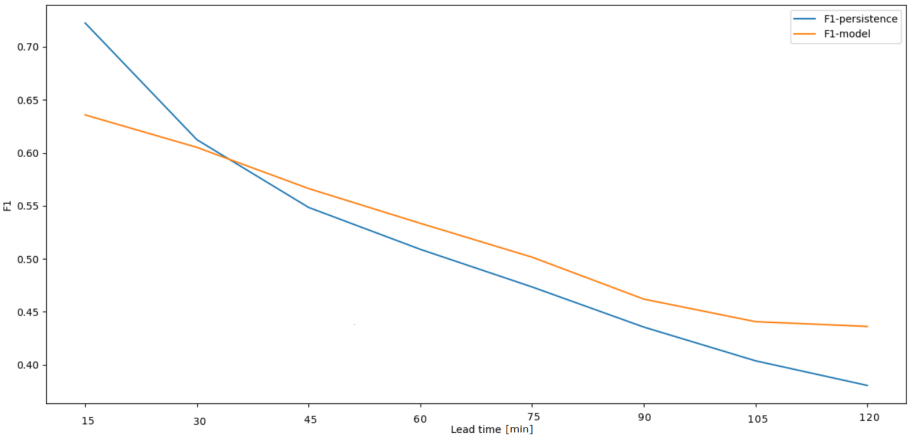

Figure 5.5. F1-score for light rainfall > 0.2mm/h for different lead times compared with persistence benchmark.

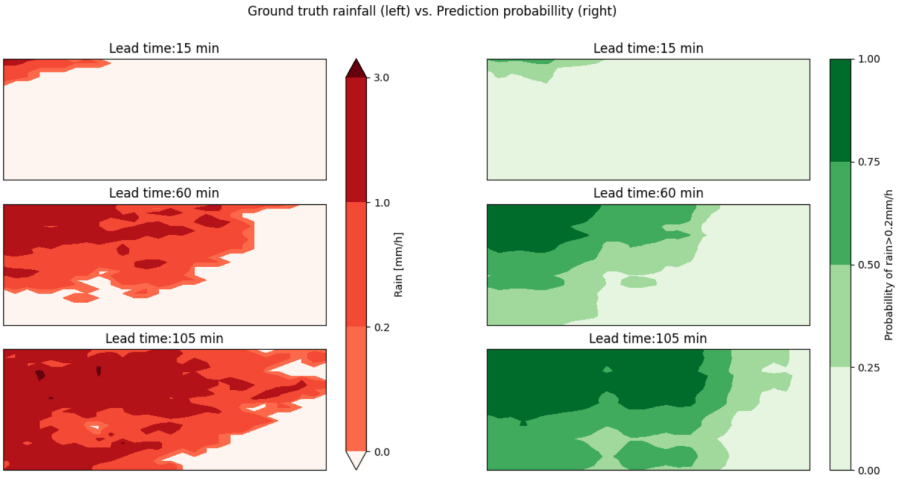

Figure 5.6. Example of successful prediction with model. As can be seen for all three lead times: the model outputs a similar shape as the ground truth.

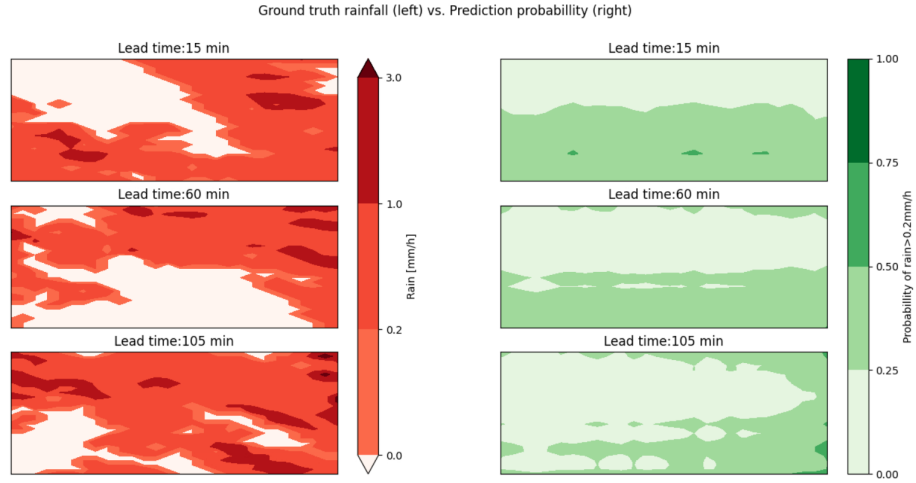

Figure 5.7. Example of failed prediction with model. As can be seen for all three lead times: the model fails to capture the shape of the precipitation fields.

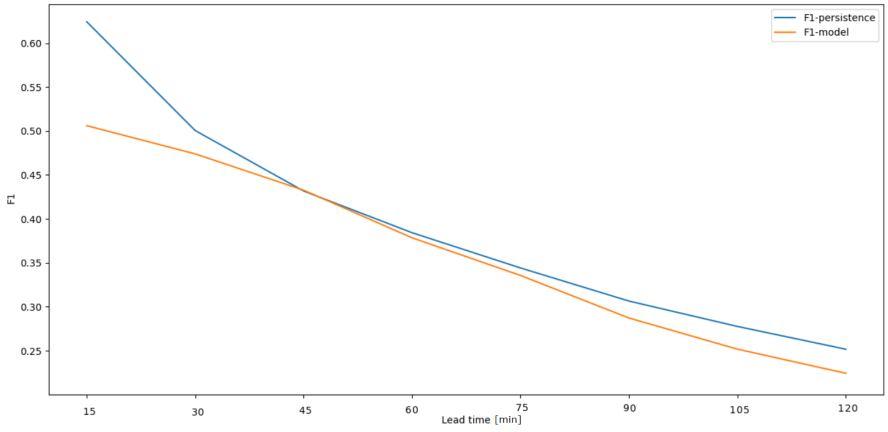

Figure 5.8. F1-score for light rainfall > 0.6mm/h for different lead times compared with persistence benchmark.

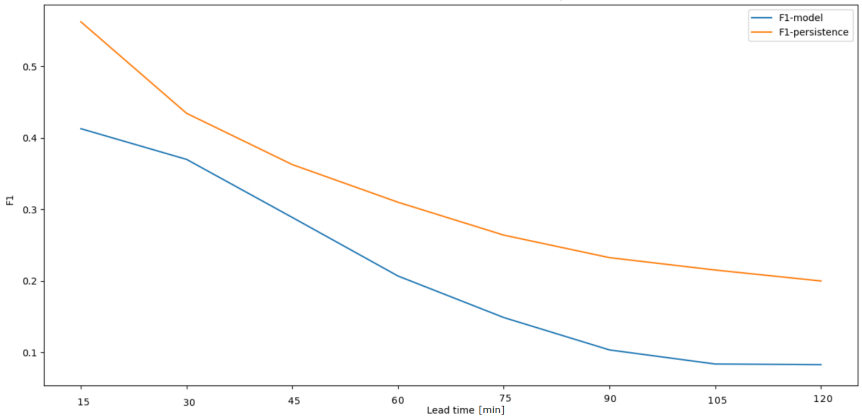

Figure 5.9. F1-score for light rainfall > 1mm/h for different lead times compared with persistence benchmark.

different lead times and the predictions seem to be more constant in time and less dependent on lead time, as illustrated in Figure 5.10. This tells us something about the value of the spatial aggregator in our network; more on this in Chapter 6. This tells us a spatial aggregator is needed, although it does not necessarily have to be a vision transformer; the followup model by Google "MetNet-2" does not use an axial attention layer but instead a series of convolutional layers [69].

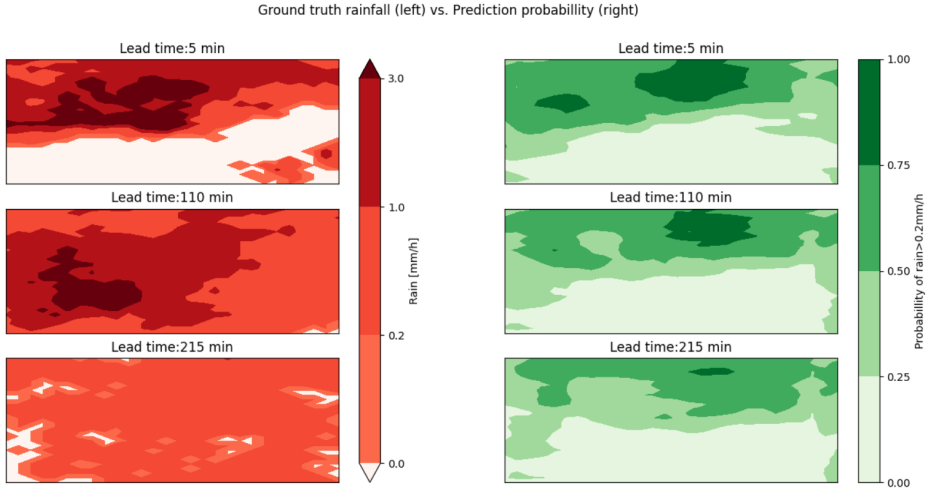

Figure 5.10. Example of failed prediction with vanilla model. The model seems to capture something of the shape of the precipitation of one lead time but cannot differentiate between different lead times; the precipitation fields in the prediction are more or less constant.

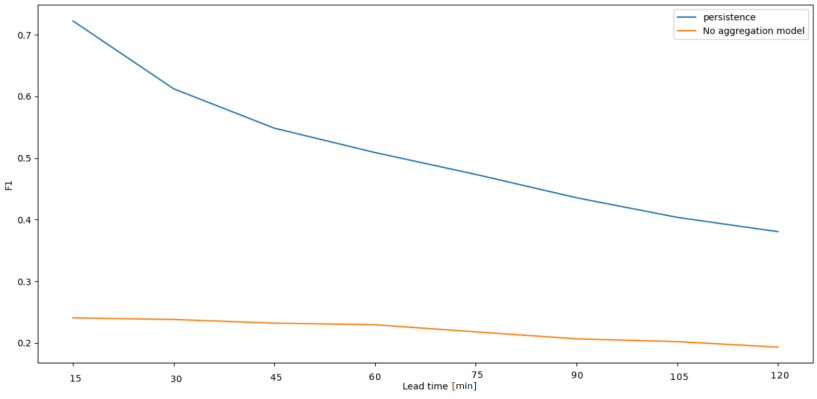

Figure 5.11. F1-score with vanilla setup, light rainfall $>$ 0.2mm/h for different lead times compared with persistence benchmark.

Chapter 6

Discussion

We did not use the same data or network as Google, so we can only compare their results to the results of our setup. Our network was not able to reproduce the same performance as the network implemented by Google. See Figure 6.1 for their results. For light rainfall > 0.2 mm at 15 minutes lead times they have an F1-score of about 0.85 while we have an F1-score of 0.63, which is even lower than persistence. At longer lead times the gap between our model and Google's increases further and at 120 minutes they achieve an F1-score of 0.7 while we drop to 0.43, which is higher than persistence but still too low in comparison to their two benchmarks (optical flow and the physical model HRRR). It should however be noted that their persistence seems to be consistently 0.1 higher than our persistence benchmark. This might be caused by a number of reasons, for example our prediction zone is on the coastline of Sweden where the weather might be more unstable or the data might be more noisy. MetNet performed their evaluation with qualitative fields around the radar towers, which means they only evaluated the model in areas where the data was of high quality. Since the cloud coverage closer to a radar tower has less noise this reduces the noise effects on the evaluation score. Nonetheless, even if we assume that our model would perform better on higher quality data and increase in F1-score by the same value as the persistence F1-score gap between our two benchmarks, namely 0.1, our model still would not reach the same performance as MetNet.

Our model was only trained on a single geographic location, which greatly reduces the amount of training data available compared with Google that used the entire United States for training their model. In fact, MetNet needed to use 1.7 million training samples before they stopped observing overfitting on the validation set with their 225 million parameter model. Our model is much smaller, with 7.3 million parameters and the dimensions of our data as well as lead time length and lead time resolution are also much smaller. This initially justified a simpler setup with a stationary model that resulted in only $4,400$ data samples (more about this later). But, as we saw in Section 5.1, our model overfits the training data, therefore

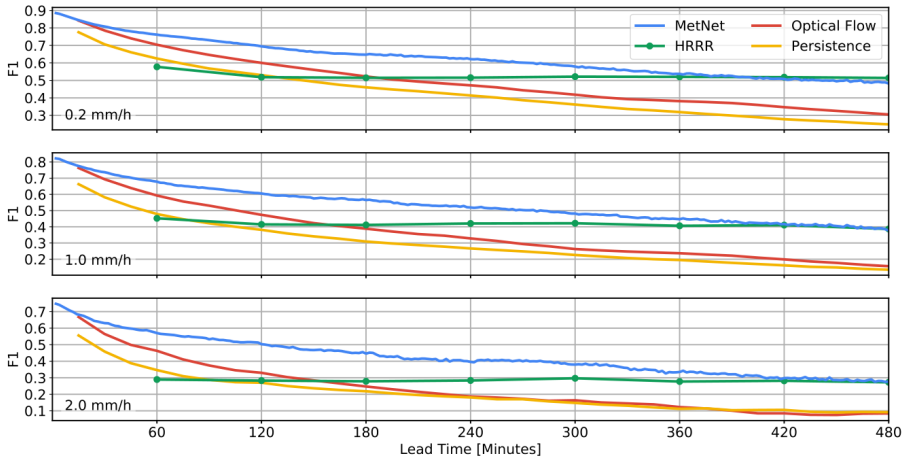

Figure 6.1. F1 performance of MetNet by Google. MetNet is a 225 million parameter DNN, Optical flow is an optical tool for precipitation forecasting and HRRR is a physical model relying on advection from wind fields. Figure extracted from Sønderby et al. [4] with permission from the authors.

we conclude that we need more data to reach the model's full potential. The used radar map of Sweden consists of 881×454 pixels and our input patch is 448×448, which takes up about half of the total area available. However, since the model is only evaluated on the small 28×28 prediction zone, we could divide the dataset into 16 non-overlapping target zones and thereby get 16 times more training data. Google also divides their data into non-overlapping targets but since the US is much bigger than Sweden they can generate about $2,000$ training samples per input frame. Since weather in the US follows the same physical laws as weather in Sweden, we believe that pretraining a Swedish model on the US dataset could give a better performing model.

There are other benefits in using a flexible geographic model. For one, all the geographic metadata in a geostationary model is constant throughout all training samples, which makes the variance of these data fields zero and thus increases the risk of the model ignoring their effects on the weather rather than just memorizing local patterns in the image. However we still hypothesized that the model would have an easier time learning the effects of, for instance, a local mountain with the information of where a mountain exists rather than without it.

For 15 and 30 minute lead times our model performs worse than persistence even though these should be the easiest to predict. An explanation for this might be that our model has not yet learned a flexible enough representation and cooperation between the three architecture layers. The downsampler is supposed to identify and compress the information of input weather states like cloud positions, cloud

fronts, mountain blockades etc. The temporal encoder is supposed to identify cloud velocities, convective initiations and diffusion rates. Finally, the spatial aggregator is supposed to analyze all this information to propagate the state forward with respect to the desired lead time. By comparing our model with the vanilla model we can see that the spatial aggregator indeed fulfills this role to some extent, since the images generated by the vanilla model are more constant in time, as shown in Figure 5.10, and the images generated by the full model can support some variability depending on lead time. The vanilla model does not have the flexibility to generate different predictions for different lead times, therefore it generates some averaged prediction over the entire span of the forecast to minimize the loss.

Another reason for failing to produce as good results as persistence for the 15 and 30 minute lead times would be that information about the initial state might be lost in the depth of our network. Since the information of the persistence plot is completely available in the input patch, the network should be able to find this state. However, since it also needs to be able to predict lead times above 30 minutes, this state is not the most important and the network rather opts to find a representation of the initial that works for all lead times. Another common architecture called a U-Net — which has shown promising results for the precipitation forecasting task [70, 71] — addresses the issue of forgetting the initial state by adding a bypass between the input layer directly to the final output computational nodes in order to increase performance [72].

The validation loss shown in Figure 5.1 has some clear spikes that need to be addressed. We believe that these are a result of the tail-heavy quality of our dataset. The network is tasked with predicting the future precipitation map but, in nature, as discussed in Section 3.6, rain is a relatively rare occurrence. So, if the network, at some tipping point, leans toward predicting rain rather than predicting no rain when it is uncertain, then the loss will spike. What causes the network then to lean towards predicting rain rather than no rain? This could have a number of explanations. The first would be the stochasticity of a single training epoch: since all samples are only paired with a random single lead time per epoch, there is some variation on which training samples are picked. This leads to a variation in the number of rainy pixels present in each epoch, which could affect such a tipping point and make the model lean towards predicting rain rather than no rain. Another reason might be the introduced stochasticity of the batch normalization in the spatial downsampler.

As discussed earlier, our model architecture is much simpler than the one implemented by Google, thus justifying our geostationary implementation even though it imposed limits on the amount of training data. However, as we can be seen in Figure 5.2, it still clearly overfits the training data. This, in hindsight, puts into question the assumption that a geostationary implementation was good enough in our case. The model has too many degrees of freedom and is trained with too little data. Even if our model is less complex than MetNet, it is tasked with describing the same type of physical phenomena. In order to learn all the intricate details of the weather system and how all the different variables dictate the motion of the

system, more data is needed to prevent overfitting in our model. Or alternatively, one could reduce the complexity of the model even further to achieve a better bias–variance tradeoff.

The original MetNet model and our model have a few significant differences. As mentioned in Section 4.1.5, our implementation of the spatial aggregator is somewhat different from the original paper by Google [4]. When running with the same parameters our implementation still only achieves 17 million parameters. When compared with their 225 million parameters we can say something is fundamentally different between the two models. When we contacted the authors to investigate this difference there seemed to be a consensus that our implementation of the axial attention layer is different [C. Sønderby, personal communication, 5 March, 2022]. According to the authors of MetNet, the majority of the parameters in their implementation come from the spatial aggregator $\sim$200 million which lets us conclude that these are differently implemented. This means that our model is not a perfect replication of Google's, neither for that matter a perfectly downscaled version. Our model overfits and is therefore not trained to its fullest potential. To increase performance we would certainly need to increase the number of data samples during training.

Another interesting difference between our networks is the Geostationary Operational Environmental Satellite (GOES) data that MetNet uses. Since Sweden is far to the north there are no geostationary satellites that can take bird's-eye pictures of Sweden, which renders the quality of equatorial satellites of less quality over this area. GOES in MetNet operates with 16 spectral bands and can capture information about humidity, unlike radar towers that only capture water droplets. This can have a substantial impact on the model's ability to detect events like convective initiations and high humidity fields that can turn into rain clouds. When the radar towers detect a decrease of water content in the atmosphere the network does not know if this water has fallen down as rain or vaporized into humid air. However, one might also argue that, with a sophisticated enough spatial downsampler and temporal encoder, it should be able to identify and differentiate between these two events. MetNet did some tests where they only used GOES and no MRMS data, which led to a significant drop in performance for the model[4]. If a model without GOES could be as good as one with GOES, then a model like MetNet could be implemented also in the northern hemisphere without reducing its performance.

In section 2.6 we discussed the bias–variance tradeoff and how a too simple model would have an excessive bias on both training and validation data while a high variance model would do well on the training data but not generalize so well to an unseen validation dataset. Judging from the results of our model on F1-score (see Figure 5.5) either the model is too simple and has a high bias, as illustrated in Figure 2.2, or the dataset is too small and the network can only learn simple patterns of the atmosphere. By stopping training when the validation loss increases we avoid overfitting and therefore excessive variance in our model, but here is the real dilemma: if we increase the model complexity further, then the overfitting will take place even earlier since the model now has more memory capacity and can

thus focus on memorizing training samples earlier. This can be thought of in the same way an $n : th$-degree polynomial can perfectly fit a regression of n data points in the plane. We would thus need to add even more data points of our distribution to account for the added complexity. In order to use the extra complexity of our model we need to increase the number of training data samples. So, to improve our results, we would look into a more complex model to decrease the bias of our network, going towards the 225 million parameter model by Google, but we would also have to significantly increase the number of training data samples to avoid overfitting with the added complexity.

A final note should also be added about the lack of an extensive hyperparameter search and the potential limits we have in our setup. Hyperparameters are important because they dictate how the training of the model is performed. For this project the hyperparameters are the learning rate, the momentum coefficient, the initialization of the weights, the mini-batch size, the batch normalization coefficient in the spatial downsampler and the number of hidden layers in the RNN. The learning rate is generally regarded as the most important hyperparameter [73], as a too small learning rate will cause the optimisation process to get stuck in local minima of the error function while a too big hyperparameter will overshoot the path to the global minima and fail to descend on the loss manifold. It has been shown [74] that the momentum coefficient plays a big role in the training of DNNs and RNNs, especially when there is no well-designed random initialization scheme. If the momentum coefficient is not tuned well, it can even result in a performance drop when compared to a model without momentum [74]. This statement is powerful and critically reduces the conclusions we can draw about our model since we have not verified that our momentum parameter is optimal or that our Gaussian weight initialization is optimal. In order to rule out a model as bad, the hyperparameter settings need to be optimized first.

Chapter 7

Conclusions

In this thesis we have investigated the use of deep neural networks (DNNs) for precipitation nowcasting with radar data in Sweden for lead times up to 120 minutes. We have implemented a simplified – but functional – network inspired by MetNet, from Google AI [4], that combines three different deep architectures: a convolutional neural network, a convolutional long short-term memory and an axial self-attention transformer. We trained the model using stochastic gradient descent and a categorical cross entropy loss function. To evaluate the network we compared it against a persistence benchmark and against a more vanilla recurrent neural network model.

The results show that the network, in terms of F1-score for a light rainfall threshold of 0.2mm/h, is able to beat persistence for lead times longer than 30 minutes but does not beat the persistence benchmark for higher thresholds like 0.6mm/h or 1.0mm/h. Comparing the vanilla RNN and the full model, it was revealed that the role of the axial self-attention spatial aggregator plays an essential role in the network's ability to be flexible and predict different precipitation maps for different lead times. However, our model's performance is not as good as MetNet [4] and we conclude that the complexity of our model is probably too small. This is also the case for the size of the training dataset, which led to high biases.

To improve our results, future work should investigate the axial self-attention layer to assess how to better replicate the actual implementation of MetNet. To account for the added complexity, more data needs to be added either by 1) extending our five year data window, 2) implement a geographically flexible input frame instead of the geostationary setup we used or 3) pretrain the network on another dataset *e.g.* the dataset MetNet used. Once a more complex model and a larger training dataset have been established, some work should be done finding the optimal hyperparameters for the setup. Due to constraints in time and resources, we could unfortunately not optimize our setup, but the effects of optimal hyperparameters could be significant. We have, nonetheless, demonstrated that there is potential in DNNs for precipitation nowcasting.

Bibliography

[1] G. Hu *et al.*, *When Face Recognition Meets with Deep Learning: an Evaluation of Convolutional Neural Networks for Face Recognition*, CoRR **abs/1504.02351** (2015), 1504.02351.

[2] M. Bansal, A. Krizhevsky and A. S. Ogale, *ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst*, CoRR **abs/1812.03079** (2018), 1812.03079.

[3] T. Wolf *et al.*, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38–45, Online, 2020, Association for Computational Linguistics.

[4] C. K. Sønderby *et al.*, Metnet: A neural weather model for precipitation forecasting, 2020.

[5] U. Andrae *et al.*, *A continuous EDA based ensemble in MetCoOp*, (2020).

[6] R. B. Alley, K. A. Emanuel and F. Zhang, *Advances in weather prediction*, Science **363**, 342 (2019), https://www.science.org/doi/pdf/10.1126/science.aav7274.

[7] K. S. Rollins and J. Shaykewich, *Using willingness-to-pay to assess the economic value of weather forecasts for multiple commercial sectors*, Meteorological Applications **10**, 31 (2003), https://rmets.onlinelibrary.wiley.com/doi/pdf/10.1017/S1350482703005048.

[8] T. J. Teisberg, R. F. Weiher and A. Khotanzad, *The Economic Value of Temperature Forecasts in Electricity Generation*, Bulletin of the American Meteorological Society **86**, 1765 (2005).

[9] J. Gagne *et al.*, *New method for aircraft fuel saving using Flight Management System and its validation on the L-1011 aircraft* , https://arc.aiaa.org/doi/pdf/10.2514/6.2013-4290.

[10] F. Hashemi and W. Decker, *Using climatic information and weather forecast for decisions in economizing irrigation water*, Agricultural Meteorology **6**, 245 (1969).

[11] D. Letson, D. S. Sutter and J. K. Lazo, *Economic Value of Hurricane Forecasts: An Overview and Research Needs*, Natural Hazards Review **8**, 78 (2007).

[12] P. Bauer, A. Thorpe and G. Brunet, *The quiet revolution of numerical weather prediction*, Nature **525**, 47 (2015).

[13] R. J. Charlson *et al.*, *Atmospheric science. Reshaping the theory of cloud formation*, Science (American Association for the Advancement of Science) **292**, 2025 (2001).

[14] G. Ayzel, T. Scheffer and M. Heistermann, *RainNet v1.0: a convolutional neural network for radar-based precipitation nowcasting*, Geoscientific Model Development **13**, 2631 (2020).

[15] X. Shi *et al.*, Advances in Neural Information Processing Systems, edited by I. Guyon *et al.*, Vol. 30, Curran Associates, Inc., 2017.

[16] A. Cleveland, *THE PHYSICAL BASIS OF LONG-RANGE WEATHER FORECASTS*, Monthly Weather Review **29**, 551 (1901).

[17] V. Bjerknes, *Das Problem der Wettervorhersage, betrachtet vom Standpunkte der Mechanik und der Physik in Meteorologische Zeitschrift 21 pp. 1-7, 1904* (Ed. Hölzel, Wien, 1904).

[18] L. F. Richardson, Weather prediction by numerical process, Weather Prediction by Numerical Process. 2nd Edn. with Foreword by Peter Lynch (2007)., Cambridge University Press., 1922.

[19] P. Lynch and L. Richardson, *The Emergence of Numerical Weather Prediction: Richardson's Dream* (Cambridge University Press, 2006).

[20] J. M. Wallace and P. V. Hobbs, 7 - atmospheric dynamics, Atmospheric Science (Second Edition), edited by J. M. Wallace and P. V. Hobbs, pp. 271–311, Academic Press, San Diego, , second edition ed., 2006.

[21] N. P. Wedi, *Increasing horizontal resolution in numerical weather prediction and climate simulations: illusion or panacea?*, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **372**, 20130289 (2014), `https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2013.0289`.

[22] T. Gneiting and A. E. Raftery, *Weather Forecasting with Ensemble Methods*, Science **310**, 248 (2005), `https://www.science.org/doi/pdf/10.1126/science.1115255`.

[23] X. J. Yang *et al.*, *The TianHe-1A Supercomputer: Its Hardware and Software*, Journal of Computer Science and Technology **26**, 344 (2011).

[24] J. N. Liu *et al.*, Proceedings on the International Conference on Artificial Intelligence (ICAI), p. 1, The Steering Committee of The World Congress in Computer Science, Computer . . . , 2014.

[25] S. Ravuri *et al.*, *Skilful precipitation nowcasting using deep generative models of radar*, Nature **597**, 672 (2021).

[26] B. Horn and B. Schunck, *Determining Optical Flow*, Artificial Intelligence **17**, 185 (1981).

[27] D. Reidmiller *et al.*, Fourth National Climate Assessment, U.S. Global Change Research Program, Washington, DC, USA, 1515, 2018.

[28] O. Melin *et al.*, *Skyfallsplan för Malmö*, (2017).

[29] J. Molinder *et al.*, *Probabilistic Forecasting of Wind Turbine Icing Related Production Losses Using Quantile Regression Forests*, Energies **14** (2021).

[30] L. Gao *et al.*, *A field study of ice accretion and its effects on the power production of utility-scale wind turbines*, Renewable Energy **167**, 917 (2021).

[31] A. G. Kraj and E. L. Bibeau, *Phases of icing on wind turbine blades characterized by ice accumulation*, Renewable Energy **35**, 966 (2010).

[32] J. Bieker and J. Kelly, Metnet, `https://github.com/openclimatefix/metnet`, 2022.

[33] F. Wirtz, Quarterly progress update, q3/2021, `https://www.openclimatefix.org/blog/2021-09-01-update`, 2021.

[34] R. Vinuesa *et al.*, *The role of artificial intelligence in achieving the Sustainable Development Goals*, Nature Communications **11**, 233 (2020).

[35] The 17 goals, `https://www.globalgoals.org/goals/`, 2015.

[36] H. Wang *et al.*, *Improving the Predictability of Severe Convective Weather Processes by Using Wind Vectors and Potential Temperature Changes: A Case Study of a Severe Thunderstorm*, Advances in Meteorology **2016**, 8320189 (2016).

[37] S. Scher and G. Messori, *How Global Warming Changes the Difficulty of Synoptic Weather Forecasting*, Geophysical Research Letters **46** (2019).

[38] M. C. Bishop, *Pattern Recognition and Machine Learning* (Springer Publishing, University of California, Berkeley, CA 94720, USA, 2006).

[39] S. Singh, Understanding the bias-variance tradeoff, 2018.

[40] E. Brynjolfsson and T. Mitchell, *What can machine learning do? Workforce implications*, Science **358**, 1530 (2017), https://www.science.org/doi/pdf/10.1126/science.aap8062.

[41] J. Connor, R. Martin and L. Atlas, *Recurrent neural networks and robust time series prediction*, IEEE Transactions on Neural Networks **5**, 240 (1994).

[42] J. Martinez, M. J. Black and J. Romero, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[43] Y. Chen *et al.*, Medical Image Computing and Computer Assisted Intervention – MICCAI 2018, edited by A. F. Frangi *et al.*, pp. 91–99, Cham, 2018, Springer International Publishing.

[44] K. Dhiram and Z. Wang, *Evaluation on radar reflectivity-rainfall rate (ZR) relationships for Guyana*, Atmospheric and Climate Sciences **6**, 489 (2016).

[45] Berzelius cluster, https://www.nsc.liu.se/systems/berzelius.

[46] A. Paszke *et al.*, Pytorch: Crossentropyloss.

[47] X. SHI *et al.*, Advances in Neural Information Processing Systems, edited by C. Cortes *et al.*, Vol. 28, Curran Associates, Inc., 2015.

[48] J. Ho *et al.*, *Axial Attention in Multidimensional Transformers*, CoRR **abs/1912.12180** (2019), 1912.12180.

[49] K. Fukushima, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Biological Cybernetics **36**, 193 (1980).

[50] D. H. Hubel and T. N. Wiesel, *Receptive Fields and Functional Architecture of Monkey Striate Cortex*, Journal of Physiology (London) **195**, 215 (1968).

[51] Y. LeCun *et al.*, Advances in Neural Information Processing Systems, edited by D. Touretzky, Vol. 2, Morgan-Kaufmann, 1989.

[52] A. Krizhevsky, I. Sutskever and G. E. Hinton, Advances in Neural Information Processing Systems, edited by F. Pereira *et al.*, Vol. 25, Curran Associates, Inc., 2012.

[53] T. Wang *et al.*, Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), pp. 3304–3308, 2012.

[54] A. Shrestha and A. Mahmood, *Review of Deep Learning Algorithms and Architectures*, IEEE Access **7**, 53040 (2019).

[55] M. A. Nielsen, *Neural Networks and Deep Learning* (Determination Press, 2015).

[56] A. Vaswani *et al.*, Advances in Neural Information Processing Systems, edited by I. Guyon *et al.*, Vol. 30, Curran Associates, Inc., 2017.

[57] A. Dosovitskiy *et al.*, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, CoRR **abs/2010.11929** (2020), 2010.11929.

[58] Z. Liu *et al.*, *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*, CoRR **abs/2103.14030** (2021), 2103.14030.

[59] Z. Liu *et al.*, *Video Swin Transformer*, CoRR **abs/2106.13230** (2021), 2106.13230.

[60] H. Cao *et al.*, Swin-unet: Unet-like pure transformer for medical image segmentation, 2021.

[61] N. O'Mahony *et al.*, Science and information conference, pp. 128–144, Springer, 2019.

[62] M. Zinkevich *et al.*, Advances in Neural Information Processing Systems, edited by J. Lafferty *et al.*, Vol. 23, Curran Associates, Inc., 2010.

[63] A. Paszke *et al.*, Pytorch: Softmax.

[64] P. Werbos, *Backpropagation through time: what it does and how to do it*, Proceedings of the IEEE **78**, 1550 (1990).

[65] I. Sutskever *et al.*, Proceedings of the 30th International Conference on Machine Learning, edited by S. Dasgupta and D. McAllester, Vol. 28, of *Proceedings of Machine Learning Research*, pp. 1139–1147, Atlanta, Georgia, USA, 2013, PMLR.

[66] A. Paszke *et al.*, Pytorch: An imperative style, high-performance deep learning library, Advances in Neural Information Processing Systems 32, edited by H. Wallach *et al.*, pp. 8024–8035, Curran Associates, Inc., 2019.

[67] W. Falcon and . The PyTorch Lightning team, PyTorch Lightning, 2019.

[68] L. Biewald, Experiment tracking with weights and biases, 2020, Software available from wandb.com.

[69] Metnet-2: Deep learning for 12-hour precipitation forecasting, https://ai.googleblog.com/2021/11/metnet-2-deep-learning-for-12-hour.html, 2021.

[70] K. Trebing and S. Mehrkanoon, *SmaAt-UNet: Precipitation Nowcasting using a Small Attention-UNet Architecture*, CoRR **abs/2007.04417** (2020), 2007.04417.

[71] J. Mathur, Illinois's climatehack team, `https://github.com/jmather625/climatehack`, 2022.

[72] O. Ronneberger, P. Fischer and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, CoRR **abs/1505.04597** (2015), `1505.04597`.

[73] L. N. Smith, 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 464–472, 2017.

[74] I. Sutskever *et al.*, Proceedings of the 30th International Conference on Machine Learning, edited by S. Dasgupta and D. McAllester, Vol. 28, of *Proceedings of Machine Learning Research*, pp. 1139–1147, Atlanta, Georgia, USA, 2013, PMLR.