UiO : Faculty of Mathematics and Natural Sciences
University of Oslo

DiffMet: Diffusion models and deep learning for precipitation nowcasting

Gard Pavels Høivang

Master Thesis

2023

Abstract

Predicting near-future rainfall, defined as *nowcasts*, is essential to various industries and sectors, including aviation safety, agriculture, flood management, and the general public. Traditional nowcasts are calculated using optical flow methods, which extrapolate radar observations. However, these methods have shortcomings and struggle to capture important non-linear convective events. Deep learning has shown promising results in mitigating these challenges. Especially promising are frameworks built on generative models (GM), enabling the generation of realistic precipitation nowcasting scenarios. Diffusion Models (DM) are a specific form of GMs and have advanced tremendously over the last three years. To our knowledge, these models have yet to be explored for nowcasting or general weather forecasting. With this as motivation, this thesis presents a novel approach to precipitation nowcasting using DMs.

These models have mainly been implemented as unconditional or label-dependent models. However, for precipitation nowcasting, generating samples conditioned on radar video from past time steps is necessary. This thesis suggests two strategies for generating such conditional samples. First, using the radar video directly as input to the network, and secondly, creating *image embeddings* of the input by processing it through a second neural network. Our results show that both models possess predictive abilities and can generate realistic nowcasts with lead times from five to 20 minutes ahead.

Model training and validation are performed on a dataset compiled by extracting and processing data from an API provided by the Norwegian meteorological institute. The primary variable of interest is radar precipitation rates from a 256×256-square kilometer area of eastern Norway and parts of Sweden, extracted from April to October for 2021 and 2022. As this results in massive amounts of data, video sequences are thresholded and selected based on precipitation intensities to create a manageable-sized data set.

We implement metrics commonly used in meteorology to validate model output and to assess nowcast probabilistic quality. We found that the model utilizing image embeddings produced higher-quality predictions than the model with direct conditional input. However, both models suffer from poor results when generating nowcasts with little- to no rain prior to lead times. This is likely caused by these scenarios being underrepresented in the compiled dataset.

Acknowledgements

This thesis is a joint work between the University of Oslo (UiO), Simula Metropolitan Center for Digital Engineering (SimulaMET), and The Norwegian Meteorological Institute (MET). Thanks to my supervisors Hugo L. Hammer (SimulaMET) and John Bjørnar Bremnes (MET), for our weekly discussions! Thanks to Michael Riegler (SimulaMET) and Nikki Nikki Vercauteren (UiO - METOS) for valuable guidance along the way. The help from all of you has been invaluable.

Also, thanks to Tonje and our two girls, Mathilde and Karen, for all the encouragement and support!

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background and Motivation

Predicting short-term rainfall events is of great interest to a wide range of users. Population growth and expanded urbanization result in an increased risk for loss of lives due to severe weather [56]. Developing methods to accurately predict the location and intensity of near-future precipitation events can aid decision-makers in critical situations and bring improvements to aviation safety, agriculture, flood management, and the general public.

These short-scale forecasts of future weather are called *nowcasts*. Today's operational nowcasting methods are based on extrapolating observed precipitation fields forward in time but have been shown to have several shortcomings. Especially challenging is the prediction of the non-linear events often connected with heavy precipitation [49]. As data from the atmosphere is continuously collected through various remote sensing systems, this makes for a perfect application of machine learning (ML). Since the first ML-based precipitation nowcasting model presented by Shi et al. in 2015, several papers have made major advancements towards reaching the performance of extrapolation-based nowcasts methods currently employed in operational nowcasting systems [44].

However, these deterministic models output a precipitation map for future events, directly predicting the cell-wise precipitation probability [49]. This inhibits these models' applicability to operational use as the nowcasts they produce are blurred and struggle to resemble real-life precipitation scenarios. These blurred predictions are not suited for quantifying the uncertainty of the produced nowcasts. A better approach is to utilize probabilistic *generative models*. These models are able to generate a distribution of realistic precipitation scenarios and quantify the uncertainty based on this distribution [36, 39].

The last couple of years has seen significant improvements in this area of generative methods. One of the most impactful breakthroughs was proposed in a 2020 paper by Ho et al, presenting Diffusion Models (DM), further improving the Denoising Diffusion Probabilistic Models (DDPM) first initialized by Sohl-Dickstein in 2015 [18, 48]. At first only of interest to small scientific research fields within ML, before exploding in popularity even among the public through image-generating frameworks like *Dall-E* and *Stable Diffusion* [5, 50]. However, their possible use cases far extend the creation of images and art from text prompts that the above-mentioned frameworks offer. Nichol and Dhariwal 2021, presented several improvements to the DDPM, boosting its performance to exceed the state-of-the-art image generators known as Generative Adversarial

Networks (GANs) [8].

1.2 Research objectives and methods

In this master thesis, we suggest using DM for precipitation nowcasting. To the best of our knowledge, this is the first-ever research on using DM for this purpose and weather forecasting in general. Given the impressive image-generative abilities of DM, implementing a DM to perform precipitation nowcasting is a natural and interesting direction to explore. However, there are several challenges and methodological and implementation decisions that must be made before state-of-the-art DM can be used for precipitation nowcasting. We summarize this in the following four primary thesis research and development objectives:

Objective 1 Develop a diffusion model framework with the ability to *condition* the model output. As an *unconditional* generative model produces output that resembles random samples from the original data distribution used for training, their use in predicting future events is limited. The primary objective of this thesis is, therefore, to explore and develop methods for *guiding* the model to generate samples resembling a predicted future state. We will explore different methods to achieve this by feeding the network conditional input from the preceding time steps.

Objective 2 Explore and implement validation metrics to quantify the probabilistic correctness of nowcasts generated from the implemented diffusion models. As above-mentioned, the power of generative methods is their ability to generate realistic future precipitation scenarios that capture the characteristics and uncertainty in these future predictions. We will combine this probabilistic behavior with specifically developed validation metrics to quantify the uncertainty in predicted future events.

Objective 3 Explore model performance in different weather scenarios to identify potential challenges and pitfalls. Methods based on both optical flow and neural networks alike have shown strengths and weaknesses related to specific scenarios connected to non-linear convective precipitation and heavy rainfall. [37, 39, 49]. The developed framework will test its abilities in some of these scenarios.

Objective 4 Compile a dataset for model training and validation consisting of radar precipitation rates. Radar precipitation rates are represented as numeric values for a given location in a two-dimensional grid space, well suited to be represented as images. This data is provided by the Norwegian Meteorological Institute through an application programming interface (API) and must be extracted and processed by developing a series of scripts. These scripts extract relevant variables from a specified domain, perform necessary processing and transform the data into a data type suited for model training on high-performing graphical processing units (GPUs). The scripts can prove helpful in later weather forecasting operations as it enables an end-to-end solution for data extraction, processing, and real-time prediction.

1.3 Ethical considerations

Trained, generative models possess powerful abilities to generate samples resembling real-life scenarios. However, they also pose the risk of generating samples that are entirely wrong but still resemble a highly realistic-looking scenario. This can lead to situations where the model fails to correctly predict a future extreme precipitation event which can have substantial negative consequences. In contrast, the model can also overestimate predictions, resulting in false alarms.

We also note that the substantial computational resources needed to train these models can negatively impact the environment through increased energy consumption and emissions.

1.4 Main contributions

This study presents a novel approach to precipitation nowcasting, implementing a DM with conditioning abilities. This model generates realistic radar images of future precipitation based on past radar observations for lead times of up to 20 minutes. As pre-built model architecture was still sparse during the development phase of this project, a complete module had to be developed, inspired by concepts and implementations from various sources. The models are trained and validated on a compiled dataset with radar observations. For validating the nowcast quality, several probabilistic metrics commonly used in the field of meteorology were implemented. These metrics are applied to a range of weather scenarios to test probabilistic prediction capability.

1.5 Thesis outline

The rest of the text is organized as follows:

Chapter 2: Precipitation and nowcasting presents theory on the microphysical processes behind precipitation and outlines the two main types of rainfall, namely convective and stratiform. Presented is also theory on the state-of-the-art baseline method for predicting precipitation on short time scales.

Chapter 3: Deep learning and neural networks first presents the theory and concepts on basic neural nets to machine learning models used for image analysis. The second part of the chapter dives deeper into the main model architecture of interest for this study, namely *diffusion models*. At last, we present releated work in the field of machine learning for precipitation nowcasting.

Chapter 4: DiffMet: A diffusion model for precipitation nowcasting details all the application and development of methodology involved in the development of the framework used for precipitation nowcasting. We denote the framework DiffMet.

Chapter 5: Case study on nowcasting with DiffMet present the results from several experiments and use cases with DiffMet. Initially, we present results from early experiments conducted on a simplified dataset

before shifting focus to actual precipitation nowcasting with radar images as input data.

Chapter 6: Conclusion and future work presents a summary of the thesis, our findings regarding the skill of DM for nowcasting, and our conclusion. Finally we present our suggestions for future work.

Appendix A: Source code presents the GitHub repository for all source code behind DiffMet, in addition to data processing and extraction.

CHAPTER 2

Precipitation and Nowcasting

Beeing able to predict the weather accurately is of great importance, both for the public and for several sectors critical for society. More specifically, the ability to accurately predict rainfall intensities for a short time range, called *precipitation nowcasting*, is of great interest. These predictions are usually made on lead times from 0-6 hours and can significantly impact aviation safety, emergency services, agriculture, and flood warning systems, among others [56]. Today's operational nowcasting systems implement optical flow methods based on data from weather radar systems to capture these events.

The model presented in Chapter 4 will, among other things, have its abilities tested at two different types of precipitation. Hence, this chapter will present the theory describing the main mechanics behind rainfall and outline these two distinguishable types of rainfall. It will also present concepts and theories behind a state-of-the-art optical method for creating precipitation nowcasts.

2.1 Precipitation

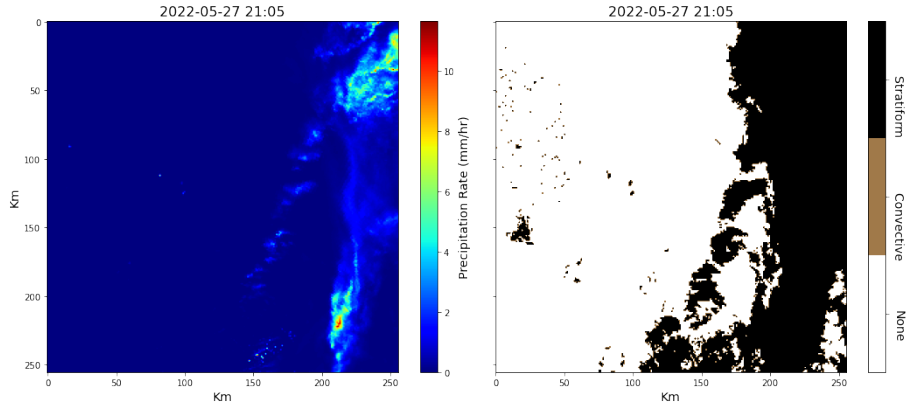

Figure 2.1: Two variables from a sample from the dataset presented in Section 4.1, visualizing parts of a stratiform precipitation region. Left: radar precipitation rates. Right: Separation of convective and stratiform precipitation. In the visible parts of this system, there are no convective regions.

The main focus of this thesis has been developing a novel approach to precipitation nowcasting. However, gaining a better understanding of some of the mechanics behind precipitation can prove useful when exploring where the model fails and where it succeeds. The main objective behind the developed nowcasting model, presented in Chapter 4, is to generate synthetic images for a given lead time, where the numeric values in the generated image represent precipitation intensities. This generation is done by feeding the model radar images with precipitation intensities from prior weather sequences. These intensities, projected on a two-dimensional grid, are the sole model input. However, the dataset used for model training contains an additional variable classifying the precipitation into one of two types. This classification enables further explorations on model performance as it allows for selecting subsets of the original test set. These subsets contain sequences with either exclusively one precipitation type or a mix of both.

This section presents these two types, namely convective and stratiform precipitation. These two distinguishable types are responsible for most precipitation in the water cycle but have several variations and are seldom present without the other. The selected domain contains little topographical variations as visualized in Figure 4.2. Orographic precipitation, the rainfall that occurs when moist air is forced upwards due to elevation in landmasses is, as a result, not discussed in this chapter.

2.1.1 Stratiform precipitation

Stratiform precipitation is produced in a cloud type named *Nimbostratus*. These clouds are formed by masses of thermodynamically stable air and are often spawned in multiples along frontal cloud systems. Visually, these clouds are typically grey and dense, often covering large areas. The precipitation that falls from Nimbostratus clouds is often long-lasting, caused by their internal air motion, which is usually slow compared to its convective counterpart. This air motion is critical for describing the mechanics behind the precipitation types, as it makes them clearly distinguishable. As the nimbostratus cloud is deep, the top level is high in altitude and therefore contains ice particles. Their slow-moving air motion, $|\bar{w}|$ can be defined kinematically compared to the fall speed usually found in ice particles,$V_{ice,typical}$, as defined by Houze jr.

$$0 < |\bar{w}| \ll V_{ice,typical} \tag{2.1}$$

This slow movement enables the ice hydrometeors in the cloud to increase in size and result in rainfall when exiting the cloud. This process can thoroughly be examined through a combination of radar and laser measurements and will be discussed in detail in the following sections. However, it is important to note that the air motion, $\bar{w}$ in Equation2.1, is denoted by its absolute value. This is based on the two sub-types of stratiform regions, defined by vertically upwards moving air or downwards. Equation 2.2 and 2.3 outlines both respectively.

$$0 < |\bar{w}| \ll V_{ice,typical} \text{ and } \bar{w} > 0 \tag{2.2}$$

$$0 < |\bar{w}| \ll V_{ice,typical} \text{ and } \bar{w} \leq 0 \tag{2.3}$$

For the air motion $|\bar{w}|$ to satisfy Equation 2.2, the overall vertical speed must be positive, resulting in an average upward motion in the region of interest. If this

is satisfied, the region is considered an active stratiform region. Equation 2.2 defines the opposite case, namely an inactive region where the overall motion of air is downward. In the latter's case, the conditions for particle growth are *not* present, and rainfall from these regions is defined as *fallout*. Explaining the mechanics of the opposite, active region, is therefore of greater interest, as the mean upward air motion in these regions allows for particle fall speeds that facilitate growth. For an active region, the height of a typical nimbostratus cloud structure usually allows the falling ice particles to grow for one to three hours. During this period, the particle goes through several distinct microphysical processes for growth. As each process will have different effects on particle radius, structure, and fall speed, it is possible to identify them with changes in radar reflectivity.

2.1.2 Convective precipitation and its presence in stratiform regions

As the previous section outlined the microphysical processes of growing particles falling slowly towards the ground, this section deals with precipitation that grows while moving *upwards*. The vertical air motion in these convective cells far exceeds the constraints set in Equation (2.1), which in turn is the key driver for growth. This rapid mean vertical air motion is caused by surface heating, lifting the air, enabling the right conditions to create cumulus- and cumulonimbus clouds. This powerful updraft will lift the particles and allow them to grow by gradually accumulating water. These lifting particles can be either ice or liquid water and will continue their journey upwards until they grow too large, which at that point, they will fall to the ground as rainfall. In contrast to their stratiform counterpart, the period of convective precipitation is a short-lived cycle. The strong updraft enables the creation of clouds and the following rainfall to be completed in ≈ 30 minutes. This distinct difference also enables the classification of precipitation type from radar echoes. At its simplest, the separation can be made based on the distinct vertical shape of maximum reflectivity caused by the solid convective updraft. Algorithms developed for separating precipitation types based on radar echos base their separation on further criteria on radar intensities and differences[51]. As the rainfall from convective cells is so distinguishable, all precipitation *not* classified as convective is automatically classified as stratiform.

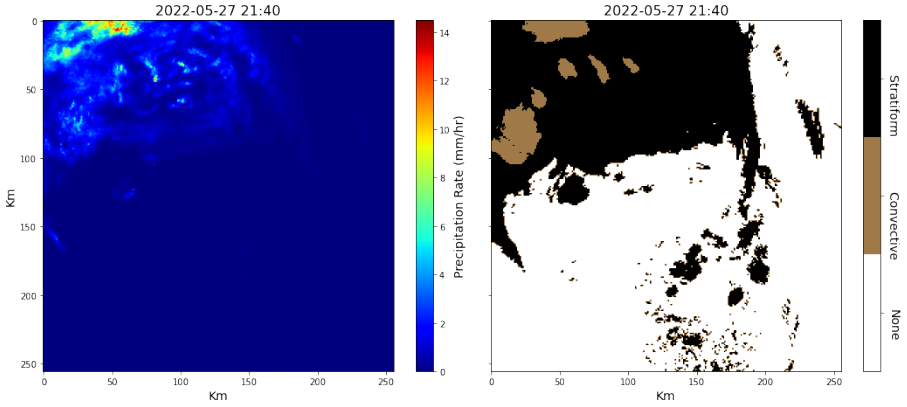

Figure 2.2: A sample from the dataset presented in Section 4.1, visualizing an occurrence of both stratiform and convective precipitation. Left: radar precipitation rates. Right: Separation of convective and stratiform precipitation. From the plot of the separated precipitation types, several generating convective cells can be seen, surrounded by stratiform precipitation. This coincides with the symbiotic relationships described in Section 2.1.2. The plot of radar precipitation rates also visualizes the high-intensity rainfall that convective precipitation can present, visible at a maximum rate of 14 mm/hr.

As above-mentioned, both convective and stratiform precipitation may occur together in the same cloud system and can happen in several different scenarios. To differentiate between these scenarios, a vertical profile of radar echoes of the region is often required, in addition to a time-height cross-section. However, the dataset available in this project only contains a two-dimensional array where each element corresponds to the precipitation type at a given spatial location. These precipitation types are illustrated in Figure 2.1 and Figure 2.2. To conclude the type of entangled convective and stratiform precipitation phenomena present in a given radar image is hence, impossible. There is, however, practical to outline some overall concepts on the most common ways these systems form relationships, as it is present, one way or another, in a significant part of the dataset.

One of these co-occurrences of convective and stratiform precipitation presents a relationship where the former adds precipitation to the latter by adding ice particles above the stratiform cloud structure. This phenomenon is called *stratiform precipitation with shallow overturning convective cells aloft*. This process is defined by the presence of a convective cell in the stratiform cloud deck and can occur in frontal clouds. The convective cell present in these regions has the same powerful updraft described above, enabling rapid growth of ice particles, which later fall out of the generating cell and into the cloud below. This updraft will, in turn, lead to greater precipitation rates from the stratiform system as these particles will go through parts of the microphysical processes outlined in Section 2.1.1. These generating convective cells are classified as *weak*, but a relationship defined by *deep* convection also exists. In these deep, generating cells, the strong updraft carries air to high altitudes, where the decrease in pressure causes a horizontal widening of the updraft. This causes a phenomenon called a *particle fountain*, where the lateral movement of particles

away from the updraft can cause the creation of nimbostratus clouds and stratiform precipitation [20].

As presented above, the physical processes behind the two presented precipitation types differ and occur at different time scales. Hence, one may expect that the model presented in Chapter 4 will achieve different results based on the type of predicted rainfall prior to lead times. This will be further examined in Section 5.2.3.

2.2 Baseline method for precipitation nowcasting

As made clear in the preceding sections, precipitation is a highly complex phenomenon controlled by various microphysical processes. *Nowcasting* is defined as the process of describing these weather systems and predicting the change over a few hours' time scale [56]. These nowcasts emphasize weather occurring on the mesoscale, defined as phenomena with horizontal scales ranging from a few to several hundred kilometers [12]. As these nowcasts can predict a wide range of weather scenarios ranging from thunderstorms, wind, and fog, this section will focus on methods used solely for nowcasting precipitation. These models differ from the numerical models typically used for forecasting weather over more extended periods, as the former relies on heavy computations based on a comprehensive combination of data sources. Methods used in nowcasting, however, advect precipitation fields based on radar observations. One of the most widely used frameworks for this short-range precipitation nowcasting is PySTEPS. This framework predicts future precipitation events by estimating optical flow, with radar data as input. This is done by first calculating the displacement of a precipitation parcel, R, given by the conservative equation for incompressible flow outlined in Equation 2.4.

$$\frac{dR}{dt} = \frac{\partial R}{\partial t} = u\frac{\partial R}{\partial x} = v\frac{\partial R}{\partial y}, \qquad u = \frac{dx}{dt}, v = \frac{dy}{dt} \tag{2.4}$$

where dR/dt = 0 is assumed, and u and v are the x and y components of the motion field [37]. The complete nowcast uses this motion field and extrapolates radar data using an advection method. This produces a deterministic nowcast for a given lead time. However, there are several predictive uncertainties connected to weather forecasts in general, stemming from reasons like the chaotic nature of the atmosphere, approximations to physical laws, and general model errors [30]. This creates the need for an ensemble of predictions, reflecting these uncertainties. To produce this ensemble, PySTEPS perturb the determinist forecast with noise through stochastic simulations.

However, due to the highly non-linear processes of precipitation, achieving appropriate ensembles proves challenging for several scenarios. This raises interest in developing alternative ML approaches to mitigate these non-linear challenges.

CHAPTER 3

Deep learning and neural networks

The previous chapter presented the theory behind precipitation and how this can be forecasted using optical flow methods. However, recent years have shown considerable efforts from the scientific community in developing weather forecasts produced by neural networks. The Convolutional LSTM presented by Shi et al. in 2015 provided results that outperformed several optical flow methods and gave rise to several advancements by Google Research and Deepmind using neural network-based methods [39, 44, 49].

In this chapter, Section 3.1 presents the theory and concepts behind neural networks, starting with a simple perceptron before moving toward complex architectures used for image analysis. In Section 3.2 introduce the theory behind diffusion models, a recently presented model architecture for image generation. The relevant metrics specifically used for nowcasting are presented in Section 3.3. Finally, Section 3.4 presents releated work in the field of nowcasting with ML.

3.1 Neural Networks

Machine learning is a rapidly growing field that combines informatics, statistics, and mathematics to build models that learn from data. While machine learning is a rapidly advancing field, its roots date back to the 1940s [31]. As today's model architectures and capabilities far surpass the ones originally developed decades ago, the core idea can still be described in the same way as they were in 1959, by A.L. Samuel, as "programming of a digital computer to behave in a way which, if done by human beings or animals, would be described as involving the process of learning" [41]. For the scope of this thesis, the application of interest is images. This chapter will therefore start by outlining a broader, more general description of machine learning and its most elementary components before shifting focus to what makes up a state-of-the-art model used for image applications.

3.1.1 An overview of machine learning

As decades have passed since the birth of the first machine learning models, the field has grown to contain a wide variety of models, commonly divided into three main sub-fields. The models in the sub-field called supervised learning commonly refer to models that assign class labels to the testing instances where the values of the predictor features are known, but the value of the class label

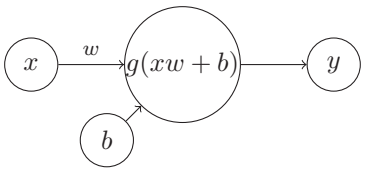

Figure 3.1: A single perceptron, with an input node x, a bias b and weights.

is unknown [25]. At training times, these models make use of already labeled data to guide the model's parameter. In this category, we find classical machine learning models like linear regression, support vector machines, and neural networks.

In contrast, the second sub-field deals with unlabeled data and is called *unsupervised learning*. As there is no labeled data to guide the learning, these models make predictions based on patterns in the data, often defined by the distance between data points. The third sub-field describes models that steer the learning process by an agent that rewards choices made by the model. This type of learning is defined as *reinforcement learning*. However, model development over recent years has shown increasing use of model architectures that combine several of the above. Examples are the self-supervised transformers that are fine-tuned with supervised learning or unsupervised generative models that implement cost functions usually found in supervised learning models [13, 54].

3.1.2 Neural Networks

The artificial neuron

Today's neural networks are massive technological workings, consisting of layer upon layer of functions. At the core, however, the most basic unit of the net is based on the workings of the human neuron. The mathematical derivation of these artificial neurons was proposed by McCulloch and Pitts [31] in 1943. The core idea that the artificial neuron mimics is that the neuron gets activated differently depending on the input. Adjusting this activation level is the core element of interest when training these networks. This subsection will outline the functionality between the simple single perceptron, how it can be arranged in layers of several perceptrons, and how this net of neurons is collectively trained on labeled data.

Passing data through a perceptron

To control this activation, each neuron's output will be a function of the input, weight, and a bias term. For a single perceptron with a single input value x, the first computed value is then simply the product of x and the weight value w, with the bias term, added. Figure 3.1 illustrates this process. The bias term is added to offset the neural network results to match the wanted output better.

The real power of the perceptron is, however, the activation function that follows. The concept behind this function is the introduction of nonlinearity. Without this activation of the node output, the network's abilities will not be able to succeed in the capabilities of a linear regression model [43]. This is due to the fact that the result of a combination of linear functions is a linear

function itself. This will greatly inhibit the abilities of the network to learn complex data patterns and hence limit its use in applications.

Furthermore, many of these non-linear activation functions also serve the purpose of a way to limit the output value of the node to stay within a fixed interval. This is an important factor in the training of the neural network, as large output values can quickly destabilize the training and optimization of the network. Depending on the problem and network architecture, different activation functions can lead to differences in performance. The function used in the final layer is also often tailored to fit the labeled output, like a logistic sigmoid function. This function will squash the output in the interval between zero and one and enable the network to produce a probability connected with the input.

Multilayer Perceptron

This section has, until now, described the workings of a single perceptron. The neural network, however, is, as the name suggests, a network of neurons connected in layers. Each added neuron and layer has the *potential* of making the network able to learn more complex data. Equation 3.1 and 3.2 derive the computation of the output value of a node y_j connected to d nodes from the previous layer. The resulting output is the activated result, y_j of the sum, h_j of the products of the value between the $i-th$ weight and output value, added with the bias term

$$h_j = \sum_{i=1}^{d} z_i w_{ij} + b_j \tag{3.1}$$

$$y_j = g(h_j) \tag{3.2}$$

These networks are often referred to as fully connected feed-forward neural networks visualized by figure 3.2. These networks will have a fixed input size, representing the number of features in the data set. This input data is then processed through the nodes and layers as described in the previous sections. The above-referenced figure demonstrates an output layer consisting of a single neuron, but this number is tailored to suit the use case.

Updating the weights

With the network architecture and the math behind the output derived, the network is ready to start the actual learning. As input to the network, the training data is arranged so that every input x_i has a matching observation y_i. How well the network performs on a given task is then done by comparing the computed output in the final layer with the corresponding observed data. This is computed by applying a *loss function* with these values as input. Equation 4.2 outlines a loss function called *Mean Squared Error*, a widely used function applied in various implementations. This function takes the mean of the squared sum of the difference between the actual value y and the predicted output y_i for all N samples.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i) \tag{3.3}$$

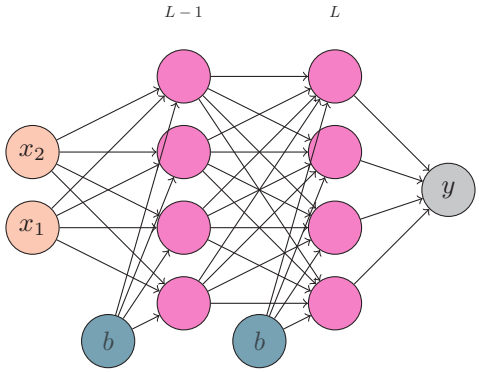

Figure 3.2: Multilayer perceptron with two input nodes, two hidden layers and one output node

However, computing this loss on every single input value independently is inefficient both in terms of computational expenses and convergence. In reality, this loss function is applied on *mini-batches* of training data. These mini-batches consist of partitions of randomly shuffled data and have the effect of each mini-batch creating a single numeric value to measure how well the network performed. To understand why this aids the network in learning, we must see how the network parameters are updated through the concept of *stochastiggradient descent* and *backpropagation*. These processes aim to guide the network in how the weights connecting the nodes and biases are updated since these are the only trainable parameters of the network. For every mini-batch of data, the cost functions evaluate every predicted output against the desired output. As a result, the mini-batch creates a collective measurement of how weights and biases should be updated, averaged over all training examples within the given batch. *How* these weights and biases should be updated is calculated through *backpropagation*. As presented in Section 3.1.2, the output of each neuron is a function of both weight, bias, the previous input, and an activation function. The core mechanic behind backpropagation is applying the *chain rule* to identify the optimal change in these weights and biases regarding the loss. Hence, backpropagation enables us to find a gradient, ∇C, that contains the derivatives explaining these updates as partial derivatives of the cost function.

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial \omega^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial \omega^{(L}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix} \tag{3.4}$$

This gradient is the best suggestion based on the given data partition from the mini-batch. Equation 3.4 outlines the gradient vector consisting of the partial derivatives of all weights and biases connected to L nodes.

3.1.3 Machine learning for image analysis

As Section 3.1 outlined a neural network architecture with input based on a set of features, this section will focus on model architecture suited for input in a more complex manner, namely images. As technological improvements during recent years have brought both image quality and the availability of camera technology to new heights, the need for intelligent systems to analyze this data is of great interest. The use of machine learning for this task is already widespread in several fields. Neural networks are now beeing used to aid medical personnel in diagnostics, for remote-sensing and monitoring earth processes, and in self-driving cars [3, 4, 63].

CNNs in machine learning

Non-image, tabular data sets contain attributes and observations making up features. Building a good-performing model is then a process of selecting the most relevant features. For images, however, this process is not as straightforward, as the image and its channels are the only data available. For a model to make meaningful predictions and classifications based on images, the input has to be transformed and processed into feature maps. In traditional image analysis, these feature maps were often engineered by carefully selecting a set of different mathematical operations. These operations acted as filters, extracting specific information from the original image. The response from the filtering operations would be stored in a new image consisting of statistical and geometrical descriptors. These descriptors would then make up the feature images and be used as input to a model classifier with trainable parameters.

As both developments in model architecture and computational power increased, new methods for computing these feature maps were developed. In 1998, a model architecture called convolutional neural networks (CNN) where proposed. This network brought several advances to the field of machine learning and image analysis. The key conceptual idea where a scheme that relied as much as possible on learning in the feature extractor itself [28]. This was done by using layers of convolution kernels with trainable weights, in addition to layers of sub-sampling. Figure 3.3 presents a visualization of a convolutional kernel with computed output. The big advance in the use of CNNs, however, came over a decade later, with the introduction of AlexNet[27] in 2012. This major leap was made possible by the advances in hardware, as the network was trained on several high-performing GPUs. This network was the first of its kind to win the LSVRC-2010 contest, achieving top error rates on the classification of 1.2 million high-resolution images into 1000 different classes.

As the field of machine learning is expanding and new model architectures are introduced, a large number have elements based on one of these exact network architectures. This section will outline the theory and concepts behind CNNs before shifting focus to a specific type often used in the diffusion models presented in 3.2, called U-Net.

The concept of Convolution

As described in section 3.1.3, data sets containing images are fundamentally different than data found in regular tabular data. Regular digital images can be represented numerically as a 3D array, where the first two dimensions represent

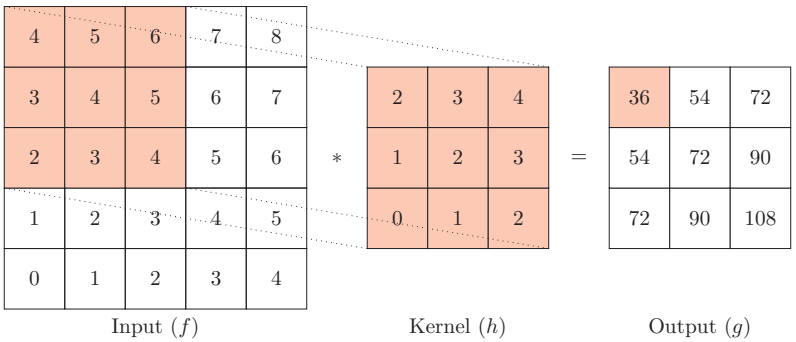

Figure 3.3: Visualization of a 5x5 matrix being convolved with a 3x3 kernel, stride equal to 1. As there is no padding on the on border of the input matrix, the resulting output is a 3x3 matrix.

the height and width of the image, while the last represents the channels. A typical optical image has 3 channels, representing the level of Red, Green, and blue (RGB) at each pixel index. A simple, naive solution on how to feed this data to a regular fully connected network, would be to flatten the dimensions to a long one-dimensional array.

This would, however, presents several challenges for the network. For one, this would lead to important spatial information in the input being lost, as the position of the indices relative to each other is vital. Secondly, unstructured nets for image or speech applications have no built-in in-variance with respect to translations or local distortions of the inputs [28]. These nets will therefore struggle to generalize on the objects of interest. LeCun 1998, uses the case of handwritten digits to demonstrate this problem, as there are numerous ways a single digit can be written in terms of writing style, size, slant, and position variations. This can be extended to all other objects represented in images. Lastly, this approach of a long, flattened input array will force the network to deal with an input layer that increases exponentially in size as the resolution and size of the image increase. A 256×256 input image with 3 color channels will have an input layer consisting of 196.608 nodes, making it extremely computationally expensive to train.

As a result, using convolution is a means to deal with these challenges. At its core, convolution is a simple mathematical operation. It consists of using a kernel to filter an input image. Figure 3.3 visualizes a discrete convolution with a 2-dimensional input, also formulated in equation 3.5. The response g for a pixel at position (x, y) is computed by positioning the filter h such that its origin is overlapping (x,y) in f. The overlapping values of the kernel and the input image are then multiplied before all products in the overlapping area are summed.

$$g(x,y) = \sum_{s=x-a}^{x+a} \sum_{t=y-b}^{y+b} h(x-s, y-t) f(s,t) \tag{3.5}$$

Figure 3.3 visualizes a convolution with two-dimensional input. It is important to note that the depth of the kernel should match the depth of the input image, so in the case of input with RGB dimensions, the kernel will

be three-dimensional with a depth of three. Furthermore, the size and stride length of the kernel will dictate the width and height of the output image. This stride is defined as the step-by-step displacement in the horizontal and vertical direction of the kernel after each convolution operation.

If the input image is kept as is, it will inevitability have its width and height reduced. This can be avoided by using various techniques to pad the outer border of the input. Widely used padding operations are "zero-padding", where padding is applied, only consisting of zeros, or "mirror padding", where the n values near the edge are mirrored in the padding. Equation 3.6 and 3.7 describe formulations on how the output shape in terms of height and width can be computed[1]. The indexing of the *padding*, *kernel_size*, and *stride* relates to the height and width dimensions, respectively

$$H_{out} = \left[\frac{H_{in} + 2 \cdot \text{padding}[0] \cdot \text{kernel\_size}[0] - 1}{\text{stride}[0]} + 1 \right] \tag{3.6}$$

$$W_{out} = \left[\frac{W_{in} + 2 \cdot \text{padding}[1] \cdot \text{kernel\_size}[1] - 1}{\text{stride}[1]} + 1 \right] \tag{3.7}$$

Convolutional layers in CNN

Section 3.1 outlined a multilayer perceptron consisting of nodes and weights, where training the network resulted in updates to the values of the weights connecting the nodes. For convolutional neural networks, the update is in the kernel values. As the network and kernels are initialized, all kernels will have values drawn from a random distribution. As training progress, the goal is to have kernel values converge to values where each layer in the network can detect the different level of details.

A well-trained network will have a first layer of kernels trained to detect the general shapes and characteristics. These can be shapes like lines, curves, edges, and colors. As the depth of the network increases, each layer will bring increased attention to specific details of the objects of interest. Each layer of convolutions produces a set of feature maps, which in turn will be used as input to the next layer of convolutions. The subsequent layers combine these features to detect higher-order features[28]. This is done through the concept of the *receptive field* that each convolutional layer holds, as input to each layer is the feature maps produced in the layer before. This enables the deeper layers to effectively process an increasingly larger portion of the original input image. The logic behind this technique is that each kernel will be optimized to detect specific features present in varying parts of the input. As the kernel strides across the input, the local receptive field for each position will use the same optimized kernel.

Pooling operation

In addition to convolution, each layer also contains a sub-sampling component. This component has the objective of reducing the spatial dimension of the feature map. This is done to reduce the precision with which the position of distinctive features are encoded in a feature map[28]. These are rather simple

[1] https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

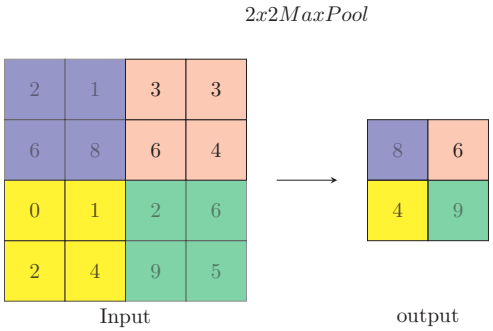

Figure 3.4: Visualization of a 2x2 max pool operation applied to a 5x5 input matrix. The resulting output matrix has colors matching the area of the input matrix where the pooling has been applied.

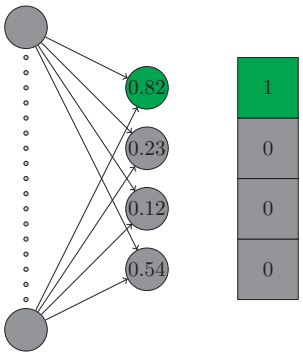

Figure 3.5: Visualization of the final layer of a CNN used for a 4-class classification problem. The network will assign the correct label as the output node with the highest probability coincides with the index of the class in the one-hot-encoded array(marked in green).

operations and mainly work by applying a sliding filter in the same way as outlined in 3.1.3. Figure 3.4 visualizes applying a 2x2 *max pooling* on an input image. The response is the maximum value contained in the input in the area of the overlapping parts between the input image and filter. As the filter response is a single value, the size of the kernel is strongly connected with the shape of the output. Figure 3.4 presents a visualization of a two × two max pool operation applied to a five × five input matrix and the resulting output.

Fully connected layer

After processing and transforming the input through all the previously mentioned filters and operations, the network architecture returns to a more traditional form. In the final parts of the CNN, the convolution layer is connected to one or several fully connected layers. If the network is constructed for classification, the final output size of the last layer will be a one-dimensional array matching the number of classes of the data set.

The numeric values of each node will then be processed through an activation function. In the case of classification, this will often be a Sigmoid function,

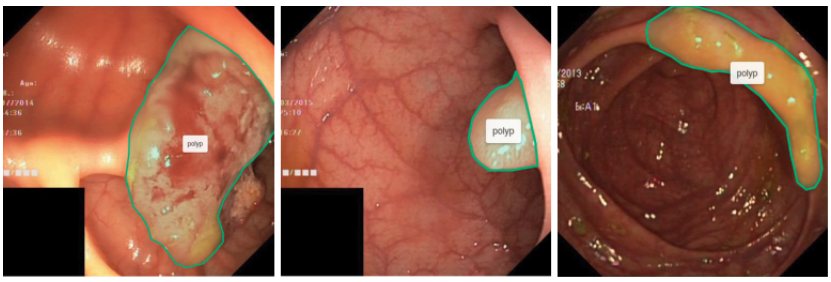

Figure 3.6: Gastrointestinal image, where the pixels containing abnormal tissue growth(polyps) have been assigned a label. This creates a mask for the images used in the training phase of the model, guiding the network to output a numerical value for each pixel. For a well-trained model with good performance, this value coincides with the ground truth class label[23] .

squashing the numeric output value in the interval between 0 and 1. This will represent the probability of an input belonging to the given class. This is then compared to the values of a one-hot-encoded label array. This is a $1 \times n$ dimensional array, where n is the number of classes. This is illustrated in Figure 3.5, where the network will assign the input image to the label corresponding to the index with the highest probability.

3.1.4 U-Net

As mentioned in the previous section, CNNs are widely used in medical imaging. This field deals with large amounts of images from advanced imaging techniques like magnetic resonance imaging and x-ray to traditional optical images. These images are then used to diagnose diseases and conditions in patients. This process requires highly trained medical personnel to pay great attention to fine details, which can often lead to conditions being misclassified or overlooked[23].

To aid in this process of image-based diagnosis, neural nets were introduced with the task of performing image segmentation. The goal of segmenting an image is to provide an output where a class has been assigned to each pixel [40]. This contrasts the classification-based CNNs discussed in Section 3.1.3, where a label is assigned to the picture as a whole. Developing efficient model architectures for this pixel-wise classification has historically demanded large data sets of annotated images and significant computational resources. To deal with these challenges, a model architecture called Unet was introduced in 2015 [40]. This network has since gained a lot of popularity and is now being used in a wide variety of tasks, far extending the original use case of biomedical image processing.

For the scope of this master thesis, the main goal is to have a network able to create a mapping pixel-by-pixel that translates precipitation rates at previous time steps into a prediction of future time steps. This relationship between input and output dimensions, therefore, makes the U-net architecture the perfect tool for the job. The following section will describe the architecture and concepts of the first presented architecture, further extending the components from Section 3.1.3.

Architecture

At a high conceptual level, the traditional U-net consists of two parts; The contracting and the expansive parts. Figure 3.8 illustrates how the Unet got its name, as the input is transformed through these components. The contracting part shares many similarities with a traditional CNN and can be seen as the *downsampling* part of the network. This part consists of repeated blocks with three main components. First, there are one or multiple convolutional operations. The feature maps produced by these are further processed by a rectified linear unit activation function as defined in Equation 3.8.

$$f(x) = max(0, x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

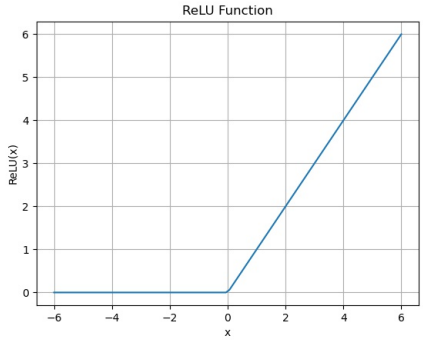

Figure 3.7: Visualization of the ReLU (rectified linear unit) activation function, on input from the interval of minus six to six.

These are un-padded convolutions, resulting in a reduction in the width and height dimensions. However, the number of convolutions decides the number of feature channels. A traditional approach is to adjust the number of convolutions so that the number of channels for each described block is doubled. The network then stores a copy of each feature mapping, later to be added to the corresponding block in the other half of the network.

After the network has reached the desired number of downsampling blocks, a new type of mathematical operation is introduced, namely the transposed convolution. This operation can transform the image dimensions in the opposite direction compared to the standard convolution, in other words, increase the resolution[9]. The effect of this is a reduction in feature channels while increasing the height and width of each mapping.

The before-mentioned copy of the feature mappings from the downsampling is also concatenated with the output from these transpose convolutions. This concatenation of feature mapping from the contracting part of the network enables precise localization and lets the network effectively propagate contextual information. A set of normal convolutions with ReLu-activated output follows this.

There are no fully connected layers in these networks, and the final output with the wanted width, height, and the number of channels is produced by a final

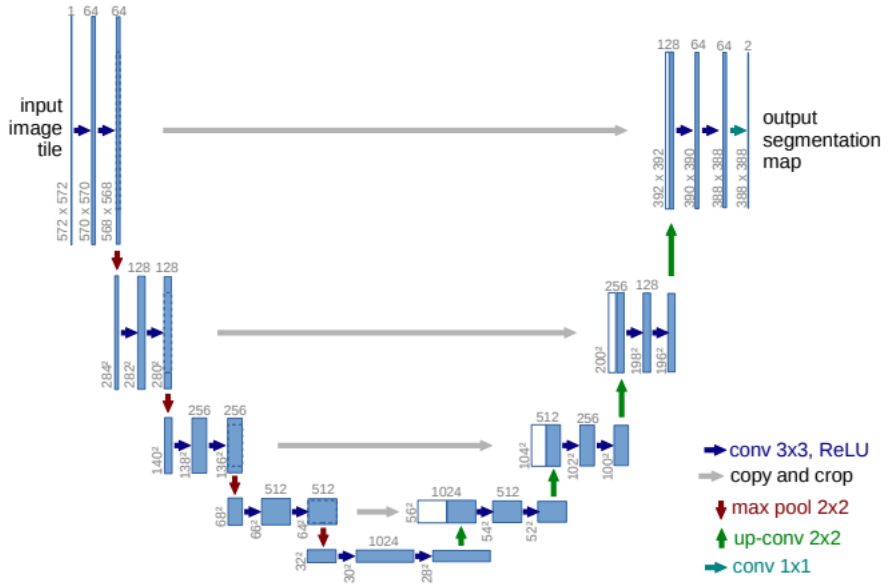

Figure 3.8: The original U-net architecture [40]. An input image of size 572x572 is passed to the network. The original image contains only 1 channel but is increased for each layer in the network using convolutional operations. Combined with max-pooling, this also results in the reduction of image height and width. After passing through the middle part of the network, often called "the bottleneck", the channels are reduced and the image size gradually increases back to its original dimensions. This architecture is often visualized as a "U", which gives the neural network its name.

convolution. The concept of this final operation is to transform a N-dimensional feature mapping to the final output channel dimension.

3.2 Diffusion Models

During 2022, generative models gained a lot of interest, both in mainstream media and in various scientific communities. These generative models could broadly be divided into two groups. Large language models (LLM) that are able to both interpret and output natural language at a high level and models that generate images from text prompts. This section will focus on the latter. These models are complex systems that consist of multiple types of networks to be able to transform language into images. However, the core mechanism for generating diverse samples of high-quality images is *diffusion models*. While the first seminal papers showed hints of potential, these models soon proved their performance and quickly gained status as real challengers to more established generative models, like GANs [8, 18].

As the development of diffusion models continues, we continue to see their use increase for solving challenges with complex, real-life data sets. They are now being utilized in a wide variety of fields and applications, including medical

imaging, remote sensing, and time-series forecasting, among others [2, 38, 58]. However, their use in precipitation nowcasting is yet to be explored. This section aims to outline the inner workings of these models, ranging from architecture to mathematics and the functionalities of its components.

3.2.1 Concept

The concept of diffusion models was first presented by Sohl-Dickstein et al. in a 2015 paper called *Deep Unsupervised Learning using Nonequilibrium Thermodynamics* [47]. This paper introduced a novel approach for modeling probability distributions inspired by non-equilibrium statistical physics. This approach was later improved upon by Ho et al. in 2020 [18], which led to a rapid advance in the capabilities of diffusion models.

At a high level, the main concept behind diffusion models is the idea of step-by-step destroying samples from the training data by adding Gaussian noise. In the final steps of this process, the amount of noise added will reduce the input image to nearly isotropic Gaussian noise. This process is illustrated in Figure 3.9. A neural network is then introduced with the goal of rebuilding the image in the same step-wise manner as the noising process. After the iterative training process is completed, the neural network should then be able to take as input random sampled Gaussian noise and generate samples similar to the training data by a learned reverse process.

3.2.2 Forward diffusion process

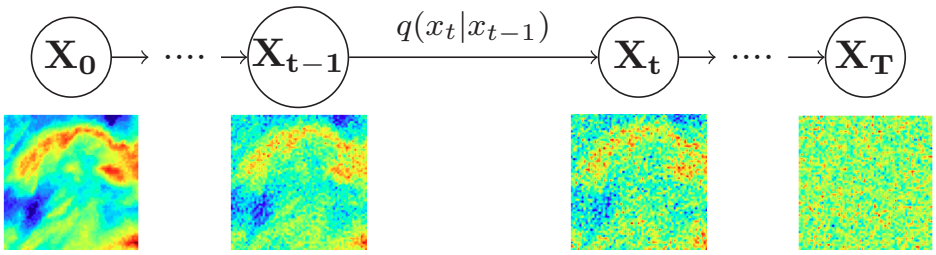

Figure 3.9: Visualization of how a sample from the compiled radar precipitation data set is gradually 'destroyed' by adding increasing amounts of Gaussian noise. X_0 is the input image. Noise is gradually added step by step T times. X_T represents the image at T-th timestep when the input image is reduced to nearly isotropic Gaussian noise. q represents the forward diffusion process(FDP). Inspired by Ho et al. 2020[18]

At a deeper level, these models can be referred to as latent variable models, where the noised images are latents of the same dimensionality as the original image [18]. This before-mentioned process of adding noise to the input data is defined as the *forward diffusion process* (FDP)

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{3.9}$$

The FDP derived in Equation 3.9 can be seen as the amount of noise in x_t, given an image with a lower amount of noise, x_{t-1}, for the distribution q. β

represents a variance schedule that dictates the amount of noise added at time step t. $\sqrt{1-\beta_t}\mathbf{x}_{t-1}$ is the mean of the distribution, while $\beta_t\mathbf{I}$ is the variance. $\mathbf{I}$ represents the identity matrix and represents multiple dimensions. Ho et al. [18] proposed a linear variance schedule, with a constant increase from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$, over $T = 1000$. This scheduling was later improved using a cosine schedule[33]. The cosine schedule has the benefit of adding less noise at the start of the process and has been shown to boost model performance for input images with a resolution of 64×64 and smaller. A visual representation of both schedules is presented in Figure 4.5.

This noising process is formulated as a Markov chain, implying that the distribution $q(x_t)$ only depends on $q(x_{t-1})$. However, this might leave an impression that one needs to compute all previous distributions, $q(\mathbf{x_{0:t-1}})$, to get $q(\mathbf{x_t})$. This would create a massive computational expense for the already mentioned 1000-time steps. However, Sohl-Dickstein et al. showed that this could be avoided by reformulating β:

$$\alpha_t = 1 - \beta_t. \tag{3.10}$$

$$\bar{\alpha_t} := \prod_{s=1}^{t} \alpha_s \tag{3.11}$$

Equation 3.9 is then rewritten using the newly introduced α :

$$\begin{aligned} q(\mathbf{x}_t|\mathbf{x}_{t-1}) &= \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \\ &= \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon \\ &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\epsilon \\ &= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon \\ &= ... \end{aligned}$$

where ϵ is sampled from $\mathcal{N}(0, I)$. If one proceeds to replace the chained α values with $\bar{\alpha}$, the final expression can be written as

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \sqrt{\bar{\alpha_t}}x_0 + \sqrt{1-\bar{\alpha_t}}\epsilon \tag{3.12}$$

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha_t}}x_0, (1-\bar{\alpha_t})\mathbf{I}) \tag{3.13}$$

This enables the model to sample noised images from a given time step, t, during training, without calculating all the previous recursive steps. This is a major contribution to manageable running times during the training, discussed further in Section 3.2.4.

3.2.3 Reverse diffusion process

As Section 3.2.2 outlined the process of gradually destroying the training data, this section describes the reverse process: how to remove noise. This process involves a neural network to approximate a distribution, defined as p, with a neural network with trainable parameters θ, which is trained to learn the transitions between time steps t and $t-1$. The learning goals of the network are, therefore, to learn the distribution defined as:

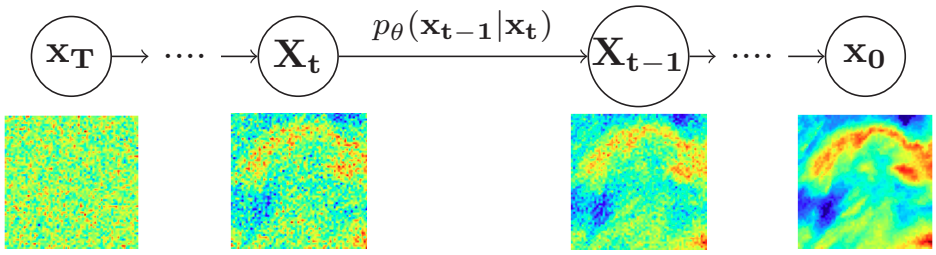

Figure 3.10: Visualization of the reverse diffusion process. p_θ denotes a distribution p approximated by a model with trainable parameters θ that can learn the transitions between a noisy image $\mathbf{x_t}$ and the less noisy image, $\mathbf{x_{t-1}}$. Inspired by Ho et al. 2020[18]

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{3.14}$$

As done by Ho et al., the model presented in this study has fixed variance. As a result, the training objective of the neural network is to learn the mean of this distribution. Initially, training was done based on the resemblance between p and q and variational autoencoders. This motivated the authors to optimize the variational lower bound to minimize the log-likelihood of the training data. However, through a series of parameterizations, they derive a loss function that learns to predict the noise, ϵ, instead of μ. This derived training objective is further simplified, as Ho et al. found empirically that a simplified variant could generate better-quality samples and simplify the implementation process.

$$L_{simple}(\theta) := \mathbb{E}_{t,x_0,\epsilon}\left[||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||\right] \tag{3.15}$$

Equation (3.15) presents the final training objective. From this equation, one can recognize $\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ from Equation (3.12) as the final expression defining the sampling of a noised sample from a given time step. With this in mind, the final loss objective can then be interpreted as the L2 distance between the actual noise ϵ and the noise generated by the parametrized model, ϵ_θ for a given timestep t, rather than the mean.

3.2.4 Training

With both the forward process, reverse process, and loss function defined, an algorithm for training the model can be defined, see Algorithm 1 [18]. A training sample x_0 is drawn from the original distribution $q(x_0)$. A random t-value between 1 and T is drawn, representing the time step. As illustrated in Figure 3.9, the level of noise increases in proportion to t. $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ represents the sampled noise from the Gaussian distribution. This ϵ is added to the input sample x_0 according to the variance schedule described in 3.2.2. The parameterized model will then predict the noise, ϵ_θ, and compare it to the actual noise, ϵ. This process is repeated until converged. Figure 3.11 visualizes one iteration in the for-loop outlined in Algorithm 1. In practice, this process is performed on batches of samples, although for illustration purposes, this figure illustrates the process for only a single sample.

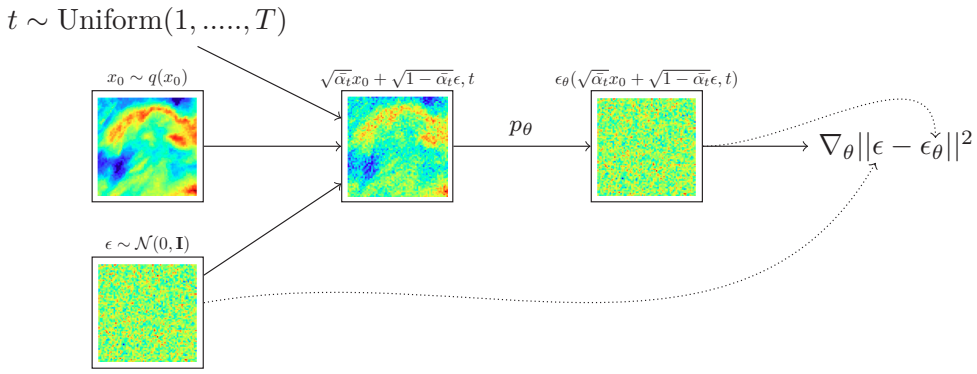

Figure 3.11: Schematic drawing of one iteration in the training of a diffusion model, based on Algorithm 1. From left to right: A random noise level, t, and Gaussian noise ϵ are drawn. These corrupt the sample from the original distribution, x_0. A neural network with trainable parameters, θ, is trained to predict the noise added to this corrupted image.

Algorithm 1 Training

repeat
- $x_0 \sim q(x_0)$
- $t \sim \text{Uniform}(1, ..., T)$
- $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
- Take gradient descent step on $\nabla_\theta ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}}\mathbf{x_0} + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2$

until converged

3.2.5 The parameterized model

The U-Net discussed in 3.1.4 is the most widely implemented neural network in diffusion models [8, 18, 33]. As discussed earlier, these models are the perfect choice when dealing with input and output images with the exact dimensions, as with diffusion models. This U-Net will take as input a corrupted image and predict the noise. However, the same neural network with shared parameters is applied for all t. Hence, the neural network depends on receiving information about the time step in question. This is because the amount of noise in an image will significantly vary depending on the t position in the $[1, T]$ interval. To solve this challenge, the U-Net architecture is also extended to handle input in the form of a sinusoidal position embedding [54]. This creates extra dimensionality and forms a positional matrix, enabling each t to be mapped to a specific vector containing the positional information.

3.2.6 Sampling post training

After convergence during training, the model should, in theory, be a noise predictor. This enables it to sample pure Gaussian noise and transform it into data points similar to the original distribution. Algorithm 1 outlined a process of randomly drawing noise levels which the network then made predictions based upon. However, the sampling process outlined in Algorithm 2 is a reverse

Algorithm 2 Sampling

$x_T \sim \mathcal{N}(0, \mathbf{I})$
for $t = T, \ldots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x_t} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)) + \sigma_t \mathbf{z}$
end for
return $\boldsymbol{x_0}$

iterative process from the maximum noise level, T, to 1. At first, pure Gaussian noise is drawn from the previously defined Gaussian distribution $\mathcal{N}(0, \mathbf{I})$. This sample, x_T, has the exact dimensions as the data points in the original training data distribution and acts as the starting point for the image that will be gradually denoised.

The reverse process consists of the network iteratively predicting the noise level in the image for the given time step, defined in the algorithm by $\epsilon_\theta(x_t, t)$. This predicted noise is multiplied with $\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}}}$ term, which relates to the variance schedule defined in Equation 3.10 and 3.11. This product is then subtracted from x_t. Together with an additional multiplication with $\frac{1}{\sqrt{\alpha_t}}$ and the addition of the posterior variance,$\sigma_t \mathbf{z}$, at time steps t, this produces the slightly less noisy x_{t-1}. It is important to note that the model always predicts the total noise in the image, regardless of t. This predicted noise is, however, scaled by the α-values, creating the gradual de-noising dictated by the β-schedule.

3.3 Verification Metrics

As the intended use of the implemented diffusion model in this study is the before-mentioned nowcasting of precipitation, two validation metrics tailored for this use case are presented in this section.

Continous Ranked Probability Score

As outlined in 2.2, there are fundamental differences between deterministic and probabilistic model output. The produced forecasts are usually connected with uncertainty to a varying degree, which in turn, creates the need for outlining a suitable metric to quantify this. *Scoring rules* have been developed for this quantification and are used to evaluate the accuracy of a forecast distribution given an observed outcome A scoring rule is hence a measurement of the performance of the forecast distribution, in contrast to the MSE defined above, where the output is the function of two points. In the scope of this thesis, the scoring rule of choice is the Continuous Ranked Probability Score (CRPS).

$$CRPS(F, y) = \mathbb{E}_F|X_1 - y| - \frac{1}{2}\mathbb{E}_{F,F}|X_1 - X_2| \tag{3.16}$$

$$CRPS_{NRG}(M, y) = \frac{1}{M}\sum_{i=1}^{M}|x_i - y| - \frac{1}{2M^2}\sum_{i,j=1}^{M}|x_i - x_j| \tag{3.17}$$

The CRPS is defined in Equation 3.16, where X_1 and X_2 are independently drawn from the predictive distribution F, and y is the observed radar image

at a given lead time [24]. When F is only known through M ensembles, the CRPS can be defined with the empirical formula presented in Equation 3.17 [62]. CRPS is negatively oriented, meaning lower is better.

Critical Success Index

The critical success index (CSI) evaluates a binary forecast and operates with a set threshold relating to precipitation intensities. CSI can be explained as the ratio of the number of hits, to the number of total events, to the number of false alarms [42]. This ratio is defined by Equation 3.18, for true positives, TP, false positives, FP, and false negatives, FN.

$$CSI = \frac{TP}{TP + FP + FN} \tag{3.18}$$

TP, FP, and FN are computed by first assigning a precipitation threshold value, t, for example, *medium rain* ($t = 5$mm/hr). The three variables are then computed for all i grid cells, where:

$$TP(F_i \geq t, O_i \geq t)$$
$$FP(F_i \geq t, O_i < t)$$
$$TP(F_i < t, O_i \geq t)$$

F_i denotes the forcasted value in grid cell i, while O_i denotes the actual observed [39]. CSI is a positively related score, meaning higher is better. Equation 3.18 makes it clear that this metric is closely related to traditional performance metrics like *precision* and recall. These are defined by $\frac{TP}{TP+FP}$ and $\frac{TP}{TP+FN}$ respectively.

3.4 Probabilistic models for weather forecasting

As noted earlier, diffusion models have, up to this point, yet to be explored for use in weather forecasting. However, several papers have been published in recent years, utilizing increasingly complex model architecture to better model performance. In this section, we present the most influential related work from the four most relevant papers. We will explain the main concepts behind each model and discuss the most important findings.

Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting

In 2015, Shi et al. proposed a Convolutional LSTM Network for precipitation nowcasting which was the first ML approach in nowcasting able to reach and exceed the performance of operational optical flow methods. The model is based around a modified version of *Recurrent Neural Nets* (RNN). These nets are especially well suited for handling temporal data as they display a cyclic architecture made up of *cells*. Each cell has internal *memory* that dictates the output value making up the input to the next cell. In addition to this processed data from the previous time step, the cell receives input data from the current time step. This effectively makes all inputs in a given sequence dependent on each other. However, the vanilla RNNs have limited effectiveness

for input sequences above five to ten timesteps due to the gradient-based updates of weights, causing the backpropagated error either to vanish or explode [11]. *The Long Short Term Memory network* (LSTM) avoids these issues by implementing improved capabilities in the cell's internal state, allowing them to learn what information to retain or forget from given time steps. The implemented nowcasting framework by Shi et al. further improves this model architecture by implementing convolutional operations in the cells, transforming the model architecture to one well-suited for handling spatiotemporal image sequences.

The output of the convLSTM is a probability map of future rainfall intensities. The presented results demonstrate the models' abilities to produce more precise nowcasts, with fewer false alarms than the then state-of-the-art optical flow model, called *ROVER2*. However, the generated nowcasts suffer from blurring as significant uncertainties are connected to the produced radar maps [44].

MetNet: A Neural Weather Model for Precipitation Forecasting

The next influential improvement was presented by Sønderby et al. 2020 by the introduction of *MetNet*. This implementation was built on the Convolutional LSTM presented by Shi et al. but had several significant improvements. The extended architecture consists of 3 main parts:

- A *Spatial Downsampler*, where the model input is processed through a series of convolutional operations to reduce the spatial size. As the input to MetNet consists of both radar images, 16 spectral bands from the GOES-16 satellite, in addition to latitude, longitude and topographic data, this was necessary to maintain manageable computational requirements.
- The next block of the network architecture is a *Temporal Encoder*, represented by the convolutional LSTM. This part of the network is responsible for learning the temporal dynamics of the input.
- The last part of the network is a *Spatial Aggregator*, consisting of a series of axial self-attention blocks. This modified version of the regular self-attention enables the network to cover the full spatial context of the input while simultaneously avoiding computational restraints.

With this architecture, MetNet produced precipitation forecasts for up to eight hours of lead time, outperforming several numerical weather prediction methods (NWP) [49]. It is important to note that the mentioned performance was benchmarked against NWPs, and not the optical flow methods presented in Section 2.2 due to the extended lead time of MetNet of up to eight hours. Despite reaching state-of-the-art performance, the network still suffered from blurred predictions du the produced probabilistic precipitation map [39].

Skilful precipitation nowcasting using deep generative models of radar

The next significant improvement was presented by a model framework based on a generative adversarial network (GAN). As GANs are generative models like the DMs presented in Section 3.2, their goal is similar: They generate examples from an estimated probability distribution, estimated by studying training

examples. How GANs generate samples differs from DMs, however. At its core, GANs consist of 2 *adversarial* parts: A *generator* and a *discriminator*. These have their own distinct objectives. The generator's objective is to generate *fake* samples that resemble samples from the original distribution. These samples are then used as input for the discriminator, which tries to determine if a sample stems from the original distribution or is a fake sample produced by the generator. Both the generator and discriminator are trained simultaneously, enabling the discriminator to increase its accuracy in separating real samples from fake, forcing the generator to produce more realistic samples [13].

As published in *Nature*, the GAN developed by Ravuri et al. at DeepMind provided the first generative approach to nowcasting that surpassed the predictive performance of a majority of the other established baselines. The model used radar precipitation rates as input to produce nowcasts on lead times ranging from five to 90 minutes on a 1,536 $\times$ 1,280-kilometer area covering the United Kingdom. In contrast to the above-presented frameworks, this model can produce realistic precipitation scenarios, as it is trained to generate samples that resemble real-life radar images instead of producing a probabilistic precipitation map. To quantify the uncertainties of the nowcast, the model generates a distribution of nowcasts for each scenario. As the generator in the GAN samples from a random latent vector to generate a sample, each prediction will be slightly different [39, 46]

3.5 Summary

In this chapter, we have presented relevant theory regarding deep learning and neural networks used for image analysis. The first section presented the basic theory behind the perceptron and how they make up a complete neural network. We also presented theory on how these networks *lean*, by updating their parameters with loss functions and backpropagation. For the last part of this section, we focused on machine learning for image analysis, by introducing convolutional neural nets and the U-net. The second part of the chapter presented theory and concepts behind diffusion models. We presented how the network is able to turn gaussian noise in to synthetic samples resembling samples from the original distribution using the reverse diffusion process. This section also outlined two essential metrics, later used for validation purposes. Lastly, we presented related work in the form of three papers that had a significant impact on the development of machine learning in precipitation nowcasting.

CHAPTER 4

DiffMet: A diffusion model for precipitation nowcasting

To further drive the technological advancements in weather forecasting, in this master thesis we have developed a novel approach for precipitation nowcasting based on diffusion models. As discussed in Section 3.4, machine learning has shown great promise for producing high-quality precipitation forecasts. In addition to creating predictions in a more time-efficient manner than their numeric, physics-based counterpart, some models have also been shown to be superior in accuracy for several precipitation cases [39]. Some of this success is partly due to the well-proven machine learning algorithms utilized in these implementations, like the GANs or LSTMs outlined in section 3.4.

This chapter, however, represents a step into the unknown, as it is to the best of our knowledge the first application of diffusion models for precipitation nowcasting and weather forecasting in general. Moreover, diffusion models are young of age and lack the set of proven hyperparameters that the above-mentioned models have established. Combined with a model architecture consisting of several moving parts, this makes for a challenging task. The upsides, however, are that diffusion models have been shown to be superior to the current state-of-the-art GAN models, both in sample quality and training stability [8]. This is hence the main motivation for choosing these models.

This chapter outlines all relevant methods making up the DiffMet module and will present a detailed look into each component. Section 4.1 presents the dataset created for training the model. This is a compiled dataset built by writing scripts for accessing an application programming interface. This gathered raw data requires several processing operations before being fit as model input to the model. This process is presented in Section 4.2. We then present in Section 4.3, a detailed look at how the complete module, based on diffusion, was done. This includes the implementation of the mechanics behind the diffusion model itself, but equally important, the strategies applied for generating forecasts. This transforms the unconditional model into one conditioned on radar video from the past 20-minute observations. Two strategies were implemented to achieve this, presented in Section 4.4. Section 4.5 details the process of generating samples of radar data post-model-training, with the computational challenges related to this. Finally, Section 4.6 presents the implementation of the special set of metrics used for measuring nowcast quality.

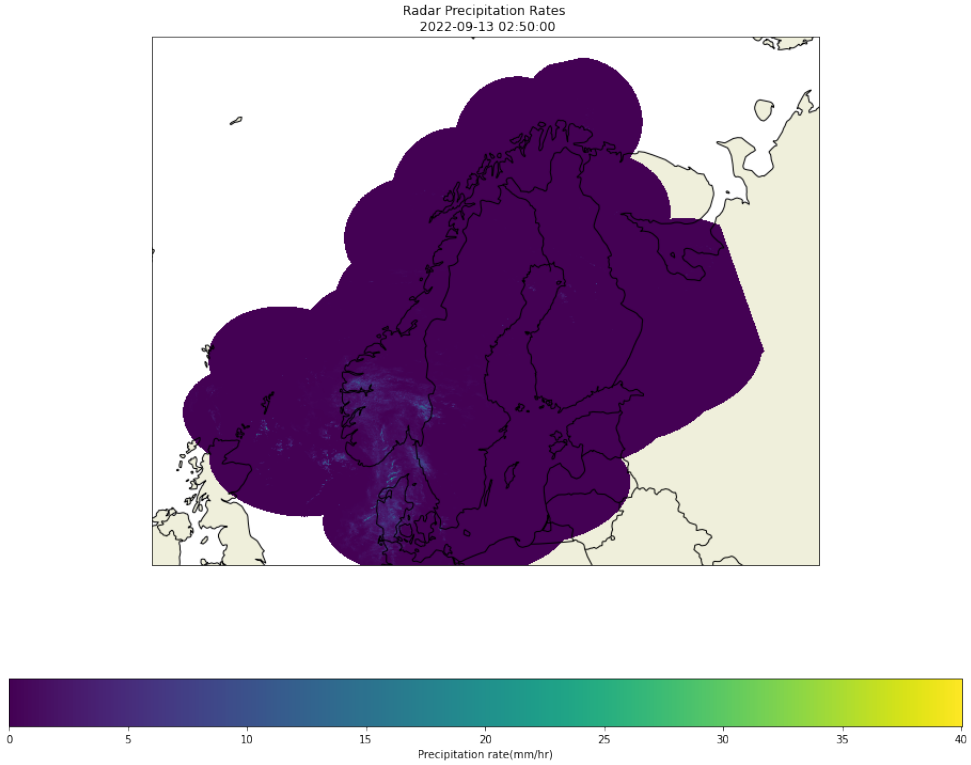

Figure 4.1: Radar precipitation rates from a random sample from the dataset. The values are projected on a map of Scandinavia, showing precipitation over Denmark and southern parts of Norway. This figure illustrates the total area covered by the ground radars.

4.1 Radar reflectivity archive

This section presents the main dataset used for model training, a custom dataset built from variables extracted from the API provided by The Norwegian Meteorological Institute (MET).

MET provides radar reflectivity data, covering Norway and the nordic countries [1]. This data is free of use to the public and can be accessed through the THREDDS data server with the OPeNDAP data access protocol [2]. Each daily 24-hour period of observations is stored in a separate file, containing 15 different variables, each containing information with a 5-minute temporal resolution and 1 square kilometer spatial resolution. The dataset covers an area of 1694 $\times$ 2134 kilometers, which amounts to a substantial-sized dataset.

The archive contains daily observations dating back to July 2020. However, the observational area changed from 2020 to 2021. Combining data from the two observational areas may provide several challenges, as one needs to be assured that the indices between samples represent the matching, spatial location. This

[1] https://thredds.met.no/thredds/catalog/remotesensing/reflectivity-nordic/catalog.html

[2] https://www.opendap.org/

is important as precipitation data represents highly spatially dependencies. It was therefore decided to only use data from 2021 and 2022. Only the months from April to October were used to ensure that the reflectivity data contained rain and not snow.

4.2 Data pre-processeing

4.2.1 Domain

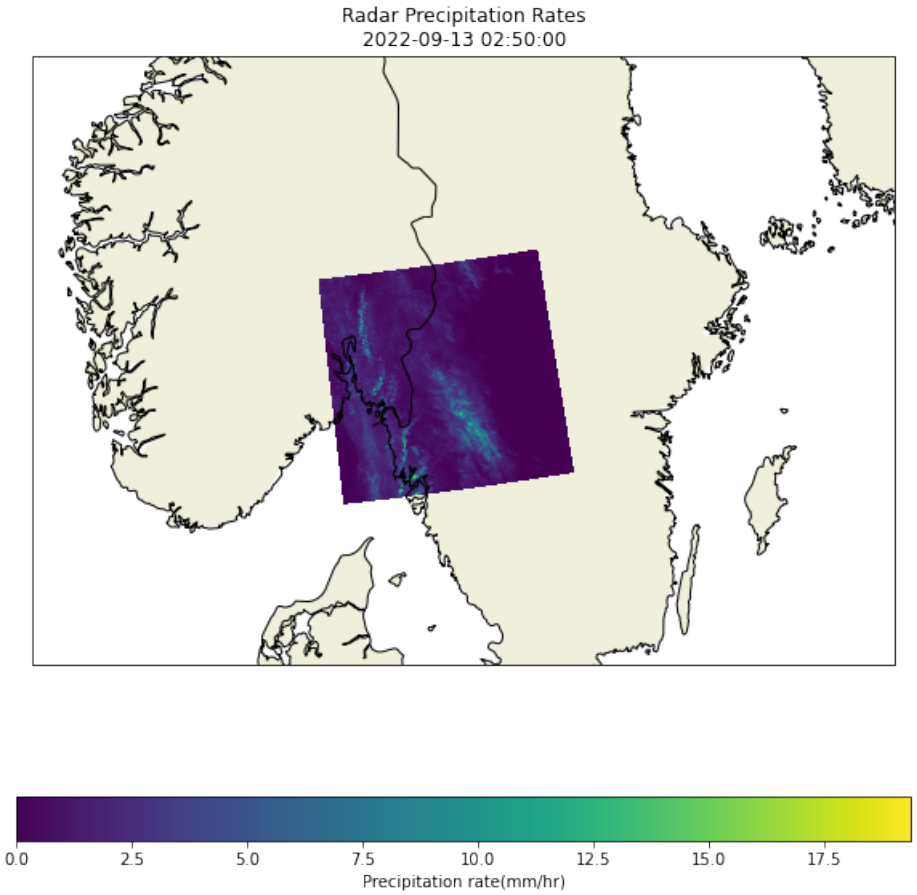

Figure 4.2: Visialization of the selected domain. Radar precipitation rates are projected over a map of Sweden and the eastern part of Norway.

As described in Chapter 2, precipitation patterns can be highly spatial dependent. These patterns are complex and already prove challenging to predict. An area of 256 by 256 kilometers was selected to optimize the model's predictive abilities. The area was selected based on radar coverage and to minimize topographical variation. The selected area spans from 10.15° to 14.81° longitude, and 58.16° to 60.53° latitude and is visualized in Figure 4.2.

4.2.2 Extracting data

To cover the above-specified period, 418 separate files had to be accessed through the THREDDS data server. These must be accessed by requesting the correct URL and specifying the variables and area of interest. Several files had missing data to varying extents. Dealing with this extraction manually is a huge time consumer, as each request will have different URLs to specify variables and mitigate the missing data. To deal with this, a series of scripts in the programming language of Python 3.9 was developed to interact with the Application Programming Interface(API). These scripts are a part of the *MetData* module and are presented in Appendix A. These scripts iterate through each day in the specified period and create requests to the data server with a URL, correctly specifying the corresponding date and variables. The functionality for handling missing data is implemented as recursive methods, iteratively adjusting the call to the server until complete data without missing values has been extracted.

4.2.3 Feature selection and engineering

The dataset contains several variables of interest; three of the 15 were selected. The main variable of interest is the *Radar Precipitation Rate*. This variable has data stored in a 3D array, representing precipitation rates on a 2D grid with 1-kilometer spatial resolution and 5-minute temporal resolution. The process of calculating precipitation from radar reflectivity is already computed beforehand by MET. The dataset also contains a variable with the same dimensionality as the Precipitation Rate but instead contains binary values describing if the precipitation is either convective or stratiform. Lastly, a variable containing date and time information was also extracted. The longitude and latitude variables were only used initially for selecting the correct geographical location and were discarded afterward to save disc space. To aid in evaluating the model performance, the Radar Precipitation Rate and the convective variable were used to engineer a new variable. The original variable provided by MET had grid cells with convective precipitation correctly specified. Still, all other grid cells were classified as stratiform, whether the cell contained precipitation or not. As this variable is planned to be used in model validation, its original form will greatly overrepresent occurrences of stratiform precipitation. As a result, an implemented method uses both the original classification variable combined with radar precipitation intensities to generate a new varable. This variable contains integers ranging from zero to two, representing convective, stratiform, or no precipitation, hence giving a more realistic representation of the actual distribution of precipitation types.

4.2.4 Data processing

A series of scripts in python was developed to process the data to a dataset suitable for machine learning. The main goal for this master project has been precipitation nowcasting, based on images of precipitation rates from previous time steps. This gave rise to the need to create video sequences of radar images. Most frameworks already developed for deep learning on images are developed for single images. As a result, a large portion of the scripts for processing

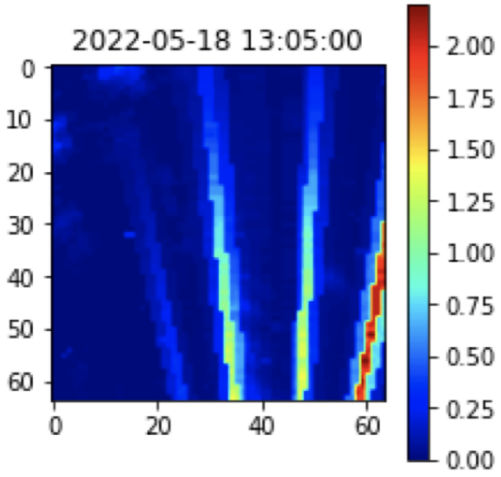

Figure 4.3: Example of a radar image with obvious radar clutter. The presented radar precipitation intensity is from a 64 × 64 square kilometer outcrop from the dataset, with intensities in the range of [0,2].

had to be developed ground up. Rather than focusing on the data in single frames, the scripts developed gather statistics of a sequence consisting of eight consecutive time steps. These statistics will be used in decision-making for whether a sequence is suited for model input or not. A challenge when gathering precipitation data is that most frames contain little to no precipitation. Simply selecting all frames will create several challenges. From a model perspective, much of the training time will be spent on all-zero frames not contributing to optimizing model performance. However, they will occupy the same amount of disc space as their precipitation-filled counterparts, leading to datasets of unnecessary size. To combat this issue, a set of threshold values are selected. To ensure non-empty sequences, a threshold for the minimum sum of total precipitation contained in the sequence is set.

As the datasets consist of remote sensing data, noise, and clutter are unavoidable. This is visible in the dataset as precipitation rates in the magnitude of 10^3. In addition, some extreme weather events see precipitation rates far exceeding what is typically observed. As all values will be further processed, normalized, and scaled within a fixed range, these outliers can have a major impact on the distribution of most of the most typical precipitation rates found in the data. Section 4.2.5 presents a detailed explanation of this further data processing. To avoid both cases, the script uses a threshold for the maximum precipitation rate contained in a single grid cell. However, Figure 4.3 presents a radar image with obvious radar clutter but with precipitation rates within the thresholded values. This illustrates that compiling a dataset with some instances of radar clutter is unavoidable. Finally, the sequence is controlled for NaN values. If the sequence values are within the set values, the sequence is saved to a four-dimensional array. This contains a 2D grid of precipitation rate values and types over the given area for eight consecutive time steps. A timestamp, given in seconds since 01.01.1970, is used as the filename instead of its original placement as a variable. This is done to reduce the dimensionality

of the training samples to keep the disc space used for each sequence to a minimum. This is important to ensure greater transfer speeds while loading data to the model, a process described in section 4.2.5. This timestamp can later be used to transform the value back to the original date and time.

Finally, all sequences are randomly divided between three folders, creating a dataset consisting of training, testing, and validation data. The total size of the dataset consists of over 10.000 sequences, making up 50 gigabytes of data. 80 % of the data is contained in the training split, while the remaining 20 % is divided equally between testing and validation sets.

4.2.5 Developing a PyTorch Dataset

PyTorch is a library optimized for deep learning on GPUs, requiring data to be loaded and transformed in specific ways. Thus, a custom PyTorch Dataset class was developed to handle these requirements [3]. The custom dataset developed is contained in the GitHub repository under `DiffMet/prec_dataset.py`.

These custom classes perform several important tasks. First, they allow for separation between model training and data handling, as each custom dataset must implement a series of pre-defined methods for reading and loading data. An initializer was the first method developed. This method acts as a way to define the purpose of the initialized dataset, as well as define the necessary transformations to be performed. This definition is important as the training, test, and validation datasets will have several differences in how they transform the data.

For the data used in training and testing, a randomly selected full eight-frame sequence consisting of the 256×256 km^2 area described in Section 4.2.1 is loaded from a file. The file is loaded as a NumPy array to perform the needed transformations [14]. The first transformation applied is a data augmentation. However, many of the more widely used augmentation techniques have to be limited in this project, as the data is highly spatially dependent. Making use of commonly applied augmentations like horizontal flips and skews will have the effect of disrupting this dependency, and is therefore avoided. However, a 64×64 random crop of the image is performed. This is done by sampling x, and y indices within a valid range shared for the whole sequence. This will also drastically reduce the computational expenses while training the network, as the input size is reduced from 256×256 to 64×64.

For the validation data, a center crop replaces this random crop. This ensures that validation metrics are calculated on the same patch of spatial data, enabling comparison between model initializations. This is important because the 256×256 area can contain different weather patterns. These different model initializations will be discussed in 4.3.

After successfully cropping the observational area, all values are normalized and scaled. This is done based on both model prerequisites and to optimize model training. First, all values x_i are scaled by $x_i = \sqrt[3]{x_i}$. This is done to scale the precipitation intensities to minimize the distance from normal intensities to extreme values and creates a range better optimized for model convergence. Then values are normalized, first to be contained in the range of [0,1], and then to [-1,1]. This ensures the neural network reverse process operates on

[3] https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

consistently scaled inputs starting from the standard normal prior $p(\mathbf{X}_T)$ [18]. Finally, all arrays are transformed to *Tensors*[4] the required data type needed for PyTorch to enable training on GPUs.

4.3 The development of DiffMet

This section presents the algorithms and scripts for the developed python module named `DiffMet`. All code developed can be found in the provided GitHub repository under the folder with the same name.

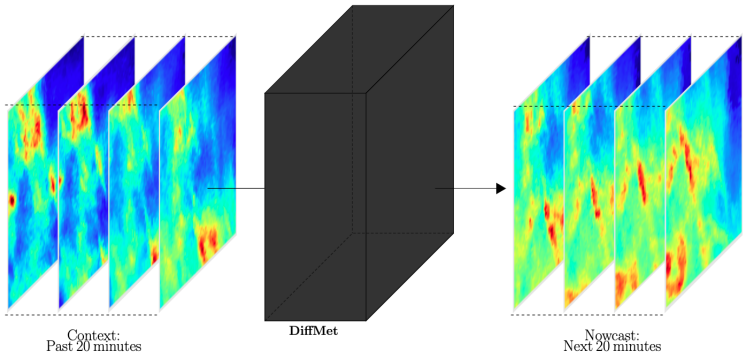

Figure 4.4: Conceptual overview of the intended purpose of nowcasting with the developed Python framework, named *DiffMet*. The framework takes as input a radar video from the past 20 minutes. The output is a video generated by the diffusion model, predicting precipitation for the next 20 minutes. Illustrated is a precipitation event from the selected domain, starting on the 11th of May, 2021, at 10:45 GMT + 1.

Machine learning frameworks like PyTorch often contain pre-built models for popular architectures. However, diffusion has only gained popularity recently, and advanced models with modification possibilities are missing from most model libraries. This hindered the possibility of producing results with out-of-the-box models. For implementation, the focus was therefor shifted towards published papers, as many provides GitHub repositories with model implementations, including the seminal paper from Ho et al. [18]. These repositories often have a code base consisting of thousands of lines of code, specifically tailored to their particular dataset and experiments. As a result, simply copying these implementations and getting the code to run proves challenging. Furthermore, extending the functionality to act as models suited for solving the challenges with nowcasting proved even more difficult.

As a result, the main body of work in this master thesis has been implementing and extending the functionality of a diffusion model from the ground up. This is done to fully control every model component, enabling further implementations and extensions. As explained in chapter 3, diffusion models consist of several components, each with their specific task. The developed diffusion model is therefore designed as a python framework, with each

[4]https://pytorch.org/tutorials/beginner/introyt/tensors_deeper_tutorial.html

component created as a separate python script. The components and finished structure are inspired by several implementations, where components have been either modified or completely rewritten[5,6,7,8]. The following subsections present the implementation of these components and how they each relate to the theory presented in chapter 3.

4.3.1 Beta schedules

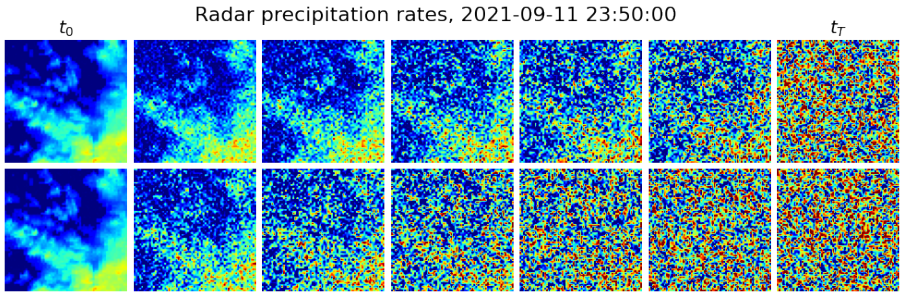

Figure 4.5: Visual comparison of applying noise to a random sample from the validation data set. The noise has been applied using the implemented method described in 4.3.2. As $T = 1000$, only a selection of values for t has been selected, ranging from $t = 0$ to $t = T$. The two rows compare the corresponding values for t. Top row: Cosine beta schedule. Bottom row: Linear beta schedule. This comparison illustrates how the linear schedule adds noise in a more rapid fashion than the cosine schedule. Inspired by the figure in Nichol and Dhariwal 2021 [33]

For gradually adding noise to the data, Ho et al. proposed a constant linear β_t schedule where $\beta_1 = 10^{-4}$ and $\beta_T = 0.02$, for $T = 1000$. However, this was later considered sub-optimal for low-resolution images 64×64 and smaller. One of the problems discovered was that the linear schedule was too noisy in the upper range of T, having minimal contributions to sample quality. As a result, Nichol and Dhariwal 2021 proposed an improved *cosine* noise schedule. This schedule provides a linear behavior in the middle range but contributes to very little change near the extremes of $t = 0$, and $t = T$ [33]. This schedule is formulated in Equation 4.1. s is a small offset to prevent too small values of β_t for small t, which was found to improve sample quality.

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \qquad f(t) = \cos\left(\frac{t/T + s}{1 + s} \cdot \frac{\pi}{2}\right)^2 \tag{4.1}$$

Figure 4.5 presents a visual example of how a radar image from the dataset is reduced to noise through the two different schedules

[5]https://github.com/hojonathanho/diffusion
[6]https://github.com/lucidrains/denoising-diffusion-pytorch
[7]https://huggingface.co/blog/annotated-diffusion
[8]https://github.com/openai/improved-diffusion

4.3.2 Forward diffusion process

The main component making up the core characteristic of the model is the *forward diffusion process*. In this script, a method for computing the noisy image at time step t, $q(\mathbf{x}_t|\mathbf{x}_0)$, has been implemented. First, the method extracts both $\sqrt{\bar{\alpha}_t}$ and $\sqrt{1-\bar{\alpha}_t}$ for the given random value of t. This is done by receiving the full lists of all values for $\sqrt{\bar{\alpha}}$ and $\sqrt{1-\bar{\alpha}}$. These terms are all pre-computed, defined by the mathematical operations presented in Ho et al.[18]. These outputs are then used to calculate $\sqrt{\bar{\alpha}}x_0 + \sqrt{1-\bar{\alpha}}, \epsilon, t$, which is the added noise to the input image $\mathbf{x}_0$. The method returns both the noised image and the random sampled noise, ϵ, as well as the noised input image, $q(\mathbf{x}_t|\mathbf{x}_0)$.

4.3.3 Loss

Choosing an appropriate loss function for the neural network is crucial, as it determines how the network weights are optimized. The goal of this function is to give a numeric value describing how well the network can predict noise in the image. The loss function is implemented with a function call to the model, returning the predicted noise. The loss function used is the Mean squared Error, presented in Equation 4.2. This returns a pixel-wise comparison on the true noise, y_{true} compared to predicted noise, $y_{predicted}$. Furthermore, as the model makes predictions on a lead time, the size of the target patch must differ from the input size. This is caused by the need for spatial context around the target patch, accounting for the displacement rate of weather patterns. For the implemented loss function, the dimensions of the final target patch is calculated as a function of lead time. Sønderby et al. found an indicative average precipitation displacement of 1 kilometer per minute. Thus, the number of minutes from the last observation to the relevant lead time, will be cropped from each side of the target patch, leaving enough spatial context around the surrounding area in the conditional input [49]. The indices specifying the cropped target patch are also used to crop the corresponding area from the true noise, making up the two inputs for comparison.

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_{\text{true}} - y_{\text{predicted}})^2 \tag{4.2}$$

4.3.4 Positional Time Embedding

As presented in section 3.2.4, the neural network used for predicting noise is the same for all t noise levels, thus requiring input arguments describing the noise level to expect. This information is passed to the model through a sinusoidal position embedding, originally developed to be used by transformers architecture in natural language processing, presented by Vaswani et al. 2017 [54]. In contrast to being utilized on sequences of words, the embedding in this implementation will take a t-value and extract a real-valued vector. This vector corresponds to a row in the complete encoding matrix, resulting in a unique sinusoid for each time step. As the matrix is computed using a combination of cosine and sinus trigonometry functions, all valuers are transformed to the range $[-1, 1]$, matching the transformed values from the radar precipitation

data set. This transformation also has the benefit of enabling the model to compare similarities between sinusoids [1].

The implemented positional encoding is based on the methods originally developed by Ho et al. [18]. The output from the positional embedding method, x_{emb}, is then processed through a linear transformation, $y = x_{emb}A^T + b$, where b is a bias term. This processing ensures a matching size between the time embedding and the images processed through the U-net, allowing the embedding to be added to this data.

4.4 Condtioning the diffusion model

All model architecture explored in the various repositories connected to the papers chosen relevant for this thesis mainly focused on either an *unconditional* or *class-conditional* implementation [18, 33]. The unconditional models are defined by a training scheme where a selection of unlabeled images is used for guiding the model in the noise prediction. When generating samples with this model after convergence, the model uses sampled Gaussian noise as the only input to produce the final output $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, where $q(\mathbf{x}_0)$ represents the real data distribution used in training. These models are well-suited for producing high-quality image synthesis results. However, the model output, $p(\mathbf{x}_0)$, is a randomly selected approximation from the original data distribution, q. Ho et al. 2020 [18] present their results using the unconditional CIFAR10 dataset, consisting of 60.000 images contacting either of 10 different classes [26].

As a way to steer the sampling process toward a desired output, Dhariwal and Nichol 2021 present a conditional diffusion model using *classifier guidance*. This approach utilizes the training of a classifier $p_\theta(y|x_t, t)$, where y is an arbitrary class label. The classifier is trained on noisy images x_t and uses the model's gradients to steer the sampling process towards the class-conditional label y [8].

However, as the generated dataset presented in section 4.2.5 consists of sequences of highly varying radar reflectivities, the number of specific class labels needed to explain each sequence would be nearly infinite. These labels are, therefore, not available, forming the need to explore alternative approaches to condition the model on these sequences.

A series of novel techniques presented by Voleti and Jolicoeur-Martineau in a 2022 paper on video synthesis inspired to implement functionality usually found in video prediction problems [55]. The framework proposed, named *Masked Conditional Video Diffusion* (MCVD), outlines a series of methods for generating next-frame predictions based on sequences of past frames. These next sections present the two methods implemented in the DiffMet framework to generate samples conditioned on images of radar precipitation from the four consecutive previous time steps. In addition, parts of the following implementation are also based on the concepts proposed by Ravuri et al. 2021, employing a Generative Adversarial Network for precipitation nowcasting, as presented in Section 3.4. However, The GitHub repository containing the code for the GAN nowcasting model is only partly open-source. It mainly consists of notebooks presenting visualizations of samples produced from an already trained model, limiting its usefulness for this thesis. Moreover, the code corresponding to the MCVD framework *is* open-source, however massive in size and lacking comments and

documentation. All implementations presented here are, therefore, only inspired by the concepts outlined in these papers but implemented from the ground up.

4.4.1 Concatination

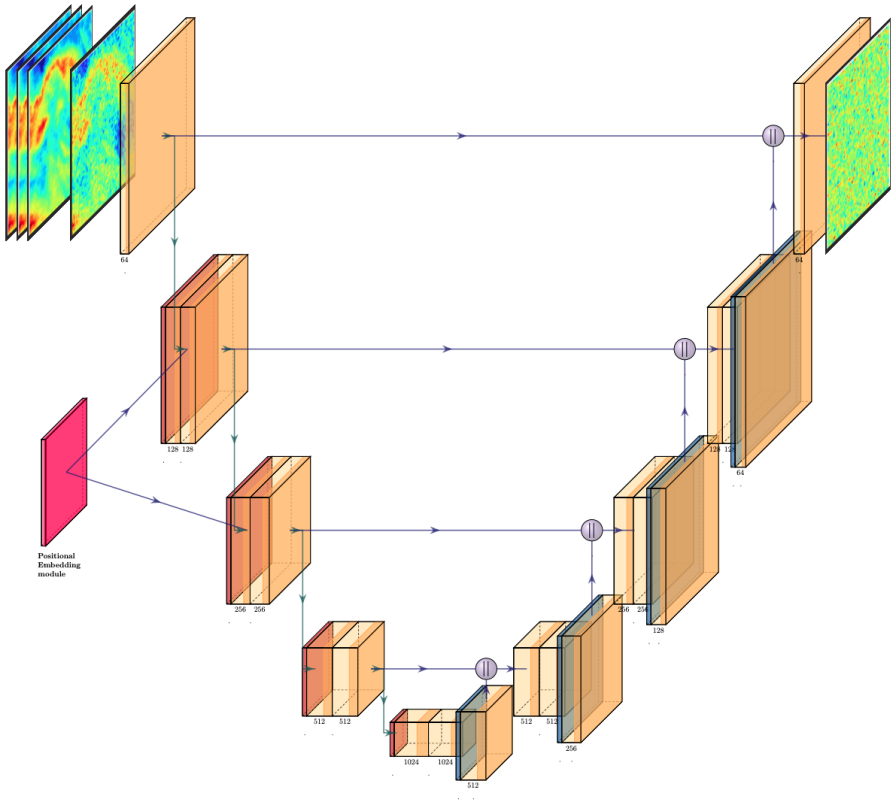

Figure 4.6: Schematic drawing of the convolutional neural network, utilizing concatenated conditional input. The network input is the noisy frame, concatenated with the conditional input from the previous time steps. For each block in the U-Net, a layer containing positional information describing the current time step is added to the data flow. The yellow blocks represent layers of convolution. In the decoder part of the network, the dark orange layer illustrates how the spatial dimensions are reduced while the number of channels is increased. The blue layer in the decoder represents the transposed convolution, increasing the size while reducing channel dimensions. As presented in Section 3.1.4, each block in the decoder part of the network has the input from the corresponding block in the encoder directly added. For illustration purposes, some model details are not visualized.

The first implementation utilizes a technique that involves several modifications to the code on the data processing side of the model. In this implementation, a slicing operation is performed on each loaded sequence from the implemented data loader presented in 4.2.5, creating a new tensor consisting of the first four frames present in the original eight-frame sequence. This

tensor makes up the sequence used for conditioning the model output. This is implemented in the training loop as a constant intake of conditional information for each noise level, t. Algorithm 3 outlines the updated training objective. $x \sim q(x)$ denotes a sequence randomly drawn from the original dataset. The model is implemented with the flexibility of choosing the number of frames to use as conditional input. x_{cond} is a tensor of n past images whose length depends on this selection. Functionality for selecting lead time has also been added, and x_{lead} will be a single frame corresponding to the index of the lead time of interest from the original sequence. This single frame is then processed through the implemented method for*forward diffusion process* presented in section 4.3.2. x_{input} denotes the concatenation of a noised image x_{lead} with the conditional sequence x_{cond}.

Algorithm 3 Training, concatenated model

repeat
- $x \sim q(x)$
- $x_{\text{cond}} = [x_0, x_1, x_2, ..., x_n]$
- $x_{lead} = x_{\text{lead time}}$
- $t \sim \text{Uniform}(1, ..., T)$
- $\epsilon \sim \mathcal{N}(0, 1)$
- $x_{\text{noised}} = \sqrt{\bar{\alpha}} x_{lead} + \sqrt{1 - \bar{\alpha}_t}\epsilon$
- $x_{\text{input}} = x_{\text{noised}} + x_{\text{cond}}$
- Take gradient descent step on $\nabla_\theta ||\epsilon - \epsilon_\theta(x_{\text{input}}|t)||^2$

until converged

To facilitate this model extension, the number of input channels in the neural network has to be adjusted to match the number of conditional frames added with the lead time frame. As Algorithm 3 outlines, the conditional information is being sent to the model for each timestep, and should, in theory, provide the model with proper guidance. Figure 4.6 presents a simplified schematic drawing of the main concepts behind this network architecture. Each convolutional block is based on the components presented in the initially proposed U-net architecture, presented in Section 3.1.4. For illustrational purposes, this overview only shows the main components of the network, highlighting how the conditional input is concatenated as input at the start of the network. Presented is also how each convolutional block receives the sinusoidal positional embedding, though the figure only visualizes this for one block. The layers of activation and normalization applied within each convolutional block are not illustrated.

4.4.2 Image Embedding

In an effort to further improve model performance, a second method was developed. This conditioning method differs widely from the concatenation explained in the previous section and involves implementing a second neural network. This second network is a *Residual Neural Network* (ResNet). The task of this network is to transform the conditional sequence into a different dimensional representation, called an image embedding. This technique is a widely used technique within the fields of computer vision and machine learning

and is motivated by the conceptual ideas outlined by Ravuri et al. 2020 and Voleti et al. 2022[39, 55].

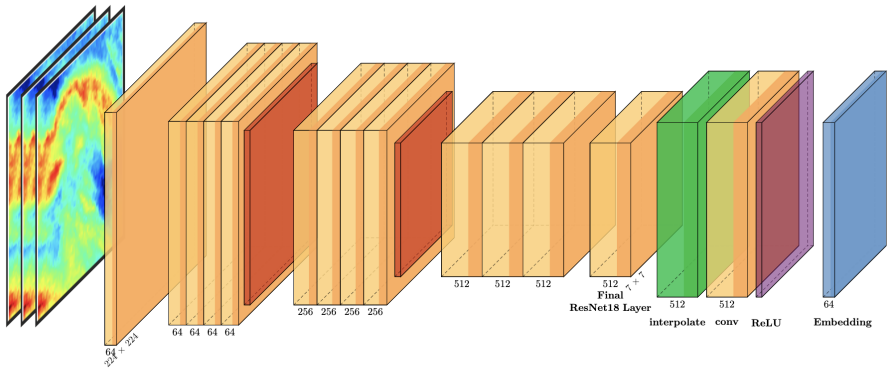

Figure 4.7: Schematic drawing of the implemented image embedding module used for conditioning the diffusion model. Radar precipitation rates from the previous three-time steps prior to lead time are being processed through the convolutional part of a ResNet18 model architecture. The produced feature maps are then processed through several operations consisting of interpolation, more convolutions, and an activation layer to generate the final image embeddings.

ResNet

The number of weight layers in convolutional neural networks closely correlates to its performance [45]. However, as the depth of the network increases, so does the difficulties of training the network. He et al. 2016 showed that adding convolutional layers to a deep network could result in saturated accuracy that eventually degrades, indicating that parts of the network represent different difficulties in terms of optimizing. To allow for deep neural nets whilst avoiding this issue, the introduced ResNet utilizes *identity connections* between layers. These connections let underlying layers learn residual functions between layers, as they showed that learning the difference between input and desired output increases overall accuracy[16]. As a result, these networks have become a staple in models used for image analysis and are often found pre-built in many machine learning frameworks. The network of choice for this implementation was the *ResNet18* architecture, consisting of 18 layers. Each layer consists of a convolutional operation followed by batch normalization. The output is then activated through a ReLU function. As PyTorch offers this architecture as a pre-built model, this makes up the first half of the embedding module [9].

However, pre-built implementations like the ResNet18 are constructed for classification tasks, where the model outputs the probabilities for an input sample belonging to a fixed number of classes. As visualized in Figure 4.7, the ResNet18 model architecture consists of the abovementioned 18 layers. After the final convolutional layer in the original implementation, the output is average pooled before being linearly transformed through a fully connected layer,

[9] https://pytorch.org/vision/master/models/generated/torchvision.models.resnet18.html

and activated with a softmax function. This fully connected layer enables the network to produce class probabilities, with each output neuron corresponding to a class probability.

As this probability output differs from the training objective in focus for this study, further implementations and adjustments had to be made to produce embedding on a better-suited dimensionality. To achieve this, the implemented module extracts the model's output from the last convolutional layer prior to the average pooling layer. At this point in the ResNet18 architecture, the output is a $512 \times 7 \times 7$ feature map tensor, transformed from the conditional input. The spatial dimensions of the feature maps in the U-Net range between 64 and eight, depending on the position in the encoder/decoder. The embedding tensor, therefore, needs to be processed through an interpolation process, where the spatial dimensions are made to match the feature maps produced by the different depths of convolutional blocks in the U-net. To obtain a matching number of channels, the embedding is processed through convolutional layers before being activated with a rectified linear unit.

Transfer learning

As presented in Section 3.1.3, the process of training a convolutional neural network is the process of updating the values of the randomly initialized weights so that the resulting convolution eventually outputs meaningful feature maps. Outlined in the same section are also the concepts on how a convolutional net will learn to identify the different levels of detail based on layer depth. However, methods exist for utilizing kernels already trained for this task of detail recognition. This method, defined as *Transfer learning*, enables the use of employing models pre-trained on domains that differ from the intended use case. The challenges proposed to a network with weights initialized in this manner propose a significantly more managable challenge than when initialized with random weight [57].

Most pre-trained models are trained on ImageNet, a large-scale annotated dataset comprising 14 million images divided into 21,841 subcategories. These subcategories, or classes, are manually labeled, and consist of typical objects and concepts like animals, plants, and vehicles [6]. However, Stewart et al. 2022 presented the *TorchGeo* framework. This framework provides weights from models trained on remote sensing data, developed specifically for models aimed at geospatial data [52]. For the implemented diffusion module in this project, a set of weights optimized on data from Sentinel-2 satellites was extracted and applied to the ResNet18 model architecture. Although the radar precipitation dataset provided by MET utilizes ground radars, an assumption is made that the geospatial data provided by the orbiting satellites is still better suited than the weights trained on the standard ImageNet.

Figure 4.8 visualizes a schematic drawing of how this embedding is infused with the feature maps at every block in the U-net.

The improved Unet

The U-net used in the image embedding model uses the same main structure as the one used for the concatenated model, but has several updated components. First, the type of normalization used in the convolutional blocks is changed. As

Algorithm 4 Training, embedding model

repeat
$\mathbf{x} \sim q(x)$
$x_{\text{cond}=[x_0,x_1,x_2,....,x_n]}$
$x_{lead} = x_{\text{lead time}}$
$t \sim (1, ..., T)$
$\epsilon \sim \mathcal{N}(0, 1)$
$\mathbf{x}_{noised} = \sqrt{\bar{\alpha}}\mathbf{x}_{lead} + \sqrt{1 - \bar{\alpha}_t}\epsilon$
$\mathbf{x}_{input} = \mathbf{x}_{noised}$
Take gradient descent step on $\nabla_\theta ||\epsilon - \epsilon_\theta(\mathbf{x}_{input}|t, \mathbf{x}_{cond})||^2$
until converged

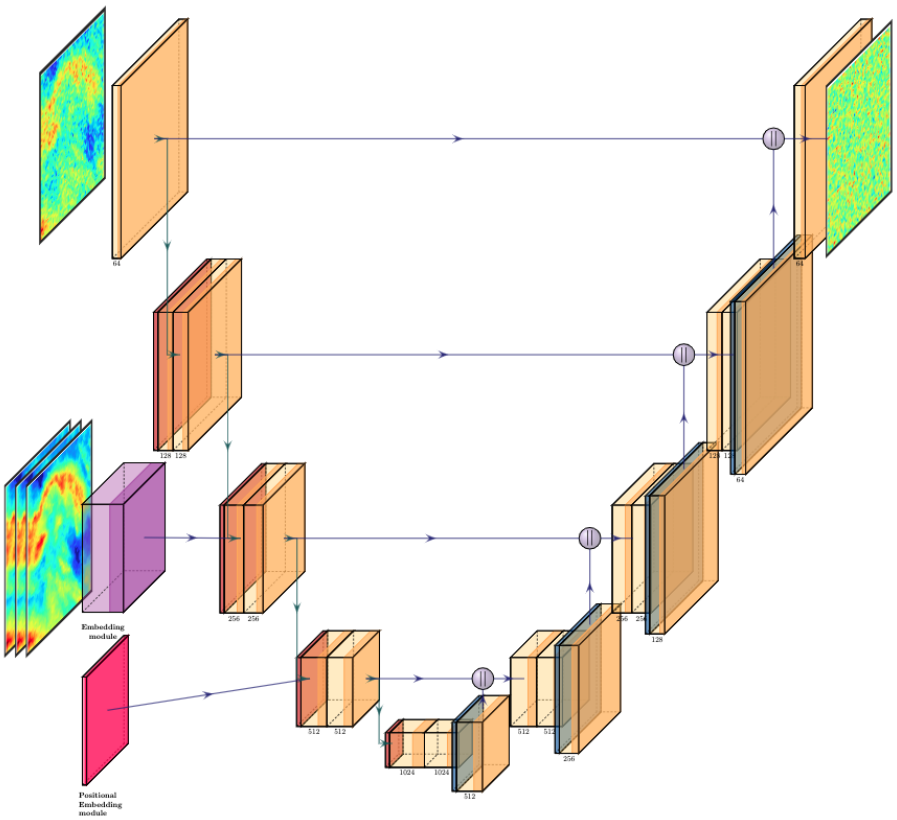

Figure 4.8: Schematic drawing of the final model architecture. For illustration purposes, several simplifications have been made.

each sample consists of a sequence of several input images, this puts constraints on the batch size used in training and validation due to the memory it requires by the GPU. As the normalization performed in *batch normalization*, as the name suggests, is performed on the statistics within the batch, this might lead to rapidly increasing errors due to small batch size. *Group Normalization* is introduced to combat this, instead calculating the mean and variance in groups divided along channel dimension [59].

The ReLU activation functions were replaced with Gaussian Error Linear Units (GELUs). These units combine the behavior of the functionality of ReLU with *drop-outs*. The functionality of drop-out is to randomly multiply some activations by zero as a way to regularize the model. This activation function has been proved superior to ReLU for computer vision uce-cases [17].

Figure 4.8 presents a conceptual visualization of the architecture behind the complete image embedding network. In this implementation, the noisy current image is the only initial model input, while each block of the U-net receives conditional information from the embedding model in addition to the positional embedding. The two embeddings are added together with the feature maps from the previous convolutional block. For illustration purposes, the drawing only illustrates the output from the embedding module and positional module being sent as input to one block each. However, every convolutional block in both the encoder and decoder part of the network receives these embeddings as input.

4.5 Sampling

An often overlooked part in the discussion on diffusion models is the topic of *sampling*, or more precisely, the computational cost of sampling. As presented in Algorithm 1, the process of training a diffusion model until convergence is the iterative process of drawing samples from the original distribution, corrupting it with a noise level according to t before the parametrized model makes a prediction on the noise contained in the image. For the process of sampling, however, there is a key difference that plays a major role in the computational cost of producing a sample which can easily be overlooked and underestimated. As Algorithm 2 outlines, generating a sample is the iterative process of gradually denoising X_T, which initially start as randomly sampled Gaussian noise. This involves making model predictions iteratively for $t = T$ to $t = 0$. Moreover, the size of T has shown to have major impacts on model performance, with higher values yielding better model performance [47]. Ho et al. 2020. used a T-value of 1000, while Nichol and Dhariwal 2021 trained all their models on 4000 diffusion steps [18]. While this has no direct impact on the computational burden of model training, producing a single sample after completed training requires the model to be called 4000 times. Thus, generating a single sample takes several minutes, even when employing high-performing modern GPUs.

This, of course, has major implications for the practical use of diffusion models, resulting in the development of *sampling noise schedules* [33]. These schedules work on a subsequence of the original noise schedule used in training, resulting in the iterative sampling process being completed in a fraction of the original steps. However, the choice of loss functions directly dictates how much sample quality suffers from this reduction. The loss function implemented

in the models developed for this project utilizes the simplified loss function, L_{simple} presented in Equation 3.15. This is a reweighted form of L_{VLB}, the loss on variational lower-bound [18]. As Nichol and Dhariwal 2021 presented a hybrid learning objective, combining both L_{simple} and L_{VLB}, they implemented a sampling noise schedule using just 100 of the original 4000 steps while still achieving near-optimal sample quality. Models trained exclusively on L_{simple}, however, has its sample quality greatly reduced. With this in mind, further efforts to implement a sampling noise schedule for DiffMet were not made.

4.6 Metrics

As presented in Section 3.3, probabilistic forecast models often utilize specific verification metrics. Many of these are specifically developed for forecasting models and are not included in most machine learning libraries. As a result of this, the two metrics that were used had to be implemented from the ground up. The choice of implemented functions was based on the selection of metrics found in related studies [10, 39, 49]. These papers include conceptual and mathematical formulations describing these metrics and were used as inspiration for the implementations made in this project.

For computational efficiency, a large part of the metrics has also been implemented in PyTorch, allowing the metrics to be calculated in the same high-performance manner as mentioned in Section 4.7.1 [34]. All implemented methods are located in `DiffMet/metrics.py` and `DiffMet/calc_metrics.py` and consist of the two primary methods `csi()` and `crps()`, as presented in Section 3.3. In addition, methods for creating pooled neighborhood metrics were developed. These are the same pooling operations described in section 3.1.3, consisting of max-, and average pooling. The motivation for this downsampling is to measure the model's performance beyond pixel-wise performance. The algorithm performs max-pooling with 2 different output sizes. This can be motivated by estimating the model's ability to predict extreme precipitation events, where exact pixel location might be challenging. The average pooling can be motivated by flood forecasts, where precipitation is accumulated over large areas [15].

4.7 Additional details

4.7.1 Software

A wide variety of software packages and libraries are required when developing end-to-end machine learning models. To manage these, Anaconda was used. This is an open-source distribution that enables you to create virtual environments specific to the project's needs. At the core of this environment is Python 3.9, which has been the main programming language for data processing and model building. This is a language that is widely used in scientific programming. Pytorch was used for building the architecture of the diffusion models[34]. This machine-learning framework is written in Python, specifically tailored for computer vision and developing deep-learning models. All visualizations of results have been produced with Matplotlib and Cartopy [21, 32]. Both frameworks are written in python and widely used for visualizing

scientific data. Figures for illustrating model architecture are produced using TikZ and PlotNeuralNet, both languages for drawing vector graphics [22, 53]. The figures of the two implemented U-Nets are inspired by examples provided through the PlotNeuralNet GitHub [35].

4.7.2 Computing resources

All models have been developed and tested in the above-mentioned local Python environment. A cloud-based computing backend was used for full-scale model training. This was provided through Colab, a cloud-based platform provided by Google Research. This enables model training on high-end Graphical Processing Units(GPUs). All models have been trained on clusters of either Nvidia A100 or NVIDIA V100. These high-performance GPUs provide between 16 to 80 gigabytes of high-bandwidth memory per GPU and are optimized for model training. However, Google Research does not provide the exact number of GPUs utilized in the cluster or the specific type of GPU.

4.8 Summary

In this chapter, we presented the developed framework for precipitation nowcasting, named DiffMet. First, Section 4.1 and Section 4.2 introduced the compiled dataset and outlined how this data was extracted from an API before being processed to be fit as input for our DM. We then presented in Section 4.3 and Section 4.4 how the theory behind DMs was implemented into a python framework and how this implementation had been further extended to include two different strategies for conditioning the model. Section 4.5 presented some computational challenges related to generating image samples post-training. In Section 4.6, we outlined the implementations of the metrics specific to measuring nowcasting quality before Section 4.7 presented the software used in addition to computational resources.

CHAPTER 5

Case study on nowcasting with DiffMet

As discussed in Section 3.4, deep learning and neural networks have shown promising results for use cases in weather forecasting. The methods have several advantages over traditional numerical methods. The latter relies on significant computing power to solve complex differential equations. They need several iterations with different initial conditions to create a probabilistic output and is required for every forecast produced. Models based on neural networks, on the other hand, are, once trained, able to generate forecasts for a fraction of the computational cost compared to their numeric counterparts.

In this chapter, we will evaluate our suggested DiffMet method in different use cases using the implemented metrics quantifying the probabilistic quality of the DiffMet forecasts. In Section 5.1, we consider the initial problem of tracking moving handwritten digits. We don't use any validation metrics but showcase how the model output compares to the wanted output by using data that are visually easy to compare. Unconditional output acts as a first test of the model's ability before the complexity grows by adding increasingly complicated conditional input.

Further in Section 5.2, we present the primary use case for DiffMet. These results are based on the prediction of short-term future precipitation by training DiffMet on real-life historic radar precipitation data. The probabilistic forecasting performance will be evaluated using the implemented metrics presented in Section 3.3. These metrics will be presented together with visualizations of model predictions to demonstrate further both the predictive and probabilistic abilities of DiffMet.

5.1 Inital Experiments – Tracking Moving Handwritten Digits

The main body of work in this thesis has been developing a generative model for generating synthetic radar images. In contrast to models used for classification or regression, these generative models often utilize a different set of validation metrics, as their training objectives differ. However, developing proper metrics for validating the model's ability to generate this output is challenging and time-consuming. Moreover, the generated radar images can be difficult to make sense of visually, as samples from the original radar dataset can easily be mistaken for noise. As described in Section 4.1, the final dataset also required

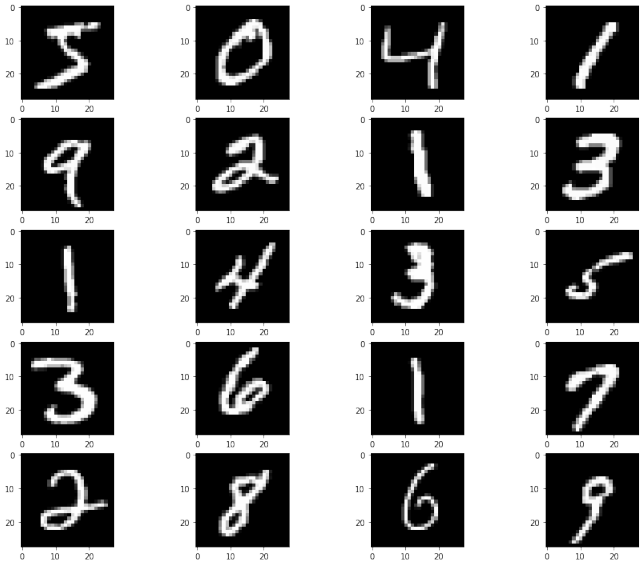

Figure 5.1: 20 randomly drawn samples from the MNIST dataset. Every sample is a handwritten digit ranging between zero and nine.

developing a significant amount of scripts to extract and process the data to its final form.

In the process of developing and implementing the diffusion model, the need for visually validating model performance on a simpler dataset was made evident. This made it possible to make progress in the development stages of the model before the final metrics were implemented. This also enabled lightweight training iterations where only a few generated samples were required to get a visual confirmation that the model output was heading in the right direction. Feeding the model a less complex dataset for testing initial performance is a common approach in machine learning and computer vision, and the selected dataset used for the initial results in this project is widely used in these fields. These next sections will present both the standard MNIST dataset as well as a modified variant of increased complexity. We will then present the model output, acting as a visual validation for model performance.

5.1.1 MNIST

The Modified National Institute of Standards and Technology database (MNIST) consists of a total of 70.000 handwritten digits between zero and nine with corresponding labels [29]. These are divided between a training set and a test set where the training set contains 60.000 images. All digits have been normalized in size and set to a fixed image size of 28×28, each containing grey levels in the range of 0 to 255 stored in one channel. As model input plays a large factor in the computational cost of training a network, this small image size makes them a perfect candidate for initial model testing.

.

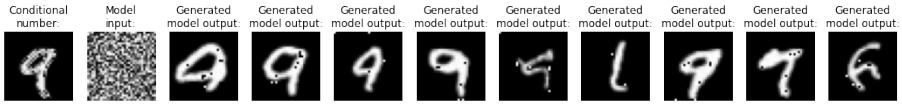

Figure 5.3: Conditioned output results from the model trained on the MNIST dataset. From left: conditional input and Gaussian noise as model input. The remainder of the row is generated model output. This sample visualizes nine generated samples, with the digit nine as conditional input. The digit that makes up the conditional input is always the same handwritten digit, both during training and sampling.

Unconditional model

In the first initial model test, the MNIST dataset was initialized through Torchvision, a library part of the PyTorch framework presented in Section 4.7.1. To ensure values are optimized for training diffusion models, all data underwent the same processing and transformation as described in Section 4.2.5. As this first experiment is *unconditional*, model training is simply a process of passing a batch of randomly drawn numbers from the original MNIST dataset, adding noise through the forward diffusion process, and having the model predict this noise. Figure 5.2 presents four samples generated from the unconditional model after training for 50 epochs. As shown, the model is clearly able to generate distinct digits resembling the handwritten digits from the original training data. Several of the samples shown have some noise still present in addition to the digit, indicating that the model is still incapable of generating perfect samples at the given time. This might be due to the restricted number of epochs, set to limit training time.

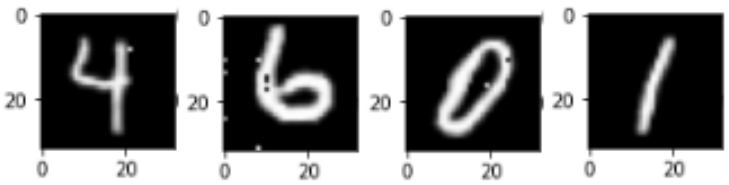

Figure 5.2: Unconditional output results from the model trained on the MNIST dataset.

Digit-condtioned model

The first significant test is to see if the model can generate samples based on conditional input. This also moves the model from a model that generates random output based on the distribution of the training data to a conditioned model that utilizes the implemented method based on concatenation presented in Section 4.4.1.

This extends the use of the MNIST dataset used in the first inital test and utilizes the corresponding labels that each digit provides. First, *one* handwritten digit is selected to represent all digits of the corresponding value for every number in the range of zero to nine. These ten selected digits will later act as the conditional input during training and sampling and are stored in a dictionary. This way, we limit the number of possible variations in shape and form for each conditional digit. This is done in an effort to maximize the model's ability to generate conditional output.

During the training of the network, labels for every sample in the batch are gathered from the data loader. These labels are then used to extract the corresponding handwritten digit from the above-mentioned dictionary of conditional input. This enables the model to concatenate the correct corresponding conditional input to the sample digit. The simplified training objective outlined in Equation 3.15 remains the same, as the model will compare the predicted noise levels in its generated output to the noise levels in the corrupted image of the digit drawn from the training data set. Hence, the conditional input only acts as a way to guide the network to produce samples from a digit-separated subset of the original distribution of all digits. Figure 5.3 demonstrates this with a wide variety of generated handwritten digits, sampled after training the model for 50 epochs. Most of the conditional output consists of variations of nine, as well as a few badly generated samples.

Sequence-condtioned model

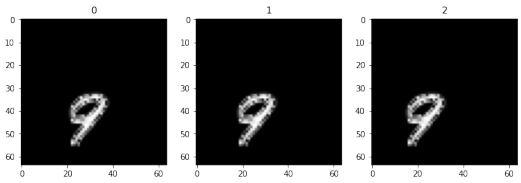

Figure 5.4: Generated sequence from the implemented Moving MNIST dataset.

To establish an experiment in the inital phase of model development that resembled the intended nowcasting use case, the complexity of both the dataset and model had to be increased. The most significant related work done on precipitation nowcasting, outlined in Section 3.4, all make use of input data consisting of sequences making up radar images of previous time steps. To achieve a dataset that mimics the traits of moving weather patterns in a simplified way, a dataset originally intended for stochastic video generation was implemented. The corresponding GitHub repository to a paper published by Denton et al. 2018 provides scripts that enable the creation of a set of video sequences based on the original MNIST dataset [7]. As the dataset was intended for use in the inital experimental phase of the project, the scripts were modified to simplify the model sequences, making model learning more manageable.

The final dataset consisted of *one* randomly sampled handwritten digit, with short deterministic movement between frames consisting of 64×64 grey-level pixels. The implemented dataset generates a batch-sized set of sequences with three frames at training time. Figure 5.4 visualized one of these sequences. Training time is decided based on the selection of batch size and the number of

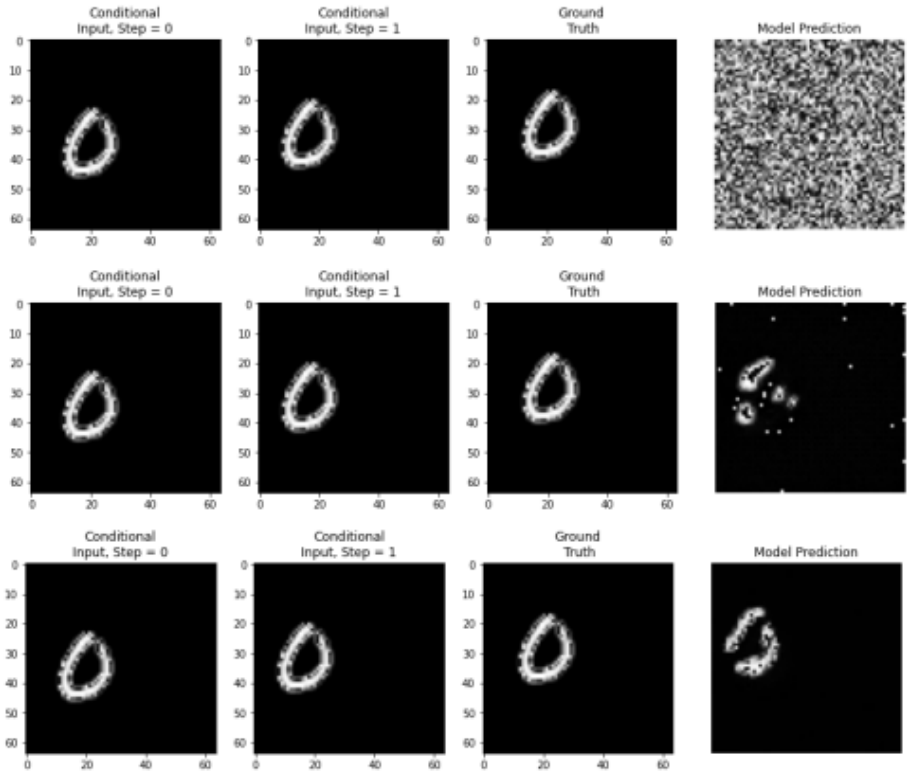

Figure 5.5: Conditioned digit-tracking results from the model trained on the Moving MNIST dataset. Each row represents generated output at epochs zero, five, and twenty, respectively. This plot also acts as a demonstration of how the initial model predictions move from the noise for the untrained model to resembling a handwritten digit in the last row.

epochs. At this stage, however, an increase in both input image size and channel dimensions results in a rapid increase in the computational cost of training and sampling. The model was, therefore, only trained for 20 epochs with a single generated prediction plotted for every fifth epoch. Figure 5.5 visualizes the generated output at three selected epochs. Drawing hard conclusions based on this limited output must be avoided, especially since we deal with probabilistic models. However, some visual remarks can be made about the model's predictive ability. As seen in the far right of the bottom row, after 20 epochs, the model seems to have learned some general digit characteristics regarding the digit *zero*. Compared to the size-normalized and centered digits in the original MNIST data set, this data presents floating digits with no fixed center. It is reasonable to assume that this might prove as an extra challenge for the network when learning these said characteristics, hence the broken curve. However, this also might be the effect of a less *fortunate* drawn sample, as some samples in Figure 5.3 displayed the same broken curves. Furthermore, it is evident that the model has learned some of the mechanics behind the digit displacement, as the location of the sampled digit coincides reasonably well with the ground truth depicted in the frame next to it.

5.1.2 Discussion and analysis of initial results

Through the initial results presented in Figures 5.2, 5.3 and 5.5, it is evident that the implemented concepts behind the diffusion model are in fact able to generate synthetic output that visually resembles samples from the original distribution, as visualized in Figure 5.1. This plot demonstrates that DiffMet is able to both recover the overall shape of the digit from the input shape, as well as track the movement. However, proceeding to the use cases with precipitation data brings a significant increase in complexity, both in the form of model input and wanted output:

- Handwritten digits are constrained by fixed shapes, made up of clearly defined lines and curves. As we proceed to the next section, it will be evident that this is not the case for radar precipitation patterns.
- The digit-conditioned model only had ten possible variations of conditional input, one for each selected hand-drawn digit. The sequence-conditioned model increased this complexity, but clearly defined shapes with little movement still represented the conditional input.
- All though the sequences in the *moving MNIST* data set have moving structures, there occurred no deformation or change in the size of the digits in-between frames. This is in sharp contrast to how radar precipitation patterns can develop frame-to-frame.
- *Moving MNIST* was generated on the fly, and each batch of sequences only exists during training of that specific epoch. In addition, each sequence only consisted of three gray-level images, stored as a datatype known as uint8. This virtually removes the need for considering disc space and computational bottlenecks with data loading. This is not the case as we proceed to model training with the real-life radar data, presented in Section4.1. These radar sequences are made up of 8 frames with radar intensities stored as floating point numbers with multiple channels and represent 60 gigabytes of data that needs its own workflow to be loaded efficiently to the model at training time.

5.2 Precipitation nowcasting with DiffMet

As presented in Section 4.5, generating synthetic radar images from the diffusion model comes at a substantial computational cost. Furthermore, as DiffMet presents a probabilistic approach to precipitation nowcasting, it is of great interest to examine the forecasted distribution in addition to single predictions. In this section, this is examined through the use of the Continuous Ranked Probability Score, presented in 3.3. This, however, requires the model to generate a predictive distribution, demanding an iterative process where the model creates a set amount of predictions for the same conditional input sequence. For the results presented in the preceding sections, all CRPS scores are reported with a sample size of ten predictions for each input sequence. As the dataset used for testing consists of over 1000 sequences, generating the necessary predictive distribution for next-frame predictions with a diffusion model trained

on the 1000 to 4000 diffusions steps used in many published papers would result in 10 - 40.000.000 forward passes through the neural network. When factoring in the floating point calculations required to extract the metrics from a database of this size, it becomes evident that this is a massive computational hurdle.

To mitigate these challenges, these next sections provide metrics and visual results from multiple carefully selected subsets of the original test dataset, computed by models selected as a function of predictive performance and computational efficiency. These subsets consist of different constellations of sequences intended to highlight the model's abilities in different scenarios. We note that optimally, these subsets should have been of larger size to ensure statistical significance, but due to the reasons outlined above, this was not feasible for the scope of this study.

Section 5.2.1 presents the configurations and training scores for four different models. We then proceed to Section 5.2.2, where we compare the two approaches for conditioning the model. Section 5.2.3 examines the model's ability to predict the next frame based on precipitation type. In Section 5.2.4, we proceed to test model performance based on precipitation intensity. Following, in Section 5.2.5, is a presentation of the results from the implemented framework for 20-minute nowcasts, combining the diffusion model with an auto-regressive approach. Finally, Section 5.3 presents a discussion of the results from all the nowcasting experiments and summarizes the findings.

5.2.1 Configurations and model training

This section presents the different model configurations used in training, as well as presenting metrics on model performance through epochs. As machine learning models can be used in a wide selection of scientific fields, there is often a need to tailor the hyperparameters to the specific use case to maximize model performance. This can be done by utilizing various optimization techniques or simply observing what empirically has been found to be optimal for similar model architectures in relevant use cases [60]. However, due to the recent rise in the popularity of diffusion models, the optimal set of hyperparameters for this type of model used in precipitation nowcasting has yet to be established. Furthermore, as made obvious in the previous section, the process of this fine-tuning comes at a computational cost too large for the scope of this project. As a result, the process of optimizing model performance was made based on testing two different sizes of noise schedules for both model implementations. Table 5.1 displays the hyperparameters and relevant components for each model implementation.

Figure 5.6 visualizes mean squared errors for training and validation each of the four implementations through 50 epochs. This figure highlights several important findings from the initial training phase. Firstly, the two best-performing models are the ones that use the increased sampling scheme with 1000 noising steps. There are only small differences separating the concatenated model and the model that utilizes image embeddings. Still, the concatenated model seems to gain an advantage over the embedded model after the 40th epoch. This correlation between increased T-value and increased performance reflects the conclusions shown in most papers [18][33][8]. Both embedding models present validation loss consistently lower than their respective training

Table 5.1: Table of hyperparameters for model training.EM: Image embedding model. CM: Concatenated model.

Hyperparameters				
	CM1	**CM2**	**EM1**	**EM2**
T	300	1000	300	1000
Optimizer	Adam	Adam	Adam	Adam
Batch Size	24	24	24	24
Learning rate	0.005	0.005	0.005	0.005
Epochs	51	51	51	51
Unet	Simple	Simple	Improved	Improved
Beta Schedule	Linear	Linear	Linear	Linear

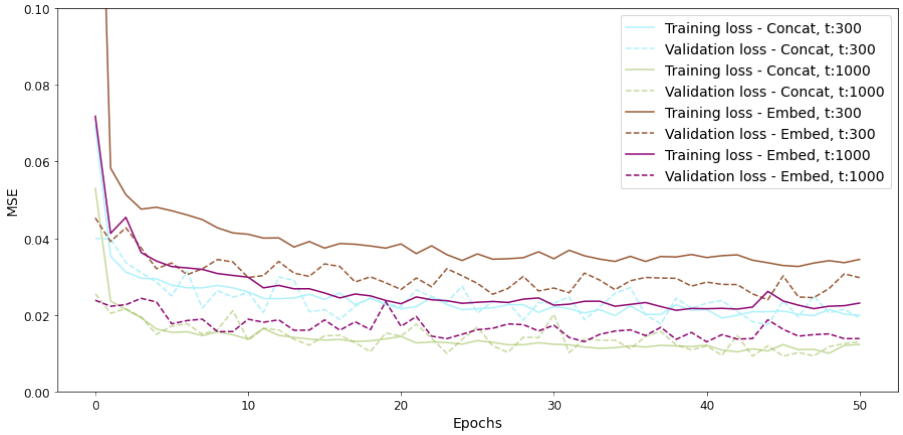

Figure 5.6: This plot visualizes the training- and validation loss (MSE) from 4 different model initializations over 50 epochs.

loss. Both are likely due to the implemented *drop-out* layer, presented in Section 4.4.2, which is only activated during the training phase.

5.2.2 Comparing image embedding input to concatenated input

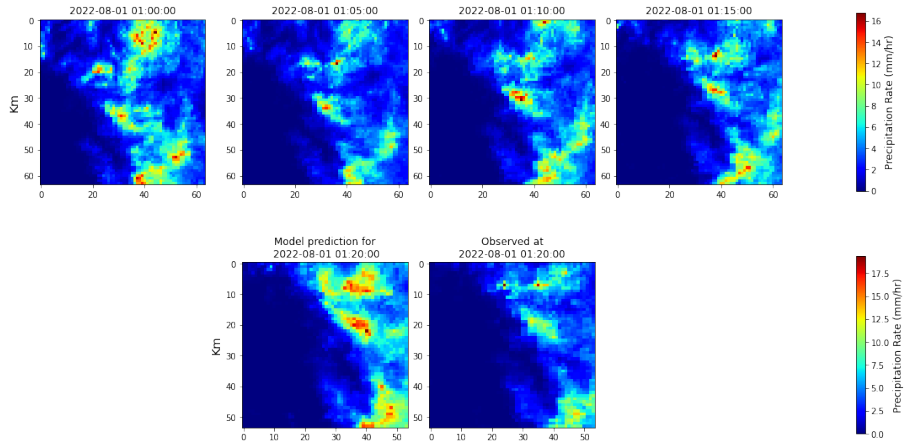

Figure 5.7: Model prediction which achieved the highest CSI when threshold set to two mm/hr. The prediction is made with the concatenated input model and the output received a CSI score of 0.9. Top row: Previous 4 timesteps that make up the conditional input. Bottom left: Model prediction at lead time. Bottom right: Observed radar precipitation rates at lead time. The change in dimensions between the conditional input and the model prediction is due to the spatial context needed for the target patch, as presented in Chapter 4

The previous section presented results during training, where the loss is calculated based on how well the model can predict noise in images for different levels of t. For the remainder of this chapter, the main focus of the metrics will shift to concentrate on the model's abilities to produce valuable precipitation nowcasts and hence introduce the methods implemented for post-training sampling to generate synthetic radar images. To ensure feasible computing times, the two models relevant for use in sampling are both implementations of the models which utilize the sampling scheme where $T = 300$. These two models will be referred to as either *Embed* or *Concat* in figures and plots, where the former is the model utilizing the extra architecture for creating image embeddings, while the latter uses the conditional input directly as concatenated input. We note that if computational capacity were not an issue, further metrics would have been produced with the larger-sized noise schedules. For comparing the two models, a subset consisting of 100 randomly drawn sequences from the original test set was used. This subset is shared between the models to ensure that the metrics produced are done on the same samples. As presented in Section 4.2.5, each sequence loaded to the model have the same 64×64 area cropped based on the image center instead of the random cropping that occurs during training.

To able the calculations for measuring the probabilistic behavior of the model, the implemented scripts generate ten predictions for each input sequence. This creates the nowcast distribution used by the *continous ranked probability score*. For calculating the *critical success index*, a randomly drawn single prediction is used. After sampling all required predictions for the whole subset, the

corresponding results are displayed either as averaged values with corresponding standard deviations or visualized through box plots. We note that as CSI is a pixel-wise comparison between a single prediction and the observed radar image, it lacks the representation of probabilistic behavior as CRPS represents. However, we assume that some of this behavior is represented through the 100 samples represented in the plots.

Figure 5.8 and 5.9 visualizes both metrics, with bars representing the mean values and their corresponding standard deviations. Figure 5.9 presents the CSI scores for three precipitation intensities, at 0.01, 2, and 5 mm/hr. The lowest threshold is set to generate a scenario where we can measure the model's performance to correctly predict rain, disregarding the precipitation intensity levels. This can be of equal value in many use cases where the knowledge of a future *if* there will be rain is more relevant than the exact *amount* of rain. As presented, both models demonstrate close to equal performance at the lowest threshold. There is, however, a large variation for both models, as can be seen from the plotted whiskers. For the two higher thresholds, the performance is drastically reduced for both models. For the threshold set at 2mm/hr, both models produce equal median values. However, the Embed model's distribution mainly centers around this median value, whereas the Concat model displays a much larger variation. For the highest threshold at five mm/hr, the image embedding model shows low performance with a notable part of the CSI scores produced below 0.10. The concatenated model displays higher values but with significant variation.

Figure 5.8 presents the CRPS scores for four different pooling operations. As presented in Section 4.6, these pooling operations remove the location-specific constraints when measuring model performance, as values are compared based on values processed from either a 4×4 or 16×16 pooled out-crop, rather than pixel to pixel. For average-pooling on both grid sizes, the Embed-model displays slightly better performance, with both lower mean values and smaller variation. For the CRPS calculated on the max-pooling operations, the difference in performance increases. For the smaller grid size, the Concat model displays a score with both larger variation and mean value compared to the Embed model. This difference intensifies for the large grid size, where the Embed model performs significantly better. A visualization of the predictive abilities of the concat-model is displayed in Figure 5.7. This sequence achieved the highest CSI score at 0.9, when the threshold was set at two mm/hr. As the median CSI score for 0.01 mm/hr is ≈ 0.75 for both models, this plot demonstrates the model's ability to generate somewhat accurate predictions for certain precipitation scenarios.

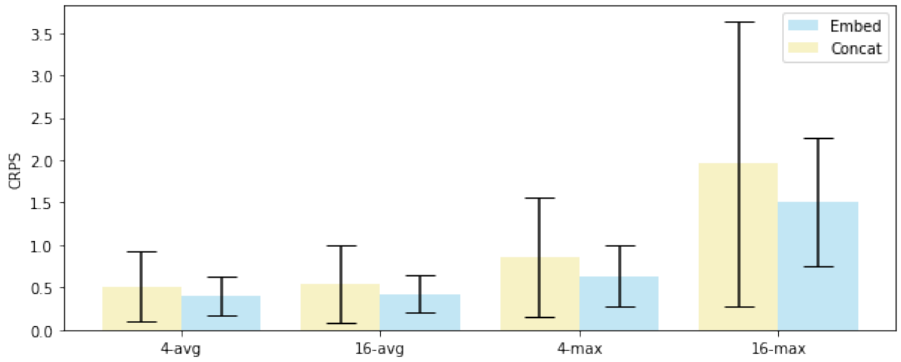

Figure 5.8: CRPS for two different pooling scales on two different pooling methods. Each model is tested on the same subset, consisting of 100 random samples from the test dataset. Each bar represents the relevant CRPS averaged over all 100 samples, with standard deviations. For CRPS, lower is better.

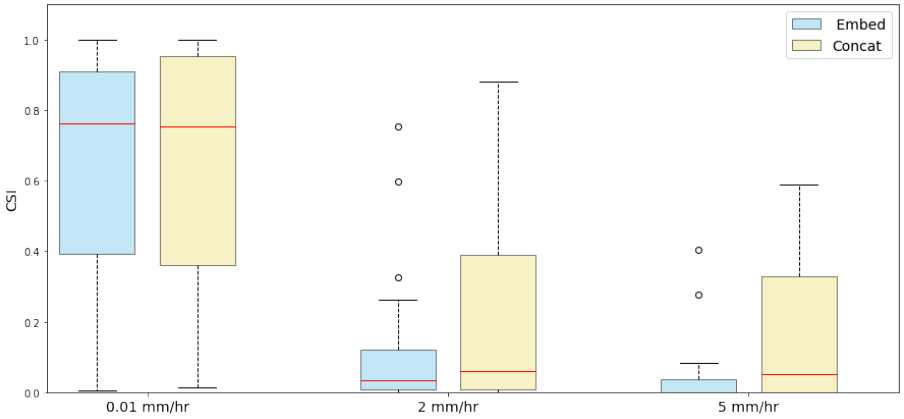

Figure 5.9: CSI for three different threshold precipitation values, tested for both model implementations. Each model is tested on the same subset, consisting of 100 random samples from the test dataset. Each bar represents the relevant CSI averaged over all 100 samples, with a standard deviation. For CSI, higher is better.

5.2.3 Comparing performance on convective and stratiform precipitation

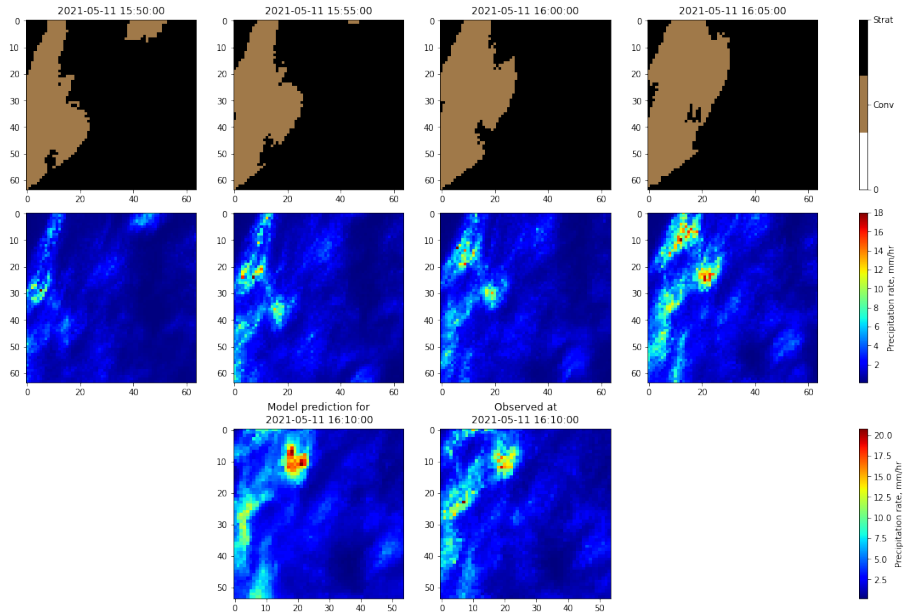

Figure 5.10: Model prediction for a sequence with both convective and stratiform precipitation. Top row: Precipitation plotted as a function of precipitation type. Middle: Radar precipitation rates. Bottom left: Model prediction for lead time. Bottom right: Observed radar precipitation at lead time.

As Section 5.2.2 made it evident that the models indeed are able to generate precipitation nowcasts, both Figure 5.9 and 5.8 presented metrics with occasionally large variations. These variations were present in both the image embedding and concatenated models but to a larger degree in the latter. However, the extended model architecture in the embedding model makes a significant difference in increased sampling times. This resulted in all the following experiments being conducted using the Concat model. To better understand the model performance, further experiments were conducted. These experiments determine if the variations in performance can be linked to certain precipitation types or intensities in the conditional input. The first of these experiments implements two new subsets from the original test data. These new subsets are created using the variable with classified precipitation types, described in Section 4.2.3. As presented in Chapter 2, stratiform precipitation is often present in regions with convective cells. This is also apparent in the dataset, as there are no cases with only convective precipitation. As a result, one subset contains observations where both convective and stratiform are present, whereas the other only contains stratiform precipitation. The size of the two subsets is held at 100 sequences each. We note that for convenience, the subset containing both convective and stratiform precipitation is hereby referred to as solely *Convective*, both in text and figure labels.

Validation metrics are produced the same manner as the ones presented in

Section 5.2.2. For the lowest threshold, Figure 5.11 shows that the convective input produced output with a slightly higher median CSI than stratiform. The convective dataset also produced predictions where most of the CSI score was above ≈ 0.55. Stratiform input resulted in much greater variation. For the 2 mm/hr threshold, this difference was even more prominent. Despite a few outlying samples, most predictions based on stratiform input received a CSI score below 0.20. For the highest threshold, the scores were somewhat more balanced, however still colored by the same difference as the middle threshold. As the change according to thresholds and precipitation types present large variations within, there is difficult to separate any significant model performance purely based on CSI. However, we note that the convective input data shows a pattern of higher-scoring predictions. Large variations are also present in the CRPS, as seen in Figure 5.12. The number of outliers from the convective subset is significantly larger than the stratiform for all pooling operations. For the max-pooling on the largest grid size, the differences in variations were significant.

Figure 5.14 - 5.13 visualizes the input sequences both in terms of the actual radar precipitation intensity as well as a visualization of the engineered feature used for creating the subsets. It is important to note that the conditional input to the network at this point is unchanged and still only consists of radar precipitation rates, as the included plots of precipitation types are solely meant for visualization purposes. Figure 5.10 and 5.13 demonstrate what seems to be two very different precipitation scenarios, although both with strong rainfall intensity. Whereas the convective precipitation in 5.10 seems to be a part of a larger stratiform region with widespread rainfall, the precipitation region in 5.13 appears to be centralized around the convective cell with a concentrated area of high-intensity rainfall. Both predictions achieved a high CSI score. The latter scenario with a concentrated precipitation region was the highest scoring of the 100 sequences, achieving a CSI of 0.84 when thresholded at the middle value of two mm/hr. Figure 5.14 demonstrate the model's predictive abilities on a stratiform precipitation scenario. This prediction achieved the best CSI of all 100 samples when thresholded at two mm/hr at 0.78.

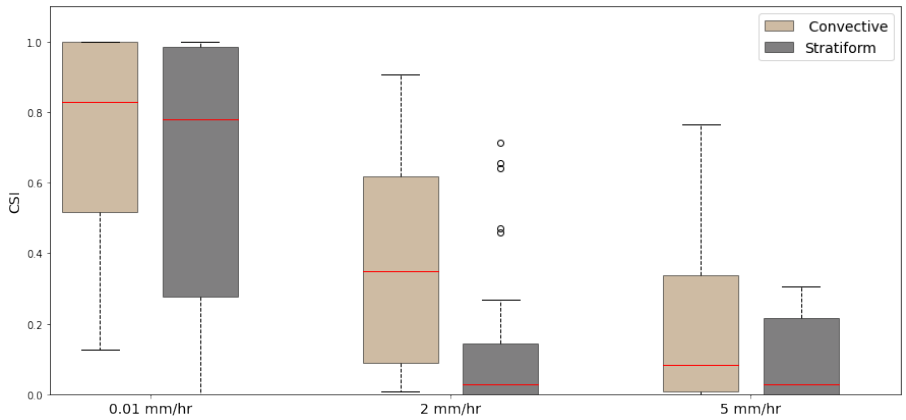

Figure 5.11: CSI at different thresholds. The metrics are computed on two subsets of the test data. One subset contains stratiform precipitation exclusively, while the other has convective precipitation in addition to stratiform. For CSI, higher is better.

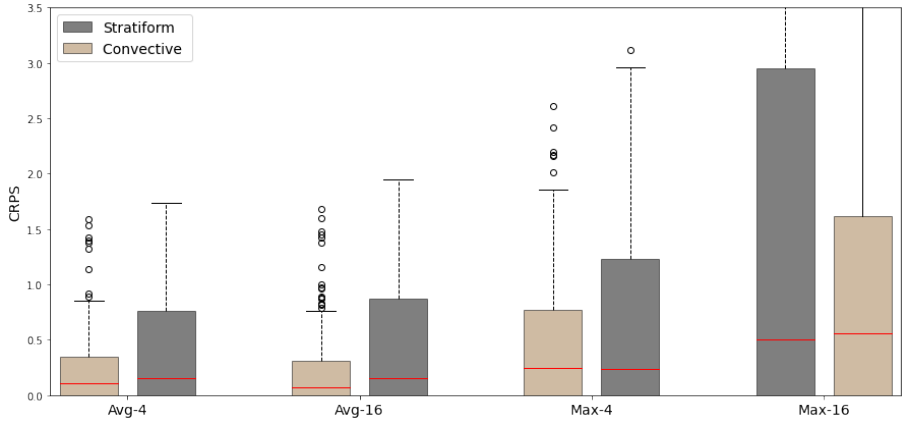

Figure 5.12: CRPS for different pooling operations. The metrics are computed on two subsets of the test data. One subset contains stratiform precipitation exclusively, while the other has convective precipitation in addition to stratiform. For each conditional input sequence, 10 predictions were made by the model. For CRPS, lower is better.

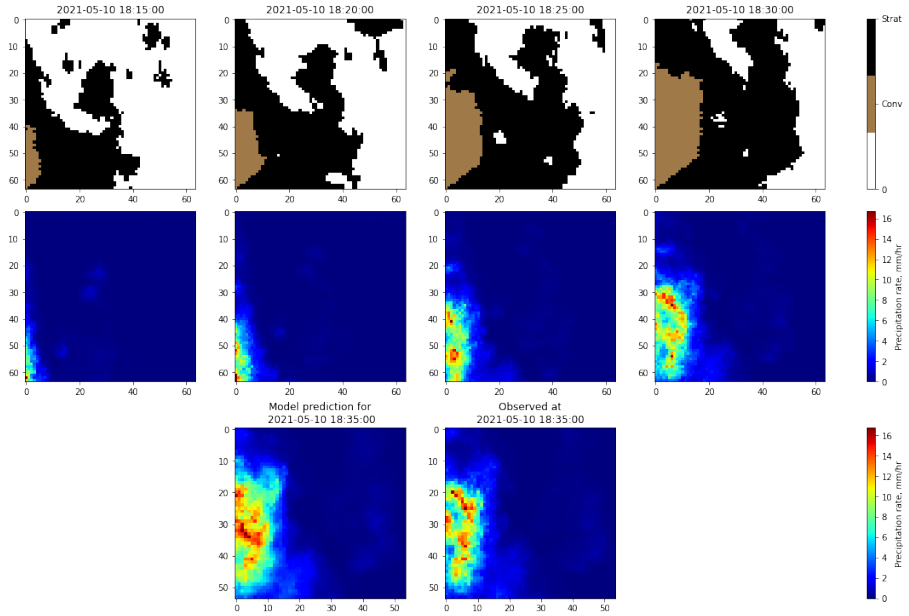

Figure 5.13: Model prediction for a sequence dominated by convective precipitation. Top row: Precipitation plotted as a function of precipitation type. Middle: Radar precipitation rates. Bottom: Left: Model prediction for lead time. Right: Observed radar precipitation at lead time. This prediction got the highest CSI when thresholded at two mm/hr, slightly above 0.9.

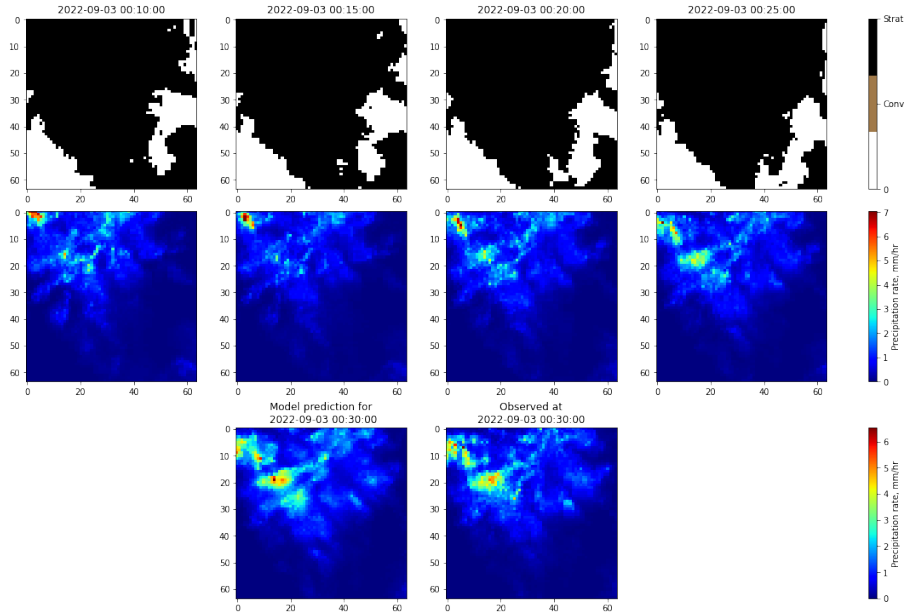

Figure 5.14: Model prediction for a sequence of solely stratiform precipitation. Top row: Precipitation plotted as a function of precipitation type. Middle: Radar precipitation rates. Bottom: Left: Model prediction for lead time. Right: Observed radar precipitation at lead time. This prediction got the highest CSI when thresholded at two mm, at approximately 0.7.

5.2.4 Comparing heavy precipitation to light precipitation

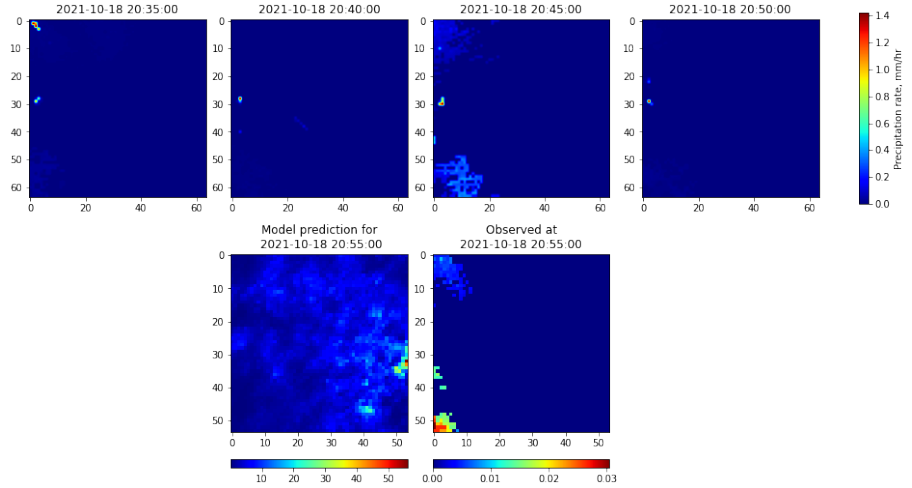

Figure 5.15: Model predictions on a conditional input sequence with little- to no rain. Top: Conditional input. Bottom left: Model prediction. Bottom right: observed radar precipitation at given lead time. This figure illustrates the model's tendency to grossly overestimate precipitation intensity in situations with low-level precipitation prior to lead time.

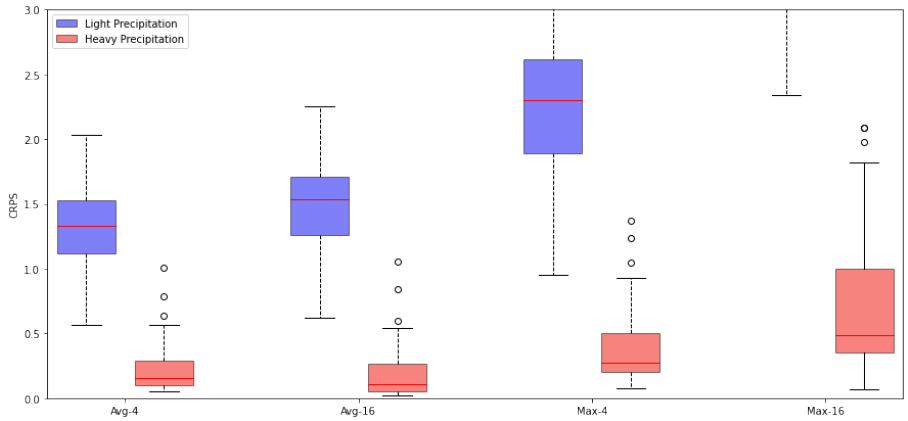

Figure 5.16: CRPS for different pooling operations. The metrics are computed on two subsets of the test data. One subset contains only light precipitation rates, while the other contains heavy precipitation. For each conditional input sequence, 10 predictions were made by the model. For CRPS, lower is better

For the last experiment conducted regarding next-frame predictions, a final data set was constructed. Based on the discovery in Section 5.2.3 regarding differences in model performance based on precipitation types, this final dataset consists of two subsets that separate sequences based on the total sum of radar precipitation rates contained in the last conditional image before lead time. The two thresholds were set at < 30 and >2500 and resulted in just above 100 sequences in each dataset. As the total grid cells in an image are 4096, this results in an average threshold intensity rate at ≈ 0.6 mm/hr per grid cell. This value can rarely be considered *heavy*. Still, the resulting subset created from this threshold is indeed weighted by sequences containing precipitation events either large in size, with high intensity, or a combination. Randomly sampled sequences were removed from each subset, resulting in equally sized subsets containing 100 samples each. Figure 5.16 visualizes the CRPS scores for both subsets, demonstrating significant differences in the model's ability to create nowcasts based on rainfall intensities prior to lead time. The model conditioned on heavy precipitation is superior on both pooling operations for both grid spaces, compared to the model conditioned on the subset with light precipitation. These results are reflected in the CSI score presented in Figure 5.17. For the lowest threshold, the model reaches an average CSI score of 0.92, a result superior to the CSI scores from all previous experiments. As the light precipitation subset contains little- to no precipitation events with rates higher than 1 mm/hr, these boxes are not plotted. Figure 5.15 visualizes a prediction sample from the dataset with light precipitation. This reflects the poor CRPS and CSI scores these precipitation scenarios received and demonstrates how the model overestimates precipitation rates by a considerable amount.

5.2.5 Producing 20-minute nowcasts

With the previous section measuring performance in various scenarios for the next-frame predictive abilities, this section presents the results from full 20-

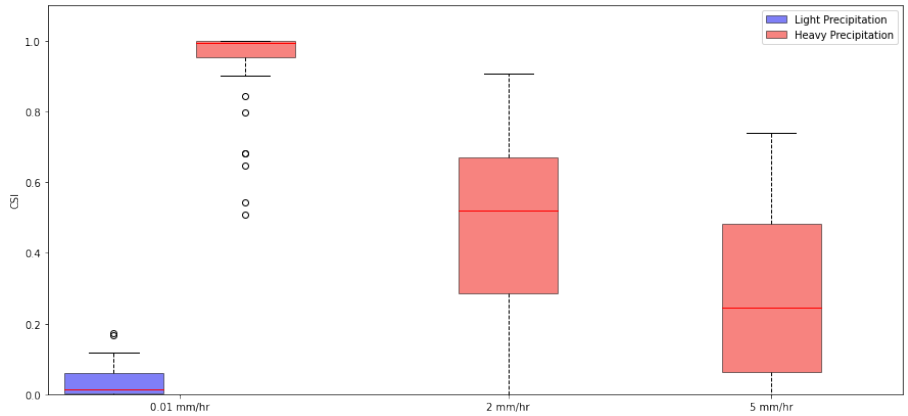

Figure 5.17: CSI scores for light- and heavy precipitation. For light precipitation, the sequences contain little to no regions with precipitation rates above 1 mm/hr, which explains the lack of visual boxes for these threshold values.

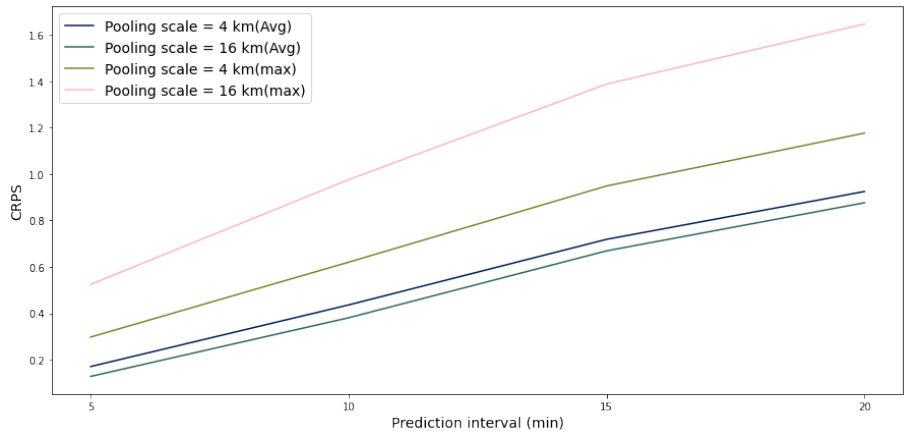

Figure 5.18: CRPS for the four different pooling operations as a function of the prediction interval.

minute nowcasts. These forecasts have a temporal resolution of five minutes, hence producing output sequences consisting of four frames. The results presented in Section 5.2.4 made it evident that the model's performance is strongly correlated to the presence of rainfall in the conditional input. As a result, this section presents results on 20-minute nowcasts based on a subset of the dataset that represents heavy precipitation. As nowcasts of this length present a significant computational increase in terms of sampling, the subset is reduced to 50 randomly drawn sequences.

The implemented methods for sampling 20-minute nowcasts rely on the same next-frame models utilized in the previous sections. This framework was then extended to iteratively produce sequences of arbitrary length, using an autoregressive approach [55]. This implies that each predicted next frame is used as conditional input to predict the following frame. The next-frame, five-minute nowcasts presented in Section 5.2.2 to 5.2.4 relied on the concept

of having input on a larger spatial dimension than the output patch to allow for precipitation displacement [49]. As the U-net relies on input on a fixed set of dimensions, this approach had to be abandoned and rather relies on other implementations, where conditional input is of the same size as the generated prediction [39].

Figure 5.18 visualizes the average CRPS for the four selected pooling operations, plotted against the prediction interval. As the first 5-minute nowcast is based on the observed, precipitation-heavy input, the model achieves low CRPS for all operations. However, when the lead time increases, so do the CRPS score, as expected.

Figure 5.19 to 5.22 visualizes nowcasts made on four different input sequences. These figures demonstrate how the probabilistic nature of diffusion models generates different outputs based on the same conditional input, in contrast to the numerical methods presented in Section 2.2. For visual reasons, only four of ten predicted sequences are presented. Figure 5.19 presents an input sequence where the model generates visually accurate, consistent results compared to the observed sequence. The precipitation region in this scenario looks to be a well-defined structure shaped by a clear difference in precipitation intensities. The second last row in this figure also displays the value of having multiple time steps as conditional input. Despite a bad first predicted frame, the model can still produce visually accurate predictions for the following time steps. Figure 5.20 represents a scenario with intensified rainfall but with the same clear-structured region seen in Figure 5.19. This also represents a scenario where the model is able to predict the precipitation location accurately for the following frames.

In contrast, the precipitation regions shown in the input sequence of Figure 5.21 are presented to showcase a scenario where the input sequence is less well-defined. This input sequence contains precipitation rates in the same range as the one presented in Figure 5.19 but produces significantly different model predictions. In this scenario, the model struggles with predicting accurate rainfall location, in addition to generating output with large variations between predictions.

Figure 5.22 presents a prediction from an input sequence drawn from the subset of light precipitation data and is not part of the CRPS metrics presented in 5.18. The forecast distribution demonstrates how the model overestimates precipitation in all predicted sequences and may visually reflect the metrics presented in Figure 5.16 and 5.17.

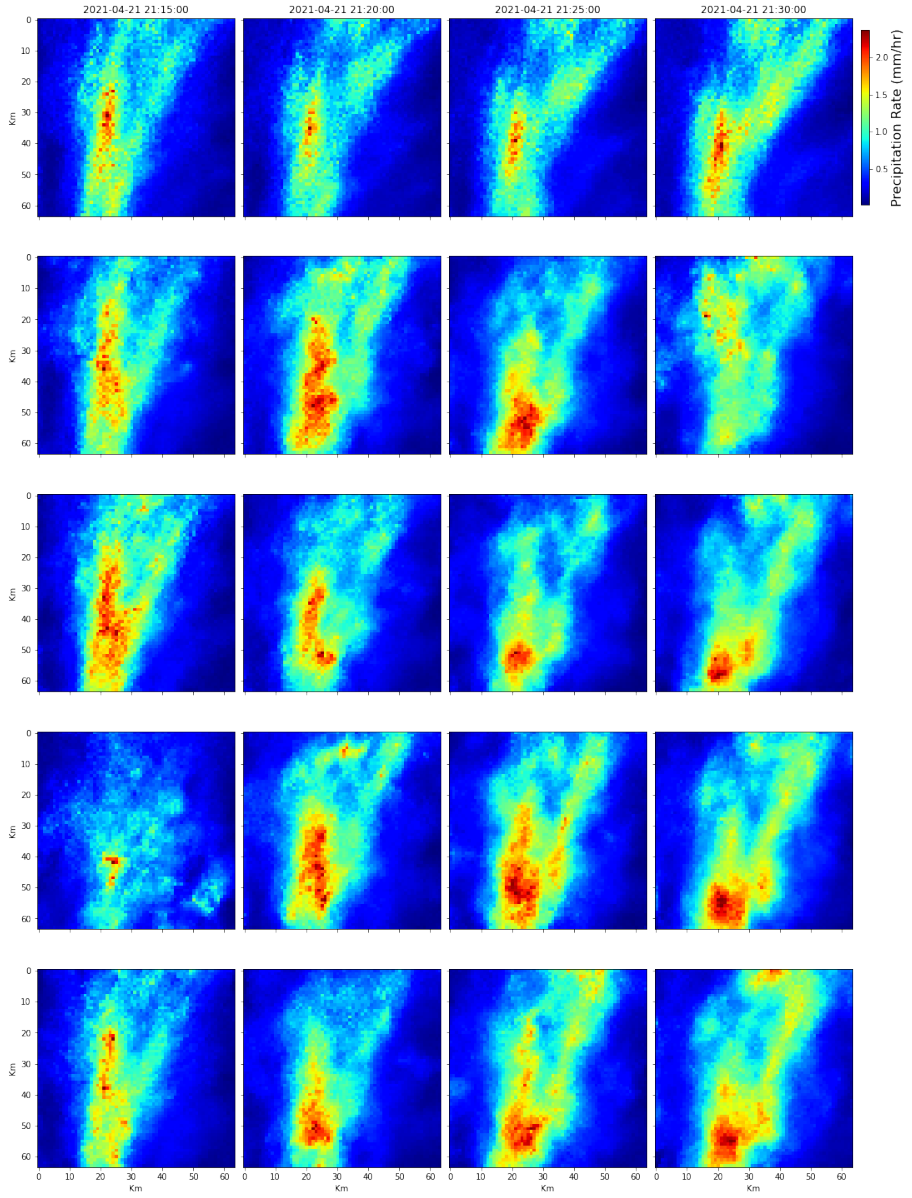

Figure 5.19: Comparison between the observed sequence and sequences predicted by the model. All generated model output is based on the same conditional input sequence. Top row: observed. Bottom four rows: model predictions. This illustrates both the predictive ability of the model, as well as its probabilistic nature. All predicted sequences reflect some of the vertical motion present in the observed sequence.

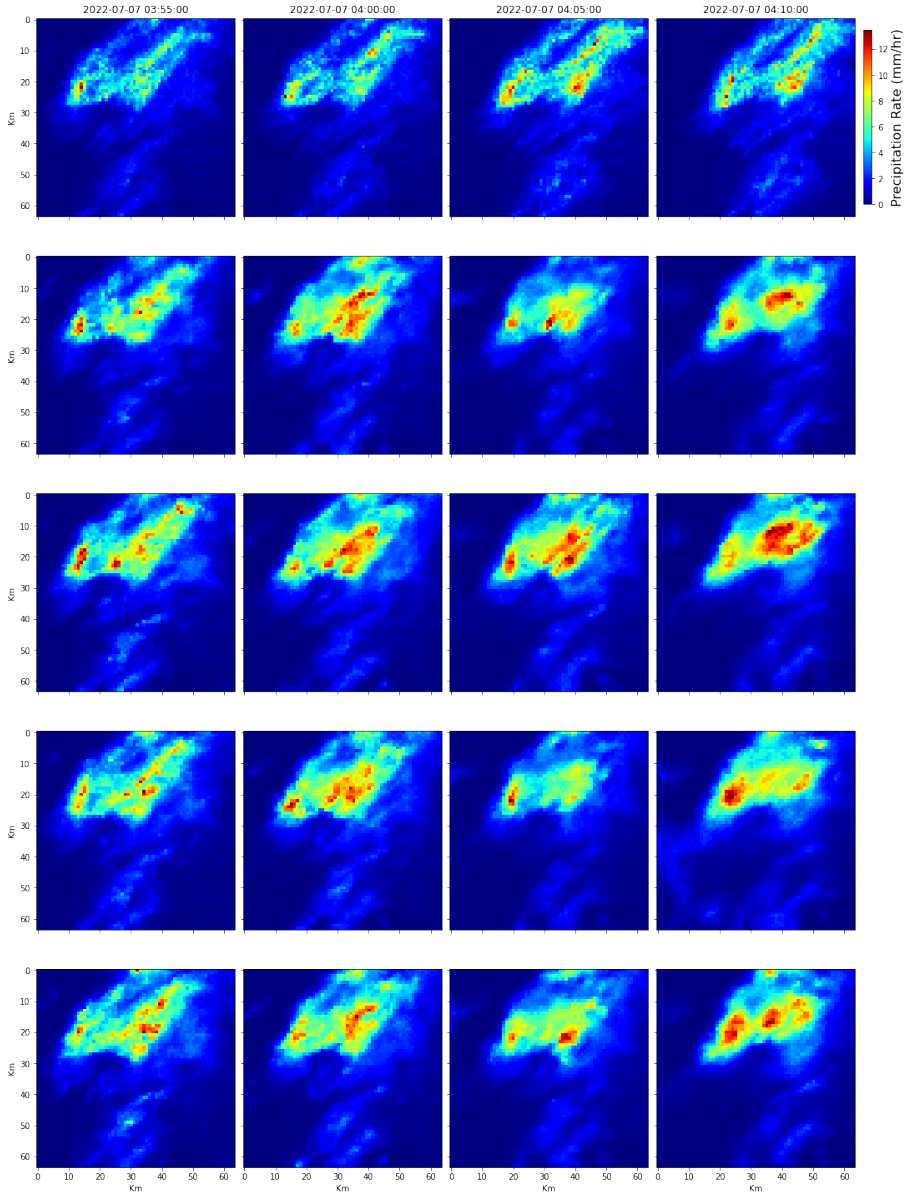

Figure 5.20: Comparison between the observed sequence and sequences predicted by the model. All generated model output is based on the same conditional input sequence. Top row: observed. Bottom four rows: model predictions. This illustrates both the predictive ability of the model, as well as its probabilistic nature.

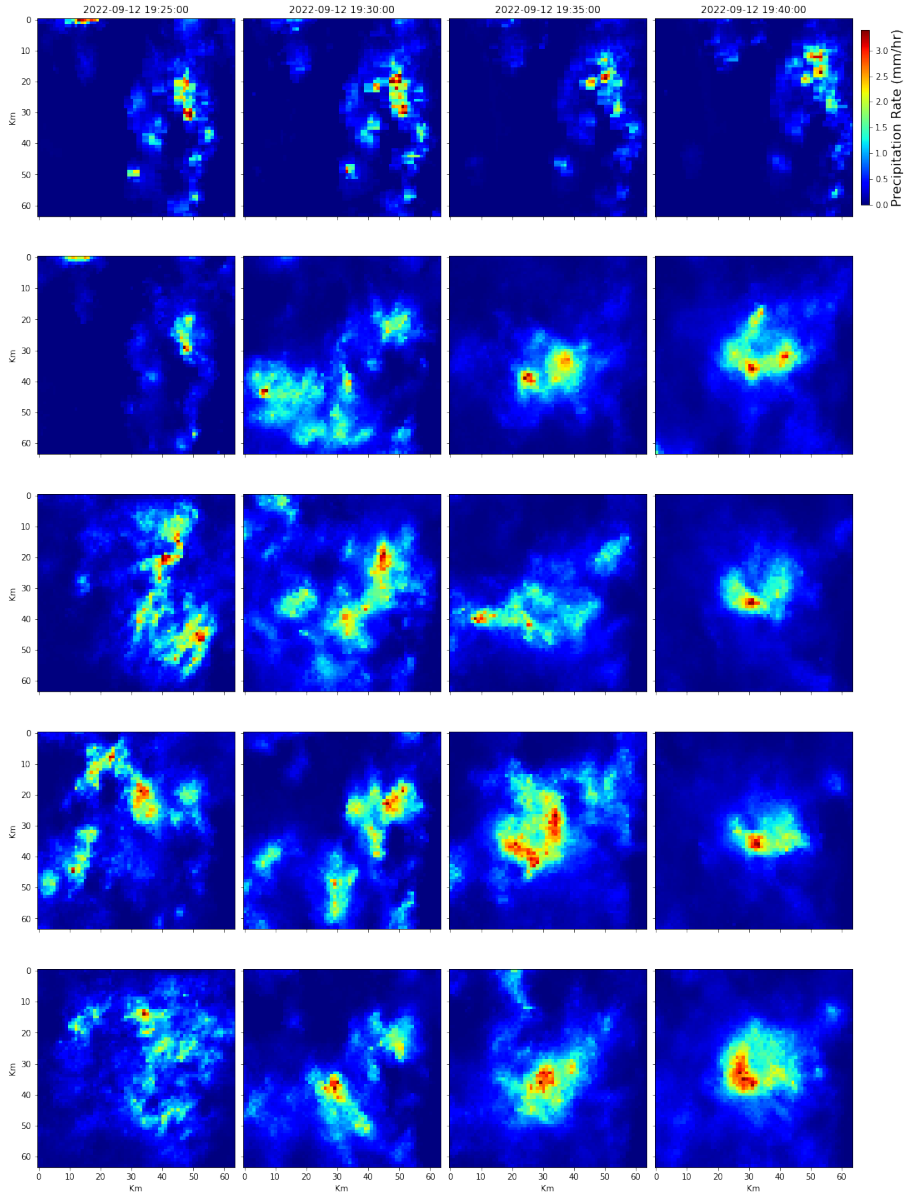

Figure 5.21: Comparison between the observed sequence and sequences predicted by the model. All generated model output is based on the same conditional input sequence. Top row: observed. Bottom four rows: model predictions. This illustrates how the model might struggle to predict patterns without a clear structure or shape.

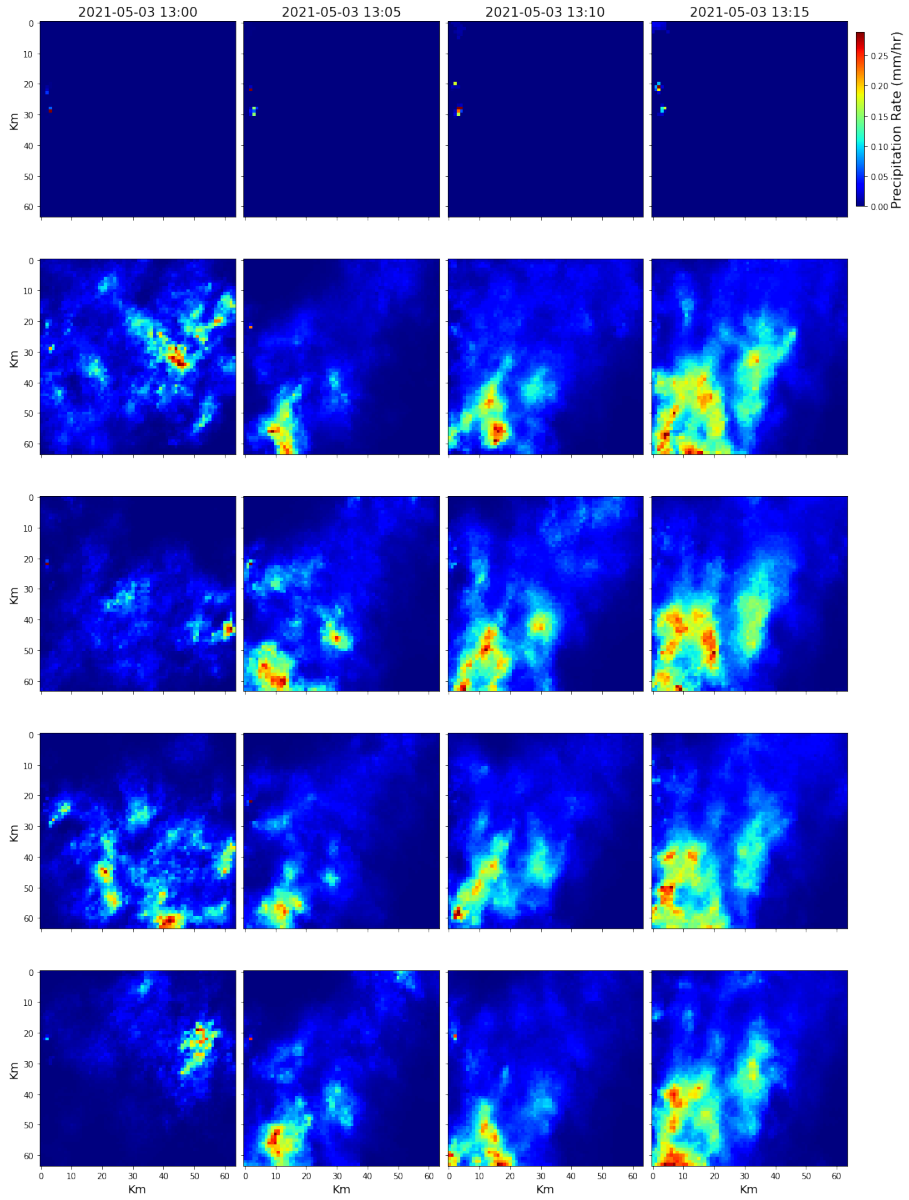

Figure 5.22: Comparison between the observed sequence and sequences predicted by the model. All generated model output is based on the same conditional input sequence. Top row: observed. Bottom four rows: model predictions. This illustrates the model's tendency to overestimate precipitation rates when the conditioned input has little- to no rainfall. The forecast distribution shows that this might very well be a common model behavior.

5.3 Discussion

Section 5.2 presented results demonstrating both the capability and pitfalls when using DiffMet to create precipitation nowcasts for different weather scenarios

and time scales. In this section, we discuss these findings and connect them to theory related to both diffusion models and precipitation.

Our analyses show that it is important to include a wide variety of metrics when evaluating the performance of DiffMet. This is made clear throughout the results but is already evident when comparing the MSE on validation scores for the two chosen models to their corresponding CRPS and CSI scores. As the concatenated model showed better performance in terms of MSE and CSI, it produced forecasts with consistently worse CRPS throughout all pooled versions. The difference between the best performance in terms of CSI and CRPS might also highlight important differences in the generated output based on model implementation. This difference might imply that the model based on image embeddings creates a more generalized prediction, with better-quality nowcasts when pooled over a neighboring area. On the other hand, the model with concatenated input creates forecasts with higher pixel-wise accuracy for precipitation intensities greater or equal to two mm/hr. This is likely caused by the non-processed input, which we assume makes the network less able to generalize on input in terms of grid location. From a nowcasting point-of-view, the ability to make skillful predictions over a generalized area is of greater interest than making accurate predictions on a grid cell level. Furthermore, CSI is biased and dependent on the frequency of events, making CRPS the best indicator for nowcast quality[42]. Furthermore, on the max-pooling of the 16×16 grid, the image embedding model displayed significantly lower CRPS with lower variations. This pooled metric quantifies the model's ability to predict extreme precipitation events, which the World Meteorological Organization emphasizes as one of the most significant use cases for operational nowcasts [56]. We note that with the subset used for creating validation metrics, the image embedding model would be the preferred model for nowcasting. However, a larger sample size would be preferable before concluding.

The CRPS values for the image embedding model also highlight the potential value of processing conditional input before feeding it to the network. The implemented embedding method used in this study represented a novel approach to processing precipitation data, using parts of a ResNet18 model pre-trained on geospatial data. It is reasonable to assume a significant performance increase if this approach is extended to match the functionality of the input processing found in state-of-the-art nowcasting and video generation frameworks [39, 49].

For the results from the experiment conducted on convective and stratiform precipitation, the results differed from what was expected. As presented in Chapter 2, convective precipitation is often defined by intense rainfall over a small region within a short time scale, in contrast to the continuous and uniform precipitation associated with stratiform precipitation. Baseline optical flow models have been shown to struggle to capture these highly non-linear convective events [37]. For the nowcasts produced by DiffMet, the results did not coincide with this.

However, two significant points come into play when analyzing these results. First, one can speculate if the differences in predictive accuracy would be more substantial if the model's lead time were extended to the 90-minute range found in the state-of-the-art nowcasting models. At these timescales, the continuous behavior of stratiform precipitation is likely a significantly more manageable task for the network to predict compared to the intense, short-lived nature of a convective region. Second, the two subsets are thresholded based on the

classification of a limited area of 64 × 64 kilometers. As reflected in Figures 5.10 to 5.14, precipitation regions often exceed this size, resulting in a cropping of the region. This cropping can create instances where the visible precipitation in a region is all classified as stratiform, only to, in reality, be a part of a convective relationship outside the cropped area. Moreover, one can speculate if the intense rainfall over a concentrated area, creating well-defined shapes, makes for an easier predictive task for the neural network. This could be the answer to the increased CSI scores for convective precipitation at higher thresholds, both compared to stratiform and the metrics presented for the two models in Section 5.2.2. This difference in predictive ability is also reflected visually in the distributions for the 20-minute nowcasts presented in Figures 5.19 to 5.22.

The significant differences regarding rainfall intensity presented in Section 5.2.4 are most likely due to the imbalance in the original dataset, dominated by sequences with substantial rainfall. As presented in Section 4.2.4, the dataset was created based on several threshold values set to ensure a sufficient amount of sequences with substantial precipitation while keeping the total size manageable. Increased overall model performance could, therefore, likely have been achieved with training on a more well-balanced dataset. Other implemented nowcasting frameworks have shown this obtained by utilizing an *importance sampling scheme*, which favors samples with heavier rainfall, and at the same time addresses the biases which this introduces [39].

The 20-minute nowcast presents several accurately predicted scenarios, where the visual variation with the nowcast distribution seems minimal. The conditional input for these well-predicted sequences substantiates what seems essential for the model to generate an accurate output, as discussed above. The same applies to less-accurate predictions. As expected, the initial CRPS at the 5-minute prediction interval coincides with values from the previous precipitation-heavy experiment. Also expected is the increase in CRPS as the lead time extends, as far-ahead predictions naturally present a greater challenge due to the chaotic nature of weather systems. The auto-regressive approach is another reason for the decline in model performance. As the predictions made after the first initial iteration are based on previous predictions as conditional input, this can easily lead to propagating errors in the following predictions.

5.4 Summary

In this chapter, we have analyzed the performance of the suggested DiffMet method in various use cases. First, we presented visual results based on the handwritten digits from the MNIST dataset. These results supported the first objective outlined in Section 1.2 by demonstrating that we have successfully developed a conditional model. The model was made conditional in two different scenarios. The first conditional experiment guided the model to output a wanted digit before increasing the complexity with tracking of handwritten digits. This latter experiment acted as a simplified scenario for the proposed nowcasting model use case.

Second, we then proceeded to present the results from nowcasting real-life precipitation events with DiffMet. The model was tested in various weather scenarios to explore model performance and identify weaknesses. We discovered that the improved network architecture with image embeddings as conditional

input outperformed the simpler model architecture when measured by CRPS. However, predictive performance has to be seen in context with computational performance. Since the improved model architecture required substantially more time for sampling, the following experiments were conducted with the simplified, concatenated conditional input. These experiments showcased the models' ability to produce nowcasts for several scenarios, including stratiform and convective precipitation. However, the model suffers from overestimating the amount of future rainfall in scenarios with little- to no rainfall prior to lead time. This is likely caused by an imbalanced dataset.

CHAPTER 6

Conclusion and future work

6.1 Summary and main contributions

In this study, we have developed a diffusion model with conditioning abilities. The model was trained on a compiled dataset of radar precipitation rates from a selected domain of eastern Norway and parts of Sweden. Two different methods for generating conditional output were implemented. The first used a more straightforward approach of directly concatenating the conditional sequence as input to the network, while the second offered a more sophisticated method of processing the input sequence through a second neural network to create image embeddings.

The initial experiments presented in Section 5.1 supported **Objective 1** from Section 1.2 by showcasing the models' ability to produce conditioned output, both in the form of single digits and in the form of tracking movement. The conditional abilities were further showcased throughout the experiments in Section 5.2 when the input data was replaced by the compiled dataset of radar precipitation rates, outlined in **Objective 4**. These experiments also supported **Ojective 2** as they presented CRPS metrics to quantify the uncertainty in predicted future events. Finally, we presented experiments conducted on a collection of subsets from the original test data to identify strengths and weaknesses related to specific weather scenarios, supporting **Objective 3**.

This study has contributed to the development of precipitation nowcasting by presenting a novel approach using diffusion models. We have demonstrated that these models can be conditioned on radar images from previous time steps to generate synthetic radar images of future time steps. Both implemented models showed predictive abilities, but the image-embedding model produced higher-quality nowcasts based on the available dataset. This was evident through better validation metrics in the form of pooled CRPS scores, which is the probabilistic performance metric emphasized by the World Meteorological Organisation [56].

6.2 Suggestions for future work

Creating a balanced dataset that includes an increased amount of samples with low- to no precipitation is a crucial step in improving overall model performance. An *importance-sampling scheme* can also potentially improve the performance by enabling well-balanced datasets which maximize scenarios of interest but still ensures no compromise in performance on low-intensity events [39].

The image embedding module can potentially also be improved, as the input sequences are characterized by complex relationships in time and motion. Recently, a few papers have suggested improvements in this direction. Yang et al. [61] present a framework for stochastic video generation where the diffusion model is conditioned on output from a convolutional recurrent neural network, while Voleti et al. [55] present an embedding module with multi-head self-attention. Ho et al. [19] present a framework for video diffusion models that implements a 3D U-net, factorized over space and time.

Lastly, modifying the loss function to the hybrid variant proposed by Nichol & Dhariwal [33] would enable using a *sampling noise scheme* with significantly fewer steps than the noise scheme used in model training. This would mitigate many of the computational obstacles related to sampling and allow for generating nowcasts over larger areas and longer lead times.

Appendices

APPENDIX A

Source code

The source code for DiffMet is found at https://github.com/gapav/DiffMet

Bibliography

[1] *A Gentle Introduction to Positional Encoding in Transformer Models.* https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/. Accessed: 2022-12-12.

[2] Bandara, W. G. C., Nair, N. G. and Patel, V. M. 'DDPM-CD: Remote Sensing Change Detection using Denoising Diffusion Probabilistic Models'. In: *arXiv preprint arXiv:2206.11892* (2022).

[3] Bernal, J. et al. 'Deep convolutional neural networks for brain image analysis on magnetic resonance imaging: a review'. In: *Artificial Intelligence in Medicine* vol. 95 (2019), pp. 64–81.

[4] Bojarski, M. et al. 'End to end learning for self-driving cars'. In: *arXiv preprint arXiv:1604.07316* (2016).

[5] *Dall-e 2 Open-AI.* https://openai.com/product/dall-e-2. Accessed: 2022-03-03.

[6] Deng, J. et al. 'Imagenet: A large-scale hierarchical image database'. In: *2009 IEEE conference on computer vision and pattern recognition.* Ieee. 2009, pp. 248–255.

[7] Denton, E. and Fergus, R. 'Stochastic video generation with a learned prior'. In: *International conference on machine learning.* PMLR. 2018, pp. 1174–1183.

[8] Dhariwal, P. and Nichol, A. 'Diffusion models beat gans on image synthesis'. In: *Advances in Neural Information Processing Systems* vol. 34 (2021), pp. 8780–8794.

[9] Dumoulin, V. and Visin, F. 'A guide to convolution arithmetic for deep learning'. In: *arXiv preprint arXiv:1603.07285* (2016).

[10] Espeholt, L. et al. 'Skillful Twelve Hour Precipitation Forecasts using Large Context Neural Networks.(2021)'. In: *arXiv preprint arXiv:2111.07470* (2021).

[11] Gers, F. A., Schmidhuber, J. and Cummins, F. 'Learning to forget: Continual prediction with LSTM'. In: *Neural computation* vol. 12, no. 10 (2000), pp. 2451–2471.

[12] Glossary, A. *Glossary of meteorology.* 2009.

[13] Goodfellow, I. 'Nips 2016 tutorial: Generative adversarial networks'. In: *arXiv preprint arXiv:1701.00160* (2016).

[14] Harris, C. R. et al. 'Array programming with NumPy'. In: *Nature* vol. 585, no. 7825 (Sept. 2020), pp. 357–362.

[15] Harris, L. et al. 'A generative deep learning approach to stochastic downscaling of precipitation forecasts'. In: *Journal of Advances in Modeling Earth Systems* vol. 14, no. 10 (2022), e2022MS003120.

[16] He, K. et al. 'Deep residual learning for image recognition'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[17] Hendrycks, D. and Gimpel, K. 'Gaussian error linear units (gelus)'. In: *arXiv preprint arXiv:1606.08415* (2016).

[18] Ho, J., Jain, A. and Abbeel, P. *Denoising Diffusion Probabilistic Models*. 2020.

[19] Ho, J. et al. 'Video diffusion models'. In: *arXiv preprint arXiv:2204.03458* (2022).

[20] Houze Jr, R. A. 'Nimbostratus and the separation of convective and stratiform precipitation'. In: *International Geophysics*. Vol. 104. Elsevier, 2014, pp. 141–163.

[21] Hunter, J. D. 'Matplotlib: A 2D graphics environment'. In: *Computing in Science & Engineering* vol. 9, no. 3 (2007), pp. 90–95.

[22] Iqbal, H. *HarisIqbal88/PlotNeuralNet v1.0.0*. Version v1.0.0. Dec. 2018.

[23] Jha, D. et al. *Kvasir-SEG: A Segmented Polyp Dataset*. 2019.

[24] Jordan, A., Krüger, F. and Lerch, S. 'Evaluating probabilistic forecasts with scoringRules'. In: *arXiv preprint arXiv:1709.04743* (2017).

[25] Kotsiantis, S. B. 'Supervised Machine Learning: A Review of Classification Techniques'. In: *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in EHealth, HCI, Information Retrieval and Pervasive Technologies*. NLD: IOS Press, 2007, pp. 3–24.

[26] Krizhevsky, A. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[27] Krizhevsky, A., Sutskever, I. and Hinton, G. E. 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by Pereira, F. et al. Vol. 25. Curran Associates, Inc., 2012.

[28] Lecun, Y. et al. 'Gradient-based learning applied to document recognition'. In: *Proceedings of the IEEE* vol. 86, no. 11 (1998), pp. 2278–2324.

[29] LeCun, Y., Cortes, C. and Burges, C. 'MNIST handwritten digit database'. In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* vol. 2 (2010).

[30] Lorenz, E. N. 'Predictability: A problem partly solved'. In: *Proc. Seminar on predictability*. Vol. 1. 1. Reading. 1996.

[31] McCulloch, W. S. and Pitts, W. 'A logical calculus of the ideas immanent in nervous activity'. In: *The bulletin of mathematical biophysics* vol. 5, no. 4 (1943), pp. 115–133.

[32] Met Office. *Cartopy: a cartographic python library with a Matplotlib interface*. Exeter, Devon, 2010 - 2015.

[33] Nichol, A. Q. and Dhariwal, P. 'Improved denoising diffusion probabilistic models'. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8162–8171.

[34] Paszke, A. et al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[35] *PlotNeuralNet*. https://github.com/HarisIqbal88/PlotNeuralNet/tree/master. Accessed: 2023-02-02.

[36] Prudden, R. et al. 'A review of radar-based nowcasting of precipitation and applicable machine learning techniques'. In: *arXiv preprint arXiv:2005.04988* (2020).

[37] Pulkkinen, S. et al. 'Pysteps: an open-source Python library for probabilistic precipitation nowcasting (v1. 0)'. In: *Geoscientific Model Development* vol. 12, no. 10 (2019), pp. 4185–4219.

[38] Rasul, K. et al. 'Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting'. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8857–8868.

[39] Ravuri, S. et al. 'Skilful precipitation nowcasting using deep generative models of radar'. In: *Nature* vol. 597, no. 7878 (2021), pp. 672–677.

[40] Ronneberger, O., Fischer, P. and Brox, T. 'U-Net: Convolutional Networks for Biomedical Image Segmentation'. In: *CoRR* vol. abs/1505.04597 (2015). arXiv: 1505.04597.

[41] Samuel, A. L. 'Some Studies in Machine Learning Using the Game of Checkers'. In: *IBM Journal of Research and Development* vol. 3, no. 3 (1959), pp. 210–229.

[42] Schaefer, J. T. 'The critical success index as an indicator of warning skill'. In: *Weather and forecasting* vol. 5, no. 4 (1990), pp. 570–575.

[43] Sharma, S., Sharma, S. and Athaiya, A. 'Activation functions in neural networks'. In: *towards data science* vol. 6, no. 12 (2017), pp. 310–316.

[44] Shi, X. et al. 'Convolutional LSTM network: A machine learning approach for precipitation nowcasting'. In: *Advances in neural information processing systems* vol. 28 (2015).

[45] Simonyan, K. and Zisserman, A. 'Very deep convolutional networks for large-scale image recognition'. In: *arXiv preprint arXiv:1409.1556* (2014).

[46] *Skilful precipitation nowcasting using deep generative models of radar, Nature*. https://www.nature.com/articles/s41586-021-03854-z. Accessed: 2022-07-07.

[47] Sohl-Dickstein, J. et al. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015.

[48] Sohl-Dickstein, J. et al. 'Deep unsupervised learning using nonequilibrium thermodynamics'. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 2256–2265.

[49] Sønderby, C. K. et al. 'Metnet: A neural weather model for precipitation forecasting'. In: *arXiv preprint arXiv:2003.12140* (2020).

[50] *Stable Diffusion Stable AI*. https://stablediffusionweb.com/. Accessed: 2022-03-03.

[51] Steiner, M., Houze Jr, R. A. and Yuter, S. E. 'Climatological characterization of three-dimensional storm structure from operational radar and rain gauge data'. In: *Journal of Applied Meteorology and Climatology* vol. 34, no. 9 (1995), pp. 1978–2007.

[52] Stewart, A. J. et al. 'TorchGeo: Deep Learning With Geospatial Data'. In: *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '22. Seattle, Washington: Association for Computing Machinery, Nov. 2022, pp. 1–12.

[53] Tantau, T. *The TikZ and PGF Packages. Manual for version 3.0.0*. 20th Dec. 2013.

[54] Vaswani, A. et al. 'Attention is all you need'. In: *Advances in neural information processing systems* vol. 30 (2017).

[55] Voleti, V., Jolicoeur-Martineau, A. and Pal, C. 'Masked conditional video diffusion for prediction, generation, and interpolation'. In: *arXiv preprint arXiv:2205.09853* (2022).

[56] Wang, Y. et al. 'Guidelines for nowcasting techniques'. In: *WMO publication, published online: https://library. wmo. int/opac/doc_num. php* vol. 645 (2017).

[57] Weiss, K., Khoshgoftaar, T. M. and Wang, D. 'A survey of transfer learning'. In: *Journal of Big data* vol. 3, no. 1 (2016), pp. 1–40.

[58] Wolleb, J. et al. 'Diffusion models for medical anomaly detection'. In: *Medical Image Computing and Computer Assisted Intervention–MICCAI 2022: 25th International Conference, Singapore, September 18–22, 2022, Proceedings, Part VIII*. Springer. 2022, pp. 35–45.

[59] Wu, Y. and He, K. 'Group normalization'. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[60] Yang, L. and Shami, A. 'On hyperparameter optimization of machine learning algorithms: Theory and practice'. In: *Neurocomputing* vol. 415 (2020), pp. 295–316.

[61] Yang, R., Srivastava, P. and Mandt, S. 'Diffusion probabilistic modeling for video generation'. In: *arXiv preprint arXiv:2203.09481* (2022).

[62] Zamo, M. and Naveau, P. 'Estimation of the continuous ranked probability score with limited information and applications to ensemble weather forecasts'. In: *Mathematical Geosciences* vol. 50, no. 2 (2018), pp. 209–234.

[63] Zhang, L., Zhang, L. and Du, B. 'Deep learning for remote sensing data: A technical tutorial on the state of the art'. In: *IEEE Geoscience and remote sensing magazine* vol. 4, no. 2 (2016), pp. 22–40.