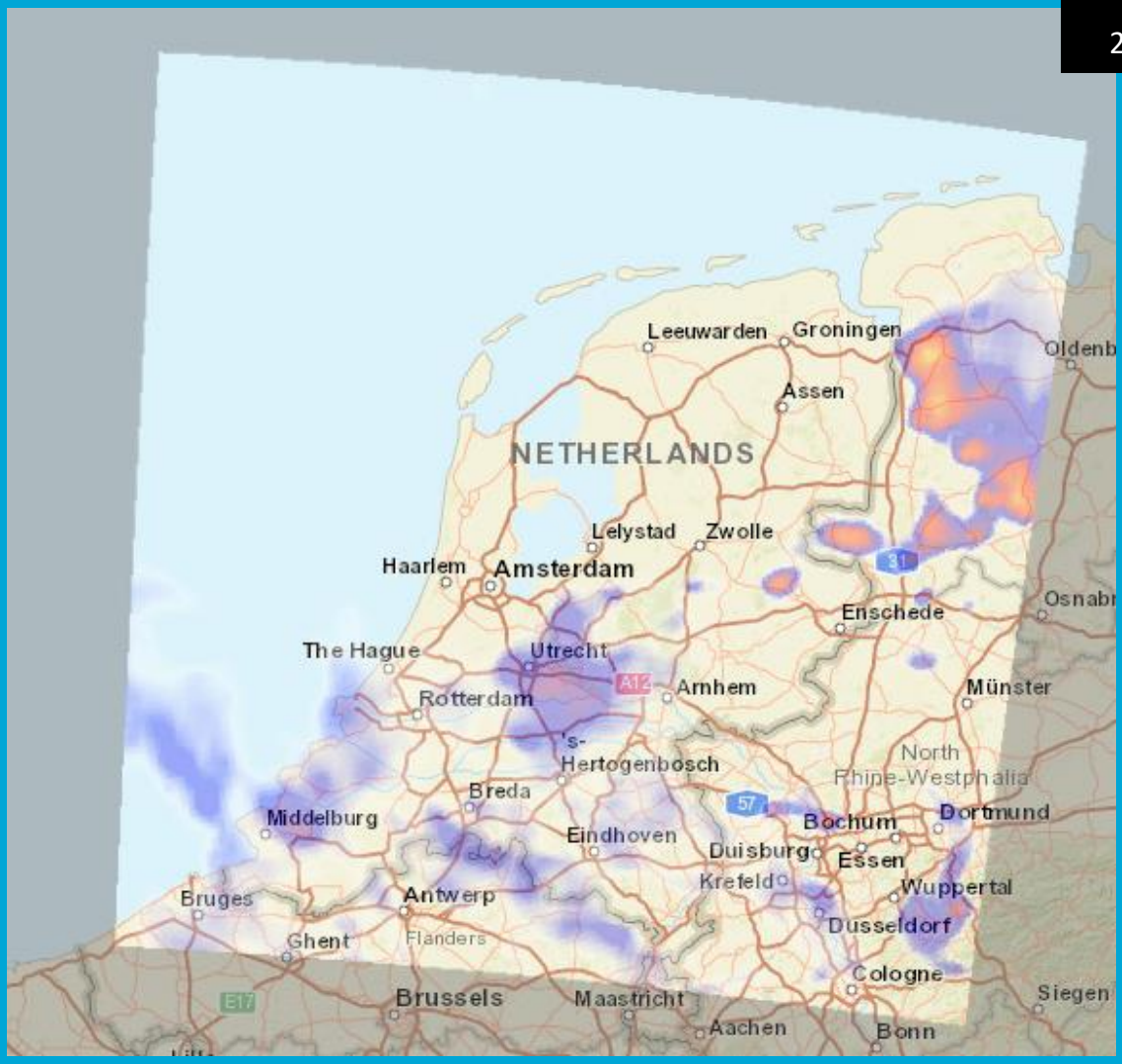
2022



Perceptual losses in precipitation nowcasting:

Exploring limits and potential

D.D. (DIEWERTJE) DEKKER

TU Delft Delft University of Technology

Perceptual losses in precipitation nowcasting:

Exploring limits and potential

by

D.D. (Diewertje) Dekker

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday July 19, 2022 at 9:45 AM.

Student number:	4592190	
Project duration:	December 6, 2021 – July 19, 2022	
Thesis committee:	Dr. M.A. (Marc) Schleiss,	TU Delft, dept. Geoscience & Remote Sensing (CiTG), Chair and Supervisor
	Dr. F. (Francesco) Fioranelli,	TU Delft, dept. Microelectronics (EEMCS)
	Dr. R. (Riccardo) Taormina,	TU Delft, dept. Watermanagement (CiTG)
	Dr. S. (Sukanta) Basu,	TU Delft, dept. Geoscience & Remote Sensing (CiTG)
	Dr. M. (Mattijn) van Hoek,	HKV Lijn in Water

TU Delft

Preface

Firstly, I would like to thank Marc Schleiss, my main supervisor from TU Delft and the chair of the committee. I really enjoyed our weekly meetings, in which we discussed new results and came up with new ideas. But I also want to express my gratitude for all the concrete feedback and for the always quick and extensive responses to my e-mails.

Secondly, I would like to thank the rest of the committee for the feedback and ideas during the update meetings, both orally and written. I also really appreciated the quick responses to questions per e-mail and the papers for inspiration you provided me with, as soon as you saw a new one published that was related to my project.

Furthermore, I want to express my gratitude to Riccardo, for the access to his workstation to train the models and for the help with setting up the anaconda/pip environments at the different computers, and the help with the attempt of using multiple GPUs in parallel. To Niels Jansen for the help with setting up the correct environments at the VR-Lab. To Mattijn van Hoek, Dorien Lugt and Thomas Stolp from HKV, for the exchange of ideas and results, since they were working on a comparable project with the same model.

Lastly, I would like to thank my father Willem-Jan, boyfirend Vidar and friend Ammie, for reviewing my thesis on writing errors at the end. But also for listening to technical/conceptual problems I was facing during my thesis. Telling them to someone out loud really helped to get all the facts straight, and their questions often pinpointed the weakness in my reasoning, due to which I could solve the problem.

D.D. (Diewertje) Dekker
Delft, July 2022

Abstract

Accurate short term rain predictions are important for flood early warning systems, emergency services, energy management and other services that that make weather dependent decisions. Recently introduced machine learning models suffer from blurry and unrealistic predictions at longer lead times, causing poor performance on the rarer heavy rainfall events. The objective of this research was to explore how the loss function in a recurrent, convolutional neural network (TrajGRU, X. Shi et al. (2017)) can be modified, to get sharper and more realistic predictions, without worsening its performance. Six perceptive loss functions, which should better represent how an image is perceived, are thoroughly analysed to understand the reasons behind their functioning in the context of precipitation nowcasting. It was shown that these perceptive losses can lead to an improvement in sharpness for the first lead time, but that the blurriness for longer lead times arises due to the imbalance in the dataset and the uncertainty of the model with respect to the location of the rain. This thesis gets concluded with recommendations on how to deal with these two problems.

Accronyms and Glossary

ANN	Artificial Neural Network
AENN	Adversarial Extrapolation Neural Network
AWL	Auxiliary Weighting Loss
CNN	Convolutional Neural Network
ConvGRU	Convolutional Gated Recurrent Unit
ConvLSTM	Convolutional Long Short-Term Memory
CSI	Critical Success index
FFL	Focal Frequency Loss
GAN	Generative Adversarial Network
GDL	Gradient Difference Loss
GRU	Gated Recurrent Unit
HSS	Heidke Skill Score
IQA	Image Quality Analysis
Lead time	Forecast horizon: the length of time into the future for which predictions are computed
LR	Learning Rate
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MSE	Mean Squared Error
MS-SSIM	Multi Scale Structural SIMilarity
NPV	Normalised Pixel Value
NWP	Numerical Weather Prediction
TrajGRU	Trajectory Gated Recurrent Unit
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RR	Rainfall Rate
SSIM	Structural SIMilarity

Contents

1 Introduction

Precipitation nowcasting is the science that deals with forecasting the location and intensity of rainfall for short lead times from zero up to six hours. Besides the fact that these predictions allow us to know whether it is safe to bike home without rain, short term rainfall forecasts also play a critical role in flood early-warning systems, air traffic control, energy management, emergency services, etc. (De Luca, 2013; Helmis & Nastos, 2012; Prudden et al., 2020; Schmid et al., 2019; Y. Wang et al., 2017).

The current models used to generate short term forecasts are numerical weather prediction (NWP) models and statistical extrapolation techniques (Prudden et al., 2020). NWP models try to simulate the physics in the atmosphere by numerically solving a set of partial differential equations. Extrapolation based methods predict the evolution of rainfall based on the last radar observations, using the assumptions that the rainfall intensity and motion fields are stationary and therefore do not change rapidly. However, both have their shortcomings and, especially for summer precipitation they suffer from quite large errors (Imhoff et al., 2020; Prudden et al., 2020), since they often miss the whole event.

Heavy summer precipitation events are often convective rainfall events, in which warm air quickly raises up, where it immediately condensates and falls down as precipitation (Schumann, 2012). These rainfall events are of shorter duration and higher intensity. They are harder to predict due to their convective nature. NWP models struggle since convection is chaotic and its physics is not well known. Extrapolation based models do not capture them, since these quickly changing events do not match their fundamental assumptions (Prudden et al., 2020).

On top of that, it has been observed that the occurrence and intensities of the convective heavy summer precipitation has increased due to climate change (Buishand et al., 2013; KNMI, n.d.; Lenderink et al., 2011). Climate change causes the atmosphere to have a higher temperature, causing it to hold more moisture, due to which more water can fall as precipitation in one go. This makes it even more important to be able to predict them correctly.

Therefore, several studies turned their interest towards deep learning models. These have the big advantage that you do not need to make any assumptions about the physical processes. However, this does come at the cost of needing a lot of data and a loss of interpretability of the model.

Many different ways of configuring and training machine learning models for rainfall prediction have been proposed over the last couple of years. One of them is Trajectory Gated Recurrent Unit (TrajGRU) from X. Shi et al. (2017) (see Chapter 3), on which van der Kooij (2021) has based her master thesis (van der Kooij, 2021). This thesis is a continuation of her work, with the main objective of making the predictions more realistic.

1.1. Problem statement

One of the problems of TrajGRU is that the predictions look rather blurry and unrealistic, making them less trustworthy for the user. The predictions look this way due to that the rain is spread out over a larger area, with a lower rainfall rate (RR). This causes the heavier rainfall events to not be predicted, which are the events of interest (Ravuri et al., 2021).

The blurriness is a common problem that many machine learning models have. Xingjian et al. (2015) and E. Shi et al. (2018) argue that this is caused by uncertainty in the data, and would therefore not be something that can be easily changed. Even though this uncertainty in the data might exist, we do think that that there are more reasons for the blurriness to arise. Deep learning models are complex. The influence of the model architecture on feature recognition, and also the error criteria (loss functions) they use to learn, are both not fully understood.

Many researchers argue similarly. Ravuri et al. (2021) argue that the blurriness arises due to the lack of constraints in the loss function, which mostly impacts the performance on rarer medium-to-heavy rain events. More specifically, many other researchers (Jing et al., 2019; Lu et al., 2017; Tian et al., 2019; Tran & Song, 2019) argue that the blurriness arises due to the use of the Mean Squared Error (MSE) or Mean Absolute Error (MAE). These losses might make a decent assumption about the global similarity of two images, but not about their local structure (Tran & Song, 2019). Instead they lead to averaging all possible predictions and the loss of details. Foresti et al. (2019) shows this mathematically by decomposing the MSE into its bias and variance components (i.e. MSE = bias^2 + var), showing that it minimises the variance in the error and therefore in the predictions.

To solve this problem of blurry forecasts, three different approaches can be distinguished. The first one simply proposes to change the loss function to a loss that better represents how an image is perceived by humans, using perceptual losses (also referred to as image quality analysis (IQA) metrics). Secondly, the network structure could be changed. A different network structure could allow the model to capture different aspects or give it more freedom in its predictions. Some examples of these network structures are PredNET (predictive neural network) (Sato et al., 2018), which is shown to perform better than TrajGRU; U-Net (Han et al., 2021), which is shown to perform equally well as TrajGRU while being more simplistic; RainPredRNN (Tuyen et al., 2022), which they show to obtain equal performance with less training time; and the combination of different models stacked after each other (Franch et al., 2020; Ko et al., 2022; C. Wang et al., 2021).

The last way to address the blurriness is the use of a Generative Adversarial Network (GAN), as was developed by Goodfellow et al. (2014). Such a model consists of 2 deep learning models that are competing. One of them is a generator that generates synthetic samples. The other is a discriminator, that tries to distinguish between the synthetic samples and the real data. The generator tries to fool the discriminator, causing the distribution of the generated images to become equal to the distribution of the data itself, which lead to more realistic and less blurry forecasts. Several researches have tried the GAN setup in the context of precipitation nowcasting. Examples of these networks are the GA-ConvGRU (Tian et al., 2019), the GAN from Singh et al. (2017), the Adversarial Extrapolation Neural Network (AENN) (Jing et al., 2019) and the Deep Generative Model Reviewed (DGMR) (Ravuri et al., 2021). They have all been shown to outperform the benchmarks they are compared with and to produce more realistic forecasts.

One big advantage of a GAN is that the loss is learned and can therefore penalize any possible structure that differs between output and target (Isola et al., 2017). This can result in better error criteria than what humans can come up with. However, the advantage of simply using a different loss function that does penalise blurriness more, is that it is computationally a lot cheaper, easier to implement and might obtain equally good results.

1.2. Research question

The goal of this research is to improve the realism of rainfall predictions by improving the sharpness and decreasing the blurriness of TrajGRU forecasts. The approach that is investigated, is the use of perceptual loss functions, since we want to understand the effects of the most simplistic solution first. The focus of the predictions are the heavy summer precipitation events, since this type of precipitation has the most impact on society.

The research question of this master thesis is:

"How can the loss function in TrajGRU be modified to get sharper and more realistic predictions, without worsening its performance?"

As was argued by van der Kooij (2021), the performance of a model depends on the end user. In this case, it is meant that the new forecasts should not perform far worse in terms of MSE, MAE and balanced loss. So the aim is to produce a sharp and meaningfull forecast, not a sharp but incorrect forecast.

In an attempt to answer this research question, the TrajGRU model is trained with 6 different loss functions and combinations of them. These models are compared to a benchmark TrajGRU model, so that the effects of the losses can be analysed. The contribution of this work is that it analyses the different loss functions thoroughly, to understand the reasons behind their functioning. This leads to the identification of two main problems for precipitation nowcasting.

The rest of this thesis is structured as follows. Chapter 2 gives an overview of research that uses different perceptive loss functions to prevent blurry predictions. In Chapter 3, the TrajGRU model is described and the benchmark is set. The dataset is presented in Chapter 4. Chapter 5 explains the methodology, both for analysing the effect of the dataset distribution on the benchmark and to answer the research question. The results are shown, analysed and discussed in Chapter 6, after which conclusions and recommendations are given in Chapter 7.

2

Related research

As mentioned in Chapter 1, there are three common solutions to prevent the blurriness of a prediction: A different model structure, a different loss function, or the use of the GAN. In this Chapter, research regarding the use of different loss functions is presented, since it is the approach that is further investigated in this study. We want to understand the effects of the most simplistic solution first and this method is easier to implement and requires less computing resources than changing the model structure.

Tran and Song (2019) proposed the use of a loss functions based on image quality analysis (IQA), to perform precipitation nowcasting with TrajGRU. The ones they investigate are the Structural Similarity (SSIM) and the Multi-Scale SSIM (MS-SSIM) (See Chapter 5 for explanation of the metric). They implemented the loss functions following the method of Zhao et al. (2016), who had already shown that MS-SSIM + MAE can result in the best quality when performing image restoration. Before they analysed the effect of the loss, they did multiple experiments. In the first one, they found that for their dataset, TrajGRU actually performed worse than the Convolutional Gated Recurrent Unit (ConvGRU) and the Convolutional Long-Short Term Memory (ConvLSTM) in terms of MSE, SSIM, MS-SSIM and Pearsons Correlation Coefficient (PCC) on the validating dataset. From this it is important to note, that depending on the dataset and the quality metrics used (X. Shi et al. (2017) used CSI and HSS), different models can appear the best, which was also mentioned by van der Kooij (2021).

In a second experiment, Tran and Song (2019) showed that TrajGRU with a different network structure, the dec-sec2sec, performs the best on their dataset. With this model they tried different loss functions and found that a combination of MSE, MAE and SSIM as loss function, reduces the blurriness visually and in terms of SSIM, MS-SSIM and PCC. In addition, they also found that using MS-SSIM seemed to be ineffective. Moreover, they found that all models quickly decreased in performance with time (after 42 min) for higher rainfall (> 40 dBZ). They suggest that the prediction quality can be enhanced by blending IQA metrics with the GAN model structure and that this will be particularly meaningful for heavy rain.

However, the result of Tran and Song (2019) which suggests that MS-SSIM is ineffective, is in contradiction with the results of Yin et al. (2021). Their study was inspired by Tran and Song (2019) and they also tested the SSIM and MS-SSIM as the loss function, but now with a ConvGRU model and for 2 typical case studies of strong convection. They found that due to multi-layer downscaling, MS-SSIM extracted more radar echo characteristics, and its extrapolation was the most realistic and accurate among all of the loss function schemes. They also found that the MS-SSIM scheme is more efficient in capturing the spatiotemporal correlations, and has the best prediction performance, especially for heavier precipitation.

Li et al. (2021) also used the SSIM to solve the blurry image problem. They analysed the difference between using the SSIM or the MSE to train 4 different model structures (Optical flow, MultiSource Data Model (MSDM), ConvLSTM and U-Net). They show that for precipitation nowcasting, the models trained with SSIM can extract patches of large-valued areas, which the models trained on the MSE tend to smooth out.

Hess and Boers (2022) used a combination of a weighted MSE and the MS-SSIM, where the weighted MSE should account for the imbalance in a precipitation dataset and the MS-SSIM should punish blurry images. Their model obtains a more accurate representation of relative rainfall frequencies and improves the forecast skill of heavy rainfall events by factors ranging from two to above six, depending on the event magnitude.

Veillette et al. (2020) and R. Zhang et al. (2018) both argue that the SSIM does not really represent how we perceive an image, so R. Zhang et al. (2018) proposed a new metric, the Learned Perceptual Image Patch Similarity (LPIPS), based on the neural network. This is a metric that is trained on a dataset of human perceptual similarity judgments, where it learned which deep features in a neural network are important for obtaining an image that is perceived equally good. A deep feature is the consistent response of a node or layer within a hierarchical model to an input that gives a response that's relevant to the model's final output. The metric is computed by an small extra neural network, which has learned weights for the deep features in the different layers of the actual network. Their results show that this result is not restricted to ImageNet-trained features, but holds across different deep architectures and levels of supervision. Jo et al. (2020) have used this loss in combination with a GAN to upsample images 16x to get super-resolution.

However, Hu et al. (2021) argue that this trained LPIPS metric is not sufficient enough for radar data, since meteorology object differ in their structure from normal images. Instead they use the masked style loss to enhance details, by comparing the texture of images. They only look at the foreground areas, ignoring background noise. They show that this loss does improve sharpness significant, also at longer lead times (2h). This loss comes from style transfer tasks (for example giving an picture the style of a Van Gogh's painting). They show that adding the style loss to different models, improves their performance, especially at longer lead times and for higher intensities.

Another attempt of using a different loss function to improve the sharpness, was done by Mathieu et al. (2015), who address the blurriness in video prediction and state that MSE loss causes these, especially when predicting further in the future. For this they proposed 2 solutions, one of which is the use of the Gradient Difference Loss (GDL). This loss penalises gradient differences between the prediction and the true output by only considering the neighbour pixel intensity differences. They only considered the neighbouring pixels to keep the training time low. They analysed the results based on the Peak Signal to Noise Ratio (PSNR), the SSIM and the Sharpness difference and showed that the GDL does cause a small improvement.

Yan et al. (2021) use the GDL loss in the context of precipitation nowcasting, in combination with different model structures (including TrajGRU). They show that it improves the CSI score and the HSS score by 0.007 for all models on average.

The use of the GDL is not often mentioned in the context of precipitation nowcasting, but it is used a lot to sharpen the images in the field of Magnetic Resonance Imaging (MRI): in the context of combining images taken with different MRI settings (Liu et al., 2020), for composing Computed Tomography (CT) images from an MRI image (Nie et al., 2017) or for enhancing the resolution of an MRI (Song et al., 2022). It is also used in the field of action or video prediction, the prediction of what will happen in future video frames from previous ones, which is for example important for autonomous vehicles. Here the GDL is also used to counteract blurriness and emphasise high frequency content (Mathieu et al., 2015; Xu et al., 2018; Zhao & Wildes, 2021). Moreover, it has also been used in the context of predicting the El Niño-Southern Oscillation (ENSO) (Mu et al., 2022).

Conclusion

From the papers that have tested the SSIM and MS-SSIM, it can be concluded that the effect of a model with a certain loss function can differ depending on the model architecture and the dataset (shown by Li et al. (2021) and from the difference between Tran and Song (2019)and Yin et al. (2021)). This thesis will further investigate these dependencies on the dataset for several loss functions, using one version of the TrajGRU model.

3

Background - TrajGRU

3.1. Neural network unit: TrajGRU

An Artificial Neural Network (ANN) consists of input layers, hidden layers and output layers. When information travels between the layers in the forward direction, a network is called a feed forward neural network.

When connections are added between nodes within the same layer and with the previous layers, the neural network is called a Recurrent Neural Network (RNN). These extra connections enable the model to capture temporal information and therefore allow for sequence learning. However, these extra connections can lead to the vanishing problem where the weights of the connections go to zero, or the exploding gradient problem where the weights become really large. To prevent these problems from occurring, the Long Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU) were proposed (A. Zhang et al., 2021). Both can distinguish between information that needs to be remembered and can be forgotten. This is saved in a 'hidden state'. The advantage of a GRU is that is has less parameters than the LSTM (Gao et al., 2020), causing it to use less memory and execute faster, which is important for precipitation nowcasting to obtain predictions in real-time. However it might be less accurate than the LSTM on large datasets (S. Wang et al., 2020).

If you want to apply these neural networks to images, you use a network structure where the LSTM or GRU are used in alternation with a convolution layer, to downsample the image and extract the important features at different scales (A. Zhang et al., 2021). These convolutions allow the network to learn spatial information. Therefore, convGRU and convLSTM can learn both spatial and temporal information. However, they only use one filter for different locations (location-invariant) and have therefore problems with rotation and scaling motions. Moreover, this causes that they can only learn familiar patterns (Tran & Song, 2019). This means that they would perform well if the training and testing dataset are consistent with each other, but not if the patterns have changed between them. Since the precipitation events are changing due to climate change, these patterns have changed and this location-invariance forms a problem.

To solve that problem, X. Shi et al. (2017) proposed a Trajectory GRU (TrajGRU). It allows spatial connections between the hidden state and the new input (these connections are defined in the filter) to change over time, depending on the direction of flow within an image sequence. This enables TrajGRU to follow rotation and scaling motions, but also to break local shapes and make a stronger assumption about the global changes between two time steps, allowing them to invent unseen patterns.

For a more detailed description on how the TrajGRU unit is built, see Section 2.4 of the thesis of van der Kooij (2021).

3.2. TrajGRU architecture

The TrajGRU model consists of TrajGRU units and convolution layers that are alternating with each other, in an encoder-forecaster structure. The encoder first downsamples an image three times, after which it gets upsampled again in the forecaster. This structure with parameter specifications is visible in the tables in Figure 3.2 and is further discussed in the next subsection.

The architecture used in this research takes 5 input images and yields 20 predictions. A schematic overview of this model is shown in Figure 3.1, where every block is one of the components specified in the table with the model parameter settings from van der Kooij (2021) (see Figure 3.2).

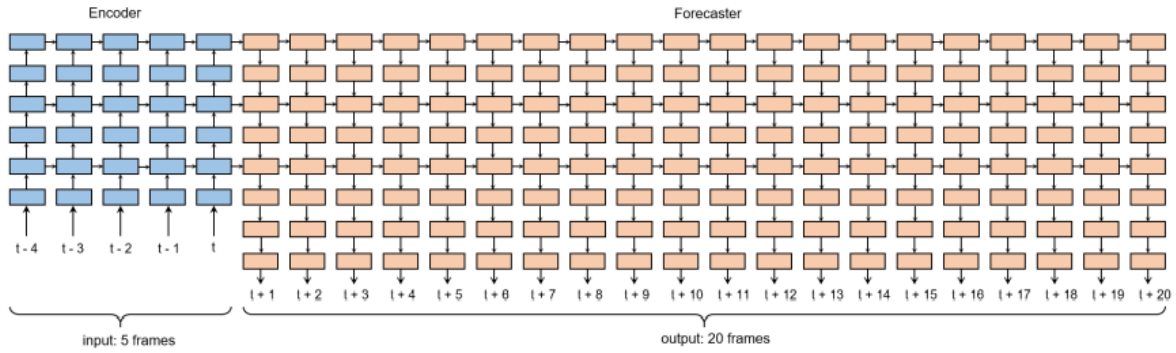


Figure 3.1: The model set up of TrajGRU with 5 input frames and 20 output frames, according to the architecture used by van der Kooij (2021). (Source: Thesis van der Kooij (2021))

3.2.1. Model parameter settings

While analysing the code that van der Kooij (2021) had used to train and run the TrajGRU model, it was found that the parameter settings (e.g. kernel size, stride, padding, etc.) of the TrajGRU architecture were slightly different from how X. Shi et al. (2017) presented them in their paper (see Figure 3.2). van der Kooij (2021) did not make these changes herself. The changes occurred while transferring the code from python with package MXNet (https://github.com/sxjscience/HKO-7) to the package PyTorch (https://github.com/Hzzone/Precipitation-Nowcasting). The effects of these changes in settings are analysed and described, which can be found in Appendix Section A.1. It was concluded that visually not many differences appeared, leading to the decision to keep the settings from van der Kooij (2021) for this research. However, the analysis of the different scores on the testing dataset indicates that the model settings from X. Shi et al. (2017) do produce better forecasts for longer lead times and higher RR. Therefore we would recommend to switch to the model settings from X. Shi et al. (2017) in the future.

van der Kooij (2021)

Name	Kernel	Stride	Pad	L	Ch I/O	In Res	Out Res	Type	In	In State
econv	7 x 7	5 x 5	1 x 1	-	1/8	480 x 480	96 x 96	Conv	input	-
ernn1	3 x 3	1 x 1	1 x 1	13	8/64	96 x 96	96 x 96	TrajGRU	econv1	-
edown1	5 x 5	3 x 3	1 x 1	-	64/192	96 x 96	32 x 32	Conv	ernn1	-
ernn2	3 x 3	1 x 1	1 x 1	13	192/192	32 x 32	32 x 32	TrajGRU	edown1	-
edown2	3 x 3	2 x 2	1 x 1	-	192/192	32 x 32	16 x 16	Conv	ernn2	-
ernn3	3 x 3	1 x 1	1 x 1	9	192/192	16 x 16	16 x 16	TrajGRU	edown2	-
frnn1	3 x 3	1 x 1	1 x 1	9	192/192	16 x 16	16 x 16	TrajGRU	-	ernn3
fup1	4 x 4	2 x 2	1 x 1	-	192/192	16 x 16	32 x 32	Deconv	frnn1	-
frnn2	3 x 3	1 x 1	1 x 1	13	192/192	32 x 32	32 x 32	TrajGRU	fup1	ernn2
fup2	5 x 5	3 x 3	1 x 1	-	192/64	32 x 32	96 x 96	Deconv	frnn2	-
frnn3	3 x 3	1 x 1	1 x 1	13	64/64	96 x 96	96 x 96	TrajGRU	fup2	ernn1
fup3	7 x 7	5 x 5	1 x 1	-	64/8	96 x 96	480 x 480	Deconv	frnn3	-
fconv4	3 x 3	1 x 1	0 x 0	-	8/8	480 x 480	480 x 480	Conv	fup3	-
fconv5	1 x 1	1 x 1	0 x 0	-	8/1	480 x 480	480 x 480	Conv	fconv4	-

Shi et al. (2017)

Name	In Kernel	In Stride	In Pad	L	Ch I/O	In Res	Out Res	Type	In	In State
econv1	7×7	5×5	1×1	-	4/8	480×480	96×96	Conv	in	-
ernn1	3×3	1×1	1×1	13	8/64	96×96	96×96	TrajGRU	econv1	-
edown1	5×5	3×3	1×1	-	64/64	96×96	32×32	Conv	ernn1	-
ernn2	3×3	1×1	1×1	13	64/192	32×32	32×32	TrajGRU	edown1	-
edown2	3×3	2×2	1×1	-	192/192	32×32	16×16	Conv	ernn2	-
ernn3	3×3	1×1	1×1	9	192/192	16×16	16×16	TrajGRU	edown2	-
frnn1	3×3	1×1	1×1	9	192/192	16×16	16×16	TrajGRU	-	ernn3
fup1	4×4	2×2	1×1	-	192/192	16×16	32×32	Deconv	frnn1	-
frnn2	3×3	1×1	1×1	13	192/192	32×32	32×32	TrajGRU	fup1	ernn2
fup2	5×5	3×3	1×1	-	192/192	32×32	96×96	Deconv	frnn2	-
frnn3	3×3	1×1	1×1	13	192/64	96×96	96×96	TrajGRU	fup2	ernn1
fdeconv4	7×7	5×5	1×1	-	64/8	96×96	480×480	Deconv	frnn3	-
fconv5	1×1	1×1	0×0	-	8/1	480×480	480×480	Conv	fdeconv4	-

- Yellow = Difference van der Kooij (2021) and Shi et al. (2017)
- Blue = Does not exist in Shi, only in van der Kooij (2021)
- Red = Different compared to code

Figure 3.2: The differences in the reported and used TrajGRU settings in the thesis of van der Kooij (2021) and the paper of X. Shi et al. (2017)

3.3. Training loss: Balanced MAE+MSE loss

To train, the model needs a function to evaluate how much the predictions deviate from the observations. Standard loss functions are the Mean Absolute Error (MAE), Mean Squared Error (MSE) or Root Mean Squared Error (RMSE).

It is also common to combine different losses together using weighted averages. For example, X. Shi et al. (2017) proposed a balanced loss function to put more emphasis on pixels with heavy precipitation. The loss is a combination of the MAE and the MSE, where both are taken with an equal weight of 1 (see Equation 3.1). But within them, pixels with a higher observed intensity of rain are weighted more than pixels with lower rain intensities (Equation 3.2 and 3.3). X. Shi et al. (2017) had divided the pixels in different categories and gave each category a weighting that increases with the RR. van der Kooij (2021) had adjusted this to a continuous loss function, where the weights are equal to the RR in mm/h themselves, with a minimum of 1 and maximum of 30, as can be seen in Equation 3.4.

$$\text{Loss} = \text{BMSE} + \text{BMAE} \tag{3.1}$$

$$\text{BMSE} = \frac{1}{20 \cdot 360 \cdot 360} \sum_{n=1}^{20} \sum_{i=1}^{360} \sum_{j=1}^{360} w_{n,i,j} \left(F_{n,i,j} - O_{n,i,j}\right)^2 \tag{3.2}$$

$$\text{BMAE} = \frac{1}{20 \cdot 360 \cdot 360} \sum_{n=1}^{20} \sum_{i=1}^{360} \sum_{j=1}^{360} w_{n,i,j} \left|F_{n,i,j} - O_{n,i,j}\right| \tag{3.3}$$

$$w_{n,i,j} = \begin{cases} 1 & \text{if } R_{n,i,j} \leq 1 \\ R_{n,i,j} & \text{if } 1 < R_{n,i,j} \leq 30 \\ 30 & \text{if } R_{n,i,j} > 30 \end{cases} \tag{3.4}$$

In these equations, n is the index for one of the 20 predictions. $F_{n,i,j}$, $O_{n,i,j}$ and $w_{n,i,j}$ are the forecast, observation and weights corresponding to the $(i,j)^{th}$ pixel in the n^{th} image. As can be seen, the sums for the pixels go up to 360, since we only focus on the research domain, as presented in Section 4.2 and depicted in Figure 4.1.

3.4. Definition of the benchmark

This research takes the best performing model of van der Kooij (2021) as a benchmark. It is not compared with any other models, since van der Kooij (2021) had already shown that this one performed better than S-PROG (Spectral PROGnosis, a deterministic extrapolation based model from Seed (2003)).

This means that the benchmark consists of TrajGRU, with the model parameter settings in the top table in Figure 3.2. The model was trained with the Balanced MAE+MSE loss as described in Section 3.3, using the largest possible dataset, that is preprocessed as described in Section 4.2, containing events that are selected according to the method described in Section 4.3.

This model was trained on an 8 GB Nvidia GeForce RT2080 GPU with batch size 2, for 100,000 iterations. It made use of a learning rate (LR) scheduler with step-decay: a decay-factor of LR · 0.1 at the 30,000th and 60,000th iteration. The Adam optimizer was used to optimize the loss.

4

Data

In this Chapter, the Royal Netherlands Meteorological Institute (KNMI) radar dataset is introduced, after which the prepossessing and event selection methods of van der Kooij (2021) are summarised. Lastly, further analysis into the distribution of the data is presented, together with a hypothesis of what its influence might be on the training of the model.

4.1. KNMI radar data

The Royal Netherlands Meteorological Institute (KNMI) has two polarimetric C-band (5.3 cm wavelength) Doppler weather radars that together cover the area of the Netherlands (Beekhuis & Holleman, 2008). A radar emits electromagnetic radiation at a given wavelength and measures the backscattered power. The latter is then converted to reflectivity z [mm^6/m^3] and, from there, to reflectivity Z [dBZ] or to rainfall rates R [mm/h], using Equation 4.1. (Marshall et al., 1955)

$$z = 10^{Z/10} = 200R^{1.6} \tag{4.1}$$

Both radars operate at ground level, but their measurements are projected onto a grid of stereographic coordinates with a fixed elevation of 1500m. Their measurements are merged together into one radar composite, by taking the weighted average, for which the weight is determined by the distance of that point to the radar. This radar composite has a 1 km x 1 km spatial resolution and a 5-minute temporal resolution. The data is available in hdf5 files, containing 2D arrays representing a 700x765 pixel image with 8-bit (0-255) radar reflectivity values. These 8-bit integers can be transformed into a reflectivity value in dBZ with Equation 4.2. The pixels for which no measurements are available, because they are outside of the range of the radars or the radar was turned off, have a no-data value of 255.

$$Z = (\text{pixel value} \cdot 0.5) - 32 \tag{4.2}$$

These radar composites are available as described above, with minimal processing: no noise filtering and no bias-correction using gauges. The reason these unprocessed composites are used, instead of further processed and therefore more accurate composites, is that the processed bias-corrected products are not available in real-time, which is crucial for performing nowcasting.

The radar composites can be obtained through the KNMI data platform via an API (https://dataplatform.knmi.nl/dataset/radar-tar-refl-composites-1-0). The product name of the archive is `radar_tar_refl_composites/ versions/1.0`. The near real-time images are available after 3 to 4 minutes and have product name `radar_reflectivity_composites/versions/2.0`.

In total, dataset covers 2008-present. However, only 2008-2020 is used, since using the same data as in the benchmark (described in Section 3.4) enables the comparison of the results.

4.2. Data preprocessing

van der Kooij (2021) had chosen to only remove clutter with high reflectivity values, since these would confuse the model most when training to predict heavy precipitation. She found that the majority of these high reflectivity values (> 50 dBZ) was of non-meteorological nature such as ship tracks and offshore wind farms. Therefore she set all the pixels with values higher than 50 dBZ to 0 dBZ.

Furthermore, TrajGRU was configured to take in 480x480 sized images. Therefore, the images are cropped as shown in Figure 4.1. The performance of the model is only assessed over a central 360x360 domain area, hereinafter referred to as the 'research domain'. The main reason for this is to avoid border effects. The model needs to be able to see the rainfall cells coming in order to make a good prediction, which is not possible for the pixels that are close to the border.

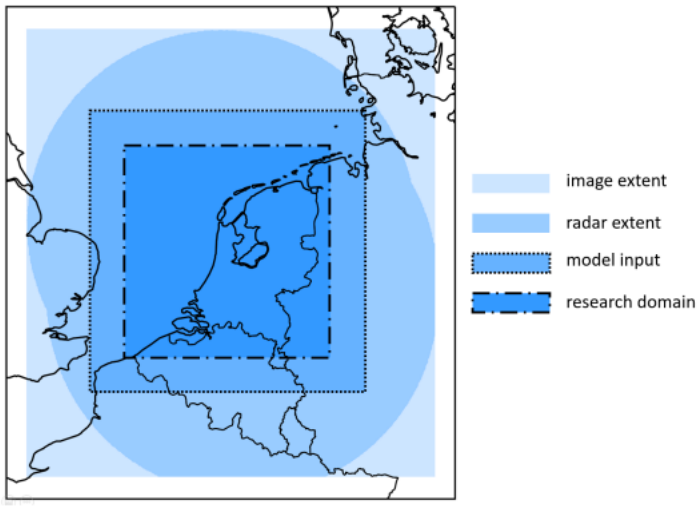


Figure 4.1: Image extent: the 700x765 array the data is provided in. Radar extent: the pixels containing radar measurements. Model input: 480x480 array. Research domain: 360x360 array. (source: Thesis van der Kooij (2021))

4.3. Event extraction

The dataset is first split into a training (2008-2018), validation (2019) and testing (2020) dataset. Since the focus is on heavy summer precipitation, it is desired to exclude events without any rain. However, the work of van der Kooij (2021) showed that it is best to use as much data as possible when training the model. Therefore, the same method is followed and the training events are selected as follows:

First all images are labelled with 'rain' if at least 1% of the area has 1mm/h of rain, or otherwise with 'no rain'. Any sequence of 25 consecutive images (5 historical and 20 future images) that does not contain any 'no-rain' labels is considered to be a 'valid event'. This results in 185.324 training events.

For the testing events, van der Kooij (2021) argued that we now do want to only focus on the heavy summer precipitation events and therefore these events were selected as follows:

Now the images are labelled with 'rain if at least 2% of the area has 1mm/h of rain. Afterwards, an event of 25 images is selected if it does not contain any 'no rain' labels and if 12 or more images contain a peak of 30 mm/h of rain. This results in 247 testing events spread out over 13 different days between June and August 2020.

These testing events are used to assess the performance of the final chosen models. However, during the procedure of developing these models, further analysis of certain events are needed to check for improvements in sharpness. This cannot be done on the testing event set, since then it would not be an independent verification dataset anymore. Therefore, this analysis is performed on four heavy summer precipitation events (see Table 4.1) in the training dataset. These events were randomly selected from 6264 training events that were labeled as heavy precipitation in the same way as was described for the testing dataset.

Note that event 2 is actually not a heavy precipitation event. The reason it got labeled as heavy

precipitation is that it contains quite some clutter, which is not filtered out by the threshold of 50 dBZ. This can be a problem during the training, which is further discussed in Section 6.1.

Event 1	2012-08-15 18:50:00
Event 2	2015-05-04 17:50:00
Event 3	2017-07-19 05:50:00
Event 4	2018-06-01 13:35:00

Table 4.1: The 4 selected events from the training data to analyse by eye during the model development.

4.4. Distribution of the data

As mentioned in Section 4.1, the data is provided as 700x765 pixel images with 8bit (0-255) pixel values that are linearly linked to radar reflectivity values. The benchmark model works with normalised pixel values (NPV). However, in the end we are interested in the RR [mm/h]. Since the NPV are related to the RR by a logarithm and a power law with exponent 1.6 (see Equation 4.1), the distribution of the dataset is different in the two domains, as shown in 4.2.

Note that in both domains, there is a big imbalance in how often the different values occur, causing more emphasis on the lower values. In the NPV domain, more than half of the pixels contain a value between 0 and 0.025.

A second important notion is that of how the RR are divided over the NPV. This is indicated with the vertical lines in Figure 4.2. It can be observed, that for the low values, an error of 0.476 in terms of NPV, actually represents an error of only 0.1 mm/h in the RR domain. At the same time, for the higher values, an error of predicting 20 mm/h instead of 30 mm/h is only represented by an error of 0.034 in the NPV domain.

Therefore, training on the NPV instead of on the RR causes a lot of extra emphasis on the low RR, instead of on the high RR. Section 5.1.2 presents how this effect of the difference in distributions is investigated. Section 5.1.1 further investigates the effects of the distribution on the application of the balanced loss that was proposed by X. Shi et al. (2017).

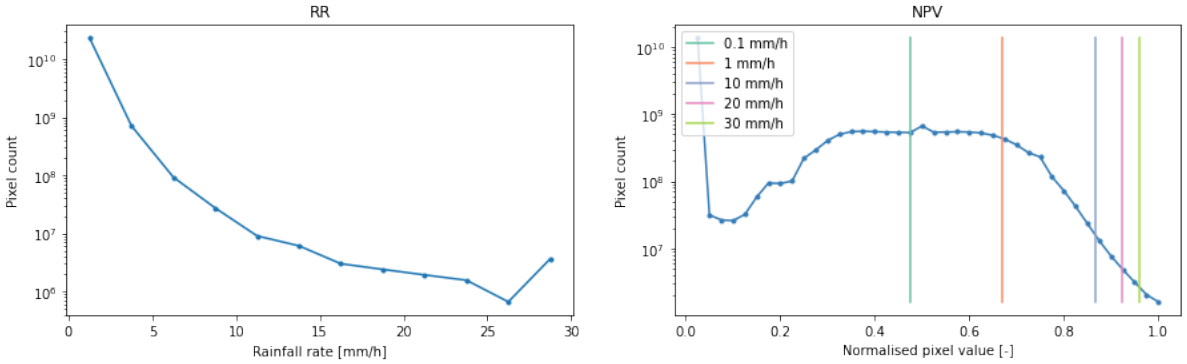


Figure 4.2: The distribution of the training data in both domains.

5

Methodology

The main aim of this research is to investigate the effect of different perceptual loss functions. However, after analysing the data distribution (see Section 4.4), further investigation on its effect on the benchmark is needed. This chapter first describes the methodology for investigating how the balanced loss affects the distribution and for analysing the difference between training in the NPV and RR domain.

Secondly, the methodology for answering the research question is introduced. The different loss functions are described in Section 5.2, followed by the method of verification (Section 5.4). The chapter concludes with an overview of all the different experiments (Section 5.5).

5.1. The effect of the data distribution on the benchmark

5.1.1. Effect of the balanced loss on the distribution

The idea behind the weights in the balanced loss, was to compensate for the positive skewedness of the distribution by assigning higher weights to the higher RR. This desired effect is depicted in Figure 5.1, and is obtained when the model is trained in the RR domain, as proposed in the previous section.

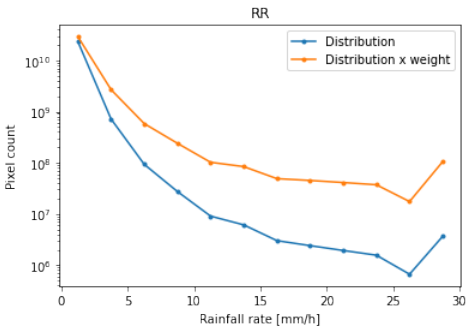


Figure 5.1: The effect of the weighting scheme from the balanced loss of the benchmark (see Equation 3.1) on the distribution of the RR in the training dataset.

However, when the model is trained on NPV, which has a different distribution than the RR, the weighting scheme needs to be adapted. The effect of the weighting scheme from the benchmark is shown in Figure 5.2. Most emphasis is put on the pixels between 1 and 10 mm/h and the first dot, representing the pixels between 0 and 0.025 mm/h, still dominates the distribution. Combining this resulting distribution with the fact that large errors for higher RR are represented by small errors in terms of NPV (see the vertical lines in Figure 5.2), it can be concluded that the balanced loss from the benchmark still emphasises the lower RR.

Theoretically, without compensating for the distribution, the higher RR should get a 1000x larger weight than the low RR, just to compensate for that the error in terms of NPV is a 1000x smaller for high RR (see Section 4.4). Including the distribution, this would mean that we want the weight difference between higher and lower RR to be even larger.

Several experiments are done to explore alternative options for the weights in order to emphasise larger RR and assess the impact of these weighting schemes on the performance of the model. These experiments are presented in the Appendix in Section A.2. They lead to the conclusion that even though the balanced loss from the benchmark model did not compensate for the imbalance in the distribution of the data as was expected, it does seem to be an optimum between fading and spreading out the rain. The emphasis on the zero pixel values is needed in order to predict dry areas correctly.

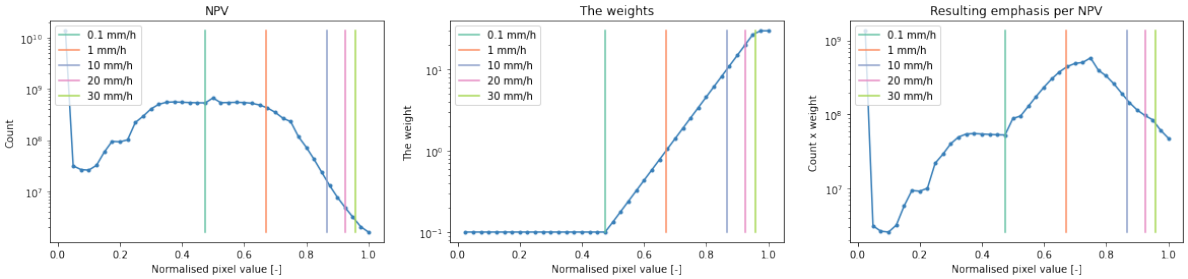


Figure 5.2: The distribution of NPV in the training dataset, the weights from the balanced loss from the benchmark (see Equation 3.1), and its effect on the NPV distribution.

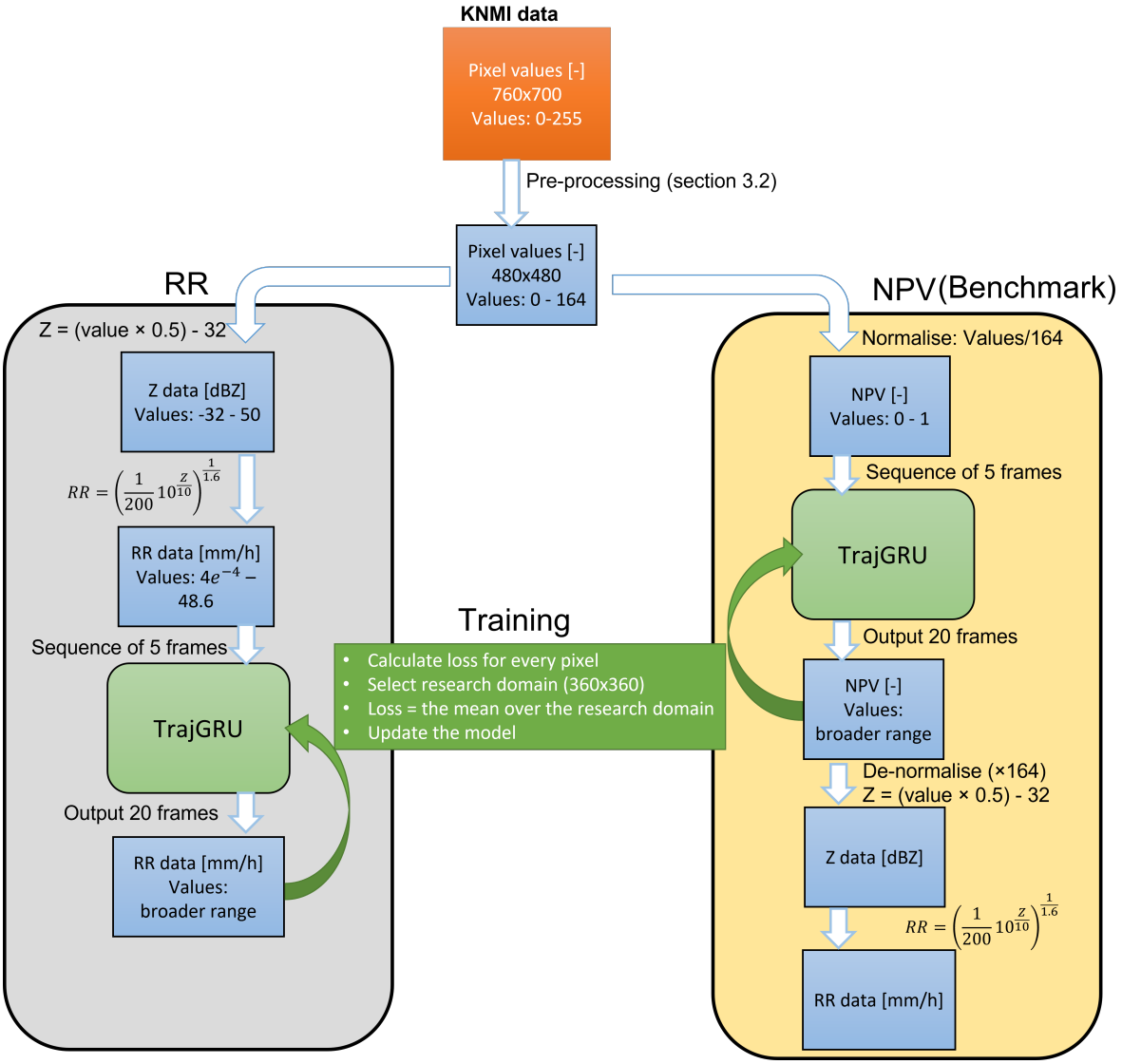


Figure 5.3: The two different options for the dataflow. It shows how the data is processed before and after it goes through the model. On the right it depicts the dataflow that was used by the benchmark. On the left it shows the newly proposed dataflow.

5.1.2. Training on rainfall rates vs normalised pixel values

As mentioned in Section 4.4, the distribution of the RR over the NPV causes an extra emphasis on the lower RR (see Figure 4.2). To avoid this problem, we propose to change the dataflow through the model and to train on the RR instead. The different dataflows are depicted in Figure 5.3. The benchmark takes the data in NPV, computes the losses and other metrics in terms of NPV and then gives the output in NPV, after which these are transformed to RR to show the final forecast. We now propose to change it such that the NPV are first transformed to RR before entering the model. The advantage of this approach would be that the error is given in mm/h, which is what we are interested in, and would therefore better represent the actual error.

To analyse the effect of changing the dataflow to the RR domain, the exact same model is used as in the benchmark, only that now it takes in RR instead of NPV.

5.2. Loss functions

The loss function plays a crucial role in determining the performance and statistical properties of deep learning models. Therefore, to answer the research question, a large number of experiments were preformed in which TrajGRU was retrained using 5 different types of perceptual loss functions. The SSIM, MS-SSIM, GDL, Wasserstein and FFL loss, which are presented in this section.

Structural Similarity (SSIM) loss

The SSIM looks at the differences in brightness, contrast and structure similarity between two images. It gives information about the intensity, grade and shape of the rainfall field. Unlike the MSE, the SSIM should reflect how we perceive the quality of an image (Z. Wang et al., 2003).

It does this by focusing on changes in local structure. A blurry prediction should get a worse SSIM value then a sharp prediction, since the local contrasts are different than in the observations.

The equation for the SSIM, for the forecast F and the observations O, is depicted in Equation 5.1. The $l(F,O)$ represents the brightness similarity, which is a term that should cause image distortions in very bright regions to be penalised less than distortions in less bright regions, since these are less perceived by the human eye. $c(F,O)$ is the contrast similarity, which causes distortions in regions with a lower contrast to be penalised more than in a higher contrast region. $s(F,O)$ is the structure similarity, representing the tendency of the forecast and observation to vary together. These terms are expressed in the means (the luminance) and standard deviations (the contrast) and its cross-correlation σ_{FO}. C_1, C_2 and C_3 ($C_3 = C_2/2$) are small positive constants to avoid division by zero and to obtain numerical stability.

$$\begin{aligned}\mathrm{SSIM}(F,O) =& l(F,O)\cdot c(F,O)\cdot s(F,O) = \left(\frac{2\mu_F\mu_O + C_1}{\mu_F^2+\mu_O^2+C_1}\right)\cdot\left(\frac{2\sigma_F\sigma_O + C_2}{\sigma_F^2+\sigma_O^2+C_2}\right)\cdot\left(\frac{\sigma_{FO}+C_3}{\sigma_F\sigma_O+C_3}\right) = \\ & \frac{(2\mu_F\mu_O + C_1)(2\sigma_{FO}+C_2)}{(\mu_F^2+\mu_O^2+C_1)(\sigma_F^2+\sigma_O^2+C_2)}\end{aligned} \tag{5.1}$$

The SSIM value is computed for every pixel in an image with a sliding window approach, using a Gaussian filter with size 11 and σ of 1.5. By using this sliding window approach, information about the local structure is obtained, which is the focus when we are interested in how we perceive an image (Tran & Song, 2019). To obtain one value for the whole image, the mean over all pixels in the research domain is taken, as can be seen in Equation 5.2.

The value of SSIM ranges between 0 and 1, with 1 indicating perfect similarity. Therefore the loss function for the SSIM can be expressed as in Equation 5.2. In the loss function, a score of 0 corresponds to perfect similarity, and 1 to no similarity at all. In this equation, the average is also taken over the total amount of predicted frames, which is 20 in our case.

$$\mathcal{L}_{\mathrm{SSIM}}(F,O) = 1 - \frac{1}{20\cdot 360\cdot 360}\sum_{n=1}^{20}\sum_{i=1}^{360}\sum_{j=1}^{360}\mathrm{SSIM}(F,O) \tag{5.2}$$

A limitation of the SSIM is that it only looks at one scale: it uses one size of Gaussian filter as moving window, which makes it dependent on the choice of the Gaussian filter. Therefore it only finds the similarity or quality of an image at that specific scale. Both Z. Wang et al. (2003) and Zhao et al.

(2016) argue and show that networks trained on the SSIM with different values of the filter σ for the Gaussian filter, results in different artifacts in the predictions. If it is chosen too small, edges are judged quite well on their sharpness, but it ignores splotchy artifacts in flat regions. However, when it is chosen too large, blurry and noisy edges are not penalised. Multiple values were considered for this thesis by testing their effect on the SSIM for different types/degrees of distortions. Since the effect was very similar, we continue with the values from the original paper.

Multi-Scale Structural Similarity (MS-SSIM) loss
Because of the disadvantage of the SSIM, Z. Wang et al. (2003) proposed the Multi-Scale SSIM (MS-SSIM), which combines multiple scales.

The original image is taken as scale 1, after which it is downsampled with a factor of 2 by applying a low-pass filter. This is done iteratively till the highest scale that is called scale M. The MS-SSIM is then calculated as in Equation 5.3. The contrast $c_j(F,O)$ and structure $s_j(F,O)$ comparison are done at every scale, while the luminance comparison $l_M(F,O)$ is only performed at scale M.

$$\text{MS-SSIM}(F,O) = l_M(F,O)^{\alpha_M} \cdot \prod_{i=1}^{M} c_i(F,O)^{\beta_j} \cdot s_i(F,O)^{\gamma_j} \tag{5.3}$$

The α_j, β_j and γ_j in this equation can be used to weight the relative importance of the three components, which were set by Z. Wang et al. (2003) to $\alpha_j = \beta_j = \gamma_j$, to simplify the parameter selection. Following Z. Wang et al. (2003), 5 different scales (4x down sampled with a factor of 2) are used and these weights are set to $([0.0448, 0.2856, 0.3001, 0.2363, 0.1333])$. Note that these weights do not sum up to 1. They were found by synthesising images with distortions at all 5 scales, and asking people to select one image of every scale of which they believed to have the same quality.

Similarly to the SSIM, the MS-SSIM obtains a value between 0 and 1, of which 1 represents perfect similarity. Therefore the MS-SSIM loss can be expressed as in equation 5.4, where the convection is changed again so that 0 represents perfect similarity and 1 no similarity.

$$\mathcal{L}_{\text{MS-SSIM}}(F,O) = 1 - \frac{1}{20 \cdot 360 \cdot 360} \sum_{n=1}^{20} \sum_{i=1}^{360} \sum_{j=1}^{360} \text{MS-SSIM}(F,O) \tag{5.4}$$

An advantage of the MS-SSIM is that it eliminates the problem of choosing the correct standard deviation of the Gaussian filter, since it looks at different scales and combines it. A disadvantage of MS-SSIM is that it it computational more expensive and therefore takes longer to compute (and to train when used during the training).

Gradient Difference Loss (GDL)
This loss penalises gradient differences between the prediction and the observations by only considering the neighbour pixel intensity differences (Mathieu et al., 2015). If a prediction is blurry and its edges fade out, the gradients between the pixels are a lot lower compared to the observations. Therefore the GDL has the potential to really sharpen an image.

The GDL as proposed by Mathieu et al. (2015) only considers the gradients between 2 pixels (see Equation 5.5), on which TrajGRU with our dataset was not able to train. Therefore, a slightly different definition of the GDL loss was used, which considers the gradient over 3 pixels (Equation 5.6).

$$\begin{aligned} &L_{GDL\_Mattieu}(F,O) = \\ &\sum_{i,j} \left\| O_{i,j} - O_{i-1,j}| - |F_{i,j} - F_{i-1,j} \right\|^{\alpha} + ||O_{i,j-1} - O_{i,j}| - |F_{i,j-1} - F_{i,j}\|^{\alpha} \end{aligned} \tag{5.5}$$

$$\begin{aligned} L_{GDL}(F,O) = \frac{1}{20 \cdot 360 \cdot 360} \sum_{n=1}^{20} \sum_{i=1}^{360} \sum_{j=1}^{360} &\left\| O_{n,i+1,j} - O_{n,i-1,j}| - |F_{n,i+1,j} - F_{n,i-1,j} \right\|^{\alpha} + \\ &||O_{n,i,j-1} - O_{n,i,j+1}| - |F_{n,i,j-1} - F_{n,i,j+1}\|^{\alpha} \end{aligned} \tag{5.6}$$

Again, the $O_{i,j}$ is the observation and $F_{i,j}$ the prediction for the $(i,j)^{th}$ pixel in the n^{th} image. The sum goes over all 360x360 pixels in the research domain, for all 20 predictions. The α in this equation is an integer of 1 or larger. Theoretically, the higher the α, the more similar the gradients should be, since it controls the rate at which the penalty increases/decreases. Different values were considered. However, because the results were very similar, only the case of $\alpha = 1$ is reported.

As mentioned by Mathieu et al. (2015), an advantage of the GDL is that it is relatively simple and therefore keeps the training time low, as opposed to other losses like the SSIM, that operate on a larger neighbourhood.

A possible disadvantage could be, that as long as the gradients are correct, the model receives a perfect GDL, regardless of a bias. Another problem is that it could start training on noise instead of the objects of interest, since noise has really high gradients and therefore may dominate the GDL.

Wasserstein loss

The SSIM, MS-SSIM and GDL all run into the problem that when a forecast is slightly shifted, they result in a large loss. So they are very sensitive for displacement errors. To prevent these large displacement errors the model might prefer smoother and more blurry predictions, even though the relation with the neighbours gets worse.

A solution for this is the Wasserstein loss / moving earth loss (Kantorovich, 1960). The idea behind this loss is that it sees every pixel value as a mass at a location and that it calculates how much it would cost to transfer these masses to the correct pixel to match the observation. Mathematically this is implemented as in Equation 5.7 by the python package GeomLoss (Feydy, 2020; Feydy et al., 2019).

$$W(O,F) = \min_{\gamma \in \mathbb{R}} \frac{1}{20 \cdot 360 \cdot 360} \sum_{n=1}^{20} \sum_{i=1}^{360} \sum_{j=1}^{360} \gamma_{n,i,j} \frac{1}{p} ||x_{n,i} - y_{n,j}||_p^p \tag{5.7}$$

In this loss function, $\gamma_{i,j}$ are the non-negative transport plans, whose rows sum up to O and columns to F. A transport plan specifies how the 'masses' should be moved. The parameter p determines the power to which the euclidean distance, between location x and y, is taken. Therefore, the higher the p, the higher the punishment for a wrong location.

To implement this loss function, the package GeomLoss is used, which uses Sinkhorn divergence to quicken up the computation of the Wasserstein distance, which is very computational expensive on its own. This makes it feasible for large number of samples, according to the recommendation of another python package POT (Flamary et al., 2021), that uses the sliced-wasserstein loss.

Focal Frequency Loss (FFL)

Another loss that is less location dependent than the SSIM, MS-SSIM and GDL, is the Focal Frequency Loss (FFL) as proposed by Jiang et al. (2021). They use it for image reconstruction and synthesis. Up till now it is not yet used for precipitation nowcasting.

Jiang et al. (2021) argue that many artifacts in images arise due to that the model misses certain frequencies. Therefore they propose to transfer the images to the frequency domain with the 2D discrete Fourier transform (Equation 5.8). Next, every pixel in the frequency domain is expressed as a vector by rewriting the exponential into a real and imaginary part $a + bi$. The frequency distance, between a single pixel in the frequency domain of the prediction and observation, is the (squared) Euclidean distance taken between those vectors, as shown in Equation 5.9. To stimulate the model to learn the frequencies that are missing, every pixel is weighted according to the distance between the prediction and observation. These weights are computed according to Equation 5.10, after which they are normalised to the range [0,1]. Therefore the weighting scheme changes continuously during the training and always emphasises the frequencies that are missing for that forecast. The complete FFL is the weighted average over all the pixels and is shown in Equation 5.11.

$$
\begin{aligned}
F(u,v) &= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\cdot e^{-i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \\
&= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\cdot\left(\cos\left(2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right)-i\sin\left(2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right)\right) \\
&= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\cdot(a+bi)
\end{aligned}
\tag{5.8}
$$

$$
d(F_O,F_F) = |F_O(u,v)-F_F(u,v)|^2 \tag{5.9}
$$

$$
w(u,v) = |F_O(u,v)-F_F(u,v)| \tag{5.10}
$$

$$
FFL(F_O,F_F) = \frac{1}{20\cdot M\cdot N}\sum_{n=i}^{20}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} w(n,u,v)|F_O(n,u,v)-F_F(n,u,v)|^2 \tag{5.11}
$$

In these equations are for an MxN image, which is 360x360 in our case. $f(x,y)$ denotes the pixel value at pixel (x,y) in the spatial domain. $F(u,v)$ represents the complex frequency value at coordinate (u,v) on the frequency spectrum. $F_O(n,u,v)$ represents a pixel at the n^{th} lead time at location (u,v) in the observation. Or in the forecast $F_F(n,u,v)$, or weights $w(n,u,v)$.

5.3. Combining loss functions

Loss functions as the GDL and FFL are never used on their own in literature, but they are used as an addition to the MSE or MAE (Jiang et al., 2021; Mathieu et al., 2015). But also for the SSIM and MS-SSIM it was shown that a model trained on a combination of them with the MSE or MAE performs the best (Hess & Boers, 2022; Tran & Song, 2019). Therefore, also the combination of these losses with the balanced loss are investigated.

There are different ways of combining loss functions, which is also referred to as task balancing. Vandenhende et al. (2021) gave an overview of different methods for multitask learning, and how they perform compared to each other. Different methods they mention are GradNorm (Chen et al., 2018), Uncertainty Loss Weighting (Kendall et al., 2018), Dynamic Weight Average (DWA) (Liu et al., 2020), Dynamic Task Prioritization (DTP) (Guo et al., 2018), using equal weights, or finding fixed weights by a grid search. They conclude that a simple grid search, in which you try out different weights for the losses, outperforms the existing techniques. However, such a grid search is very computational expensive. On top of that they found that multi-task learning does often not perform better than single-task learning, which is a similar finding as Gong et al. (2019), who compared different methods to each other.

All above mentioned methods are in use and different papers show different methods to perform the best. For this thesis is was chosen to try 2 different methods. First to analyse the performance when both losses are weighted equally, since this is a fairly simple option. Secondly, to use the Revised Uncertainty Loss Weighting (Auxiliary Weighting Loss (AWL)) from Liebel and Körner (2018). It was chosen to test this one, since the Uncertainty Loss Weighting method from Kendall et al. (2018) is cited the most compared to the other methods, and Liebel and Körner (2018) has shown to achieve an improvement upon this method.

AWL is very similar to the Uncertainty Loss Weighting method from Kendall et al. (2018). They propose to weight the different tasks by there task-dependent/homoscedastic uncertainty. This is uncertainty in the input data and is different per task and therefore captures the relative confidence between the tasks. They do this by maximising the log likelihood, which leads to their final expression shown in Equation 5.12.

$$\mathcal{L}_{Kendall}(W, \sigma_1, \sigma_2) = \frac{1}{2\sigma_1^2}\mathcal{L}_1(W) + \frac{1}{2\sigma_2^2}\mathcal{L}_2(W) + \log \sigma_1 \sigma_2 \tag{5.12}$$

In this equation, the $\mathcal{L}_{1,2}$ represents the loss function you are using. W represents the model parameters and $\sigma_{1,2}$ is the noise parameter belonging to that loss. This is a parameter that is initialised at 1 for all the losses, after which it gets altered during the training. Therefore it is not an actual measure of the noise in the input or output, but rather a representation of the confidence in the loss function. The $\log \sigma_1 \sigma_2$ is a regularisation term that prevents the noise to increase too much, meaning that it would effectively ignoring the data.

Liebel and Körner (2018) use the same principle, they only change the regularisation term to prevent negative loss values. This expression is shown in Equation 5.13.

$$\mathcal{L}_{AWL}(W, \sigma_1, \sigma_2) = \frac{1}{2\sigma_1^2}\mathcal{L}_1(W) + \frac{1}{2\sigma_2^2}\mathcal{L}_2(W) + \ln(1 + \sigma_1^2) + \ln(1 + \sigma_2^2) \tag{5.13}$$

Both combination methods were tested on the same model with the same loss function, to see if they behaved differently. The results are shown in Section 6.6.

5.4. Verification

As mentioned before, Eva's model is used as benchmark. However, there is no universally accepted verification metric to judge how realistic and sharp a forecast is. Therefore, subjective human judgement is used to get a first impression of the sharpness and realism of the predictions. This is done based on the 4 selected heavy summer precipitation events, mentioned in section 4.3.

To complement, several metrics are used. The realism and sharpness are quantified by analysing the metrics that are also used in the training: SSIM, MS-SSIM, GDL, Wasserstein and FFL loss. These are analysed in 2 ways. Firstly by plotting them over the lead time, to see whether models perform differently with increasing lead time. Secondly, these metrics are analysed per category of rain intensity, to be able to analyse what their effect is on the prediction of different intensities of rain (mm/h). The categories that are used are: [0, 0.1], (0.1, 0.5], (5, 1], (1, 5], (5, 10], (10, 20], (20, 30], > 30.

Next to analysing the sharpness, also the the balanced loss, MAE, MSE, CSI and Frequency Bias are analysed per rain intensity and per lead time, to check if the quality of the forecasts stays the same. These metrics are presented below.

However, note that every metric emphasises a different aspect in the prediction and this thesis simply uses a selection to get an impression of the performance. It needs to be kept in mind that the performance depends on what the end user considers to be important. Therefore, the analysis by eye and the analysis of the metrics can be used to define the pros and cons of different models, but there is no way to determine whether a method is objectively better than another.

Mean Absolute Error (MAE): Measures the magnitude of the absolute error. The lower the better.

$$\text{MAE} = \frac{1}{20 \cdot 360 \cdot 360} \sum_{n=1}^{20} \sum_{i=1}^{360} \sum_{j=1}^{360} |F_{n,i,j} - O_{n,i,j}| \tag{5.14}$$

Mean Squared Error (MSE): Measures the magnitude of the squared error. It punishes large errors to a greater extent than the MAE. The lower the better.

$$\text{MAE} = \frac{1}{20 \cdot 360 \cdot 360} \sum_{n=1}^{20} \sum_{i=1}^{360} \sum_{j=1}^{360} (F_{n,i,j} - O_{n,i,j})^2 \tag{5.15}$$

Critical Succes Index (CSI): This is a categorical metric (see Figure 5.4). It measures how well it corresponds that the category is predicted compared to that it is observed. It ranges from 0 to 1, where 1 indicates the perfect forecast.

$$\text{CSI} = \frac{\text{hits}}{\text{hits} + \text{misses} + \text{false alarms}} \tag{5.16}$$

Frequency bias: Also a categorical metric (see Figure 5.4). It measures the ratio between forecasted positives and observed positives. It ranges from 0 to infinity. A score >1 indicates overforecasting and <1 underforecasting. 1 is the perfect score: the number of forecasted positives is the same as the number of observed positives, but this does not say anything about whether these were correct.

$$\text{Frequency Bias} = \frac{\text{forecasted positives}}{\text{observed positives}} = \frac{\text{hits} + \text{false alarms}}{\text{hits} + \text{misses}} \tag{5.17}$$

Event forecast	Event observed		
	Yes	No	Total
Positive	True Positives (TP)	False Positives (FP)	Forecast positives
Negative	False Negatives (FN)	True Negatives (TN)	Forecast negatives
Total	Observed positives	Observed negatives	Total

Figure 5.4: The contingency table based on which the categorical metrics are computed. Depending on the threshold, the pixels are converted into binary maps. Positive if the pixel was above the threshold, otherwise negative. This table can be computed computed by comparing the binary maps of the forecast and the observation. (Source: Adjusted from Figure 3.9 in the Thesis of van der Kooij (2021).)

5.5. Experiments

In an attempt to answer the research question: "How can the loss function in TrajGRU be modified to get sharper and more realistic predictions, without worsening its performance?", multiple experiments with the above mentioned losses or combinations of them are performed. Table 5.1 gives the overview.

Firstly, some experiments are listed to investigate the effect of the architecture and the data distribution on the model. Secondly, all losses are used on their own and with multiple alterations, to investigate the complications they bring (see Table 5.1). These are presented and explained in Sections 6.2 - 6.4. There are no experiments conducted with the MS-SSIM on its own, since it suffers from the same issues as the SSIM.

Lastly, based on the issues that were found, combinations of the different losses are examined, as presented in Table 5.1. These models are compared to the benchmark and used to form the final conclusions. Those results can be found in Section 6.6.

All the models were trained with the same training/validation/testing data as was used for the benchmark (see Section 3.4). Also, unless specified differently, all the other settings were kept the same. The models were trained on a GPU: NVIDIA GeForce RTX 3090 24Gb GDDR6X or on 8 Tesla K80 GPUs at the VR-LAB from TU Delft (http://www.srderoode.nl/pubs/VRLAB-Manual.pdf). On the first device it took around 24 hours to train a model, while on the second device it took around 4-5 days. Models trained with the Wasserstein or MS-SSIM loss (4.8 s/it) took longer to train compared to the balanced, SSIM, GDL and FFL loss (1.11 s/it).

Guidelines on how to set up a similar environment at different computers, in order to make sure that the experiments are comparable, are presented in Appendix section A.5.

Name of model	**Explanation and Task**
Benchmark	Balanced loss, trained on NPV, with the table settings from van der Kooij (2021)
Balanced_SHI	Balanced loss, trained on NPV, with the table settings from X. Shi et al. (2017)
Benchmark_RR	Balanced loss, but trained on RR instead of NVP
Balanced_V (3/4/5/6/9/10/13)	Explained in Section 5.1.1
SSIM	Only the SSIM loss with ($k_1 = 0.01$ and $k_2 = 0.03$)
SSIM_high	Only the SSIM loss with ($k_1 = 0.1$ and $k_2 = 0.3$)
SSIM_low	Only the SSIM loss with ($k_1 = 0.00001$ and $k_2 = 0.00003$)
SSIM_RR	SSIM loss with $k_1 = 0.01$ and $k_2 = 0.03$, but on RR instead of NVP
SSIM_penalty	Explained in Section 6.2.3
SSIM_penalty_20_80	Explained in Section 6.2.3
GDL	only the GDL loss
MS-GDL	GDL computed at different scales, inspired by MS-SSIM
GDL_bias	GDL + mean bias over the wet pixels in the observations
GDL_max_bias	GDL + max bias over the wet pixels in the observations
Wass_NPV (p=2 and p=1)	Only the Wasserstein loss on NPV (GeomLoss)
Wass_RR (p=2 and p=1)	Only the Wasserstein loss on RR (GeomLoss)
FFL_NPV FFL_RR	Only the FFL loss on the NPV
FFL_RR	Only the FFL loss on the RR
SSIM + Balanced (eq)	Combination of the SSIM (RR) and Balanced loss (NPV), both weighted once
SSIM + Balanced (awl)	Combination of the SSIM (RR) and Balanced loss (NPV), weights determined by AWL
MS-SSIM + Balanced (awl)	Combination of the MS-SSIM (RR) and Balanced loss (NPV), weights determined by AWL
GDL (NPV and RR) + Balanced (eq)	Combination of the GDL (NPV or RR) and Balanced loss (NPV), with weights $\frac{1}{2}$ and 1 respectively for NPV, and 1 and 1 for RR

Table 5.1: An overview of all the different experiments conducted to analyse effect of the model architecture, data distribution and the influence of the different loss functions.

6

Results

This chapter presents, analyses and discusses the results of training with a different data format and/or a different loss function.

In the process of training the models, several refinements have been made to the code of TrajGRU that van der Kooij (2021) has been working with. For example to decrease the training time, all computations are done on the GPU instead of switching between GPU and CPU. These are all documented and traceable. More details can be found in the `Notice.txt` file in the code folder on GitHub: `https://github.com/Diewertje11/Precipitation_nowcasting_MScThesis_Diewertje`.

6.1. Training on rainfall rates vs normalised pixel values

As discussed in the methodology, the data-distribution in NPV (see Figure 4.2) causes the error for lower RR to be larger than for high RR, due to which the lower values get emphasised during the training. By training on the RR instead, the error is more representative of the error of interest, since it puts the correct emphasis on all RR. With this correction, the balanced loss fulfils its intended function of emphasising the higher RR.

This section only presents the results of training on RR vs NPV for the balanced loss. The effect can differ when other loss functions are used and these are investigated as well and presented with the results of these loss functions.

Figure 6.1 shows that the precipitation field gets spread out with increasing lead time, when the model is trained on RR with the balanced loss function. In this case, the high RR need to be predicted correctly both in size and in location, since the balanced loss is location specific. However, the location is hard to predict for the model and therefore it is safer to predict a medium amount of rain for a larger area, in order to limit the error size for pixels with high precipitation in the observation. At the same time, predicting such a medium amount of rain in an observed dry area does not get punished much, due to the low weights in the balanced loss. Therefore it can be said that the model prefers to overpredict low rainfall intensities, to make the error for the high rainfall intensity pixels smaller.

Even though the predictions are worse visually, the balanced loss for these predictions is better. During this analysis, the loss is calculated on the RR, since that is the domain of interest. This lower balanced loss can be explained by the over- and underprediction of the higher RR. The maximum predicted RR is very different for both models. In terms of NPV, the observations have a maximum of 1, which corresponds to 48.6 mm/h. Predicting event 4 with the benchmark gives a maximum NPV of 1.08, which represents a RR of 129 mm/h, causing a huge overprediction. This phenomena does not occur during training on the RR, and therefore the predictions stay a lot closer to the actual observed range of RR, causing the balanced loss to be lower. However, even though the balanced loss is better for the model trained on the RR, the prediction is not better, as can be seen by eye.

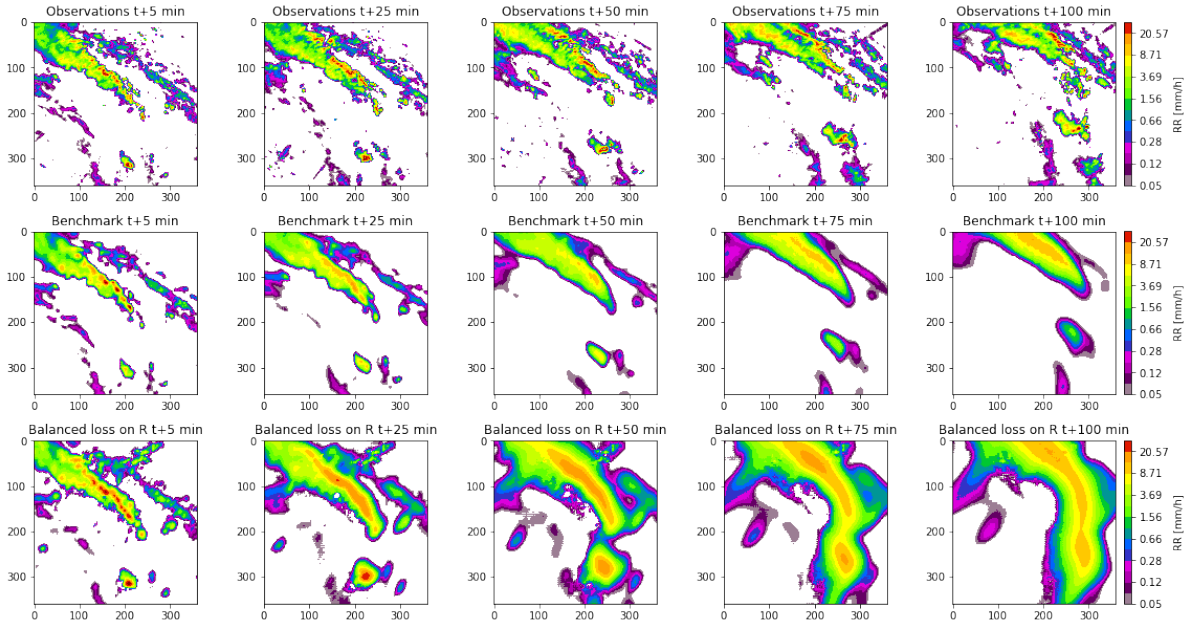


Figure 6.1: This figure displays the observations on the first row, the predictions of the benchmark on the second row and the predictions of a model with the balanced loss trained on the RR on the third row. All predictions are for event 4 and depicted in RR for lead times +5, +25, +50, +75 and +100 min.

A last important remark is the effect of clutter on the two different models. As depicted in Figure 6.2, the clutter removal during the data preprocessing, does not remove all the high value clutter. As a result, some sequences in the training data (e.g. Event 3) get classified as 'heavy precipitation' while they actually just contain clutter. The benchmark can handle this clutter quite well, while the model that is trained on the RR tries to predict this clutter. The reason is once again that the error for the high precipitation pixels is a lot larger when you train on the RR and therefore it also puts a lot emphasis on the clutter.

This leads to the conclusion that training on the RR with the balanced loss function could only work if more extensive clutter removal is performed.

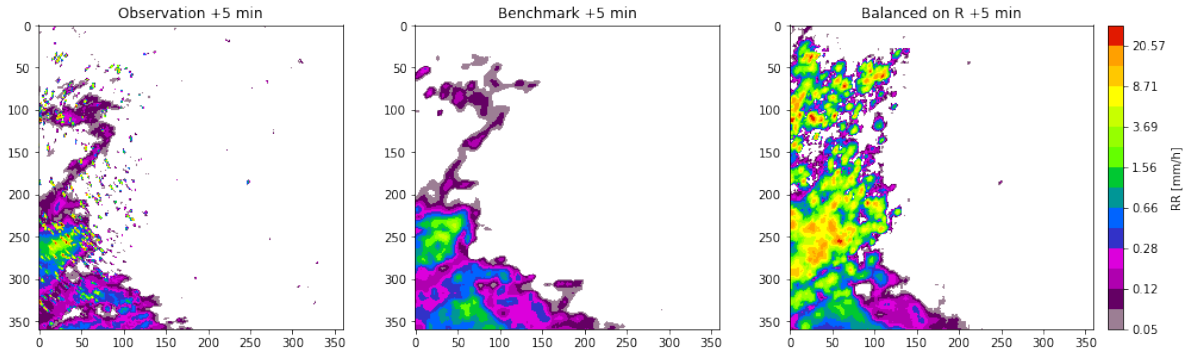


Figure 6.2: The observations in RR of Event 3, that contains a lot of clutter. And the effect of this clutter on the predictions of the benchmark model and the model trained on the RR.

Conclusion

Even though training on the RR would use a better representation of the actual error, these models tend to overpredict low rainfall intensities in order to obtain a smaller error for higher rainfall intensities. Therefore they spread out the rain. Moreover, these models start to train on the clutter when only simple clutter removal is applied. This leads to the conclusion that it is preferred to train on the NPV when using the balanced loss and simple clutter removal.

Discussion
The difference between training on the RR or NPV is not clearly stated in previous research. Han et al. (2021), Hess and Boers (2022), Li et al. (2021), X. Shi et al. (2017), Tran and Song (2019), Yan et al. (2021), and Yin et al. (2021) and Jing et al. (2019), all train on reflectivity or normalised reflectivity values in dBZ. They do not mention any effects of transforming them to rainfall rates, even though this should influence the functioning of their loss functions as shown in this section and in the next sections. So this does form a problem for practical applications and should be kept in mind for further research.

Ravuri et al. (2021) seem to train on the rainfall rates, but do not mention any reasons or any problems. That they do not run into problems might be due to that they use a different model architecture, a different loss, and they use data where the clutter is filtered out with clutter phase alignment (CPA) in combination with a dynamic clutter map.

6.2. Issues when applying the SSIM

The formula for the SSIM is repeated below, but now the expression for the constants '$C_{1,2}$' are filled in. As mentioned before, C_1 and C_2 are small positive constants, added to avoid division by zero and to obtain numerical stability. They can be expressed as $C_1 = (k_1 \cdot max\_value)^2$ and $C_2 = (k_2 \cdot max\_value)^2$, where the max_val is equal to 1 when working with NPV. Z. Wang et al. (2003) proposed to use $k_1 = 0.01$ and $k_3 = 0.03$. However, one major outcome of this research was the realization that the use of SSIM on NPV together with certain combinations of C1 and C2 can have unintended, negative effects on the rewards/penalties given to the model during the training.

$$\text{SSIM}(F,O) = \frac{(2\mu_F\mu_O + C_1)(2\sigma_{FO} + C_2)}{(\mu_F^2 + \mu_O^2 + C_1)(\sigma_F^2 + \sigma_O^2 + C_2)} = \frac{(2\mu_F\mu_O + k_1^2)}{(\mu_F^2 + \mu_O^2 + k_1^2)} \cdot \frac{(2\sigma_{FO} + k_2^2)}{(\sigma_F^2 + \sigma_O^2 + k_2^2)} \tag{6.1}$$

To illustrate these issues, we consider the RR and NPV corresponding to Event 1. The observation and prediction according to the benchmark are shown for +5 min in Figure 6.3, both in terms of RR and in terms of NPV. First, the effect of the constants on the SSIM score for an already decent forecast (according to the benchmark) are assessed in subsections 6.2.1. Secondly, the effect of the data distribution of the NPV on the SSIM score is investigated, for the case when we do not have a good prediction yet (6.2.2). This section gets concluded with the results of 2 methods proposed to deal with these issues, in Section 6.2.3.

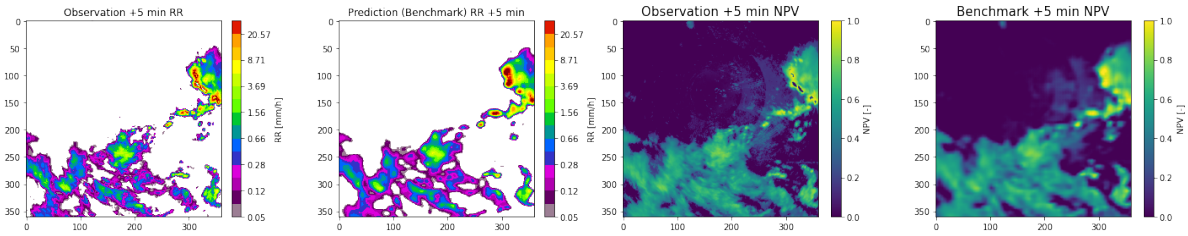


Figure 6.3: The observation and prediction for +5 min for Event 1. Both in RR and in NPV.

6.2.1. Constants causing a lack of penalty for low rainfall rates

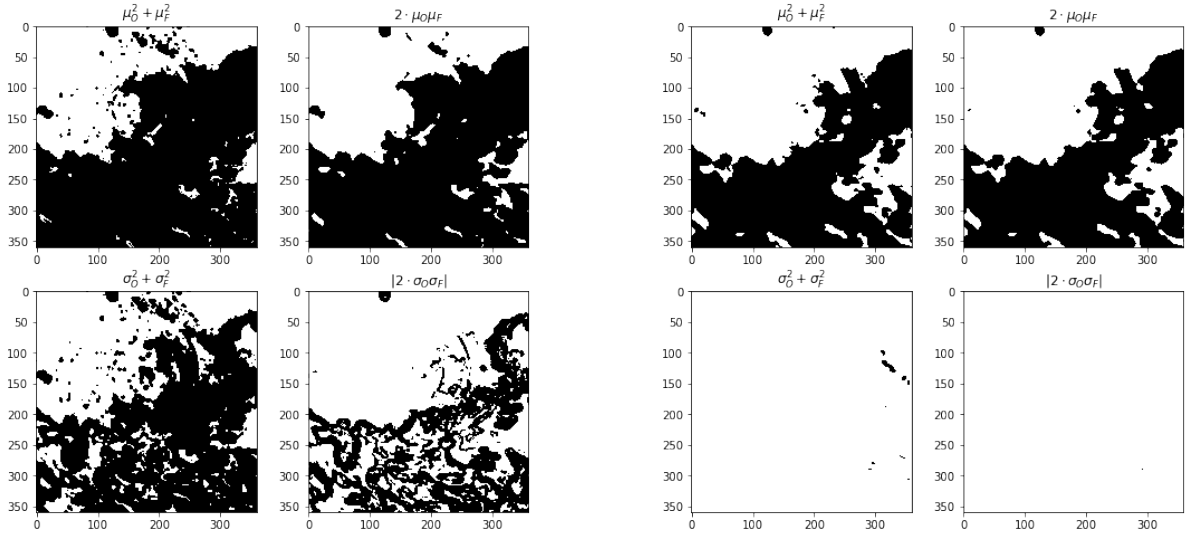
Large areas in the radar images contain patches with a homogeneous value of zero, for example the top left corner of Figure 6.3. In that case, the SSIM is calculated as shown in Equation 6.2, which shows clearly that the calculation gets dominated by the constants. This is not only the case for homogeneous zero valued pixels, but for all pixels with μ and σ values lower than k_1^2 or k_2^2 respectively. These pixels are indicated as white in Figure 6.4a for the case of the by Z. Wang et al. (2003) proposed $k_1 = 0.01$ and $k_3 = 0.03$. The black areas coincide with the precipitation areas, where the constants are irrelevant and the SSIM actually measures the similarity. For the white pixels, the constants dominate the calculation. This area always gets assigned a SSIM of 1, indicating perfect similarity and therefore ignoring small errors in low RR, which is desired (See the middle image in Figure 6.5). Therefore, the constants do not only avoid the division by zero, but also dominate the calculation of the SSIM for all pixels that are close to 0 (both in observation and forecast).

$$SSIM(F,O) = \frac{(2\mu_F\mu_O + k_1^2)}{(\mu_F^2 + \mu_O^2 + k_1^2)} \cdot \frac{(2\sigma_{FO} + k_2^2)}{(\sigma_F^2 + \sigma_O^2 + k_2^2)} = \frac{0 + k_1^2}{\mu_F^2 + 0 + k_1^2} \cdot \frac{0 + k_2^2}{\sigma_F^2 + 0^2 + k_2^2} \tag{6.2}$$

Smaller or larger constants

The left image in Figure 6.5 shows that for lower constants k_1 and k_2, the desired side-effect of ignoring small errors for low RR will not occur anymore, resulting in low SSIM values. Therefore a too low value of k_1 and k_2, puts more emphasis on predicting the zero and low valued pixels correctly during the training, which is not desired.

Increasing the constants results in a larger area where the constants dominate the calculation, as depicted in Figure 6.4b. For the μ values, the black areas (pixels where the constant does not dominate) coincide slightly better with the precipitating areas in Figure 6.3, and would therefore theoretically lead to a better representative SSIM. However, in this case the variance part of the SSIM always gets dominated by the constant, which would mean that it will only work with the means and will miss a lot of information about the structure in the precipitation field. This can be seen in the left image of Figure 6.5, where the SSIM is 1 for almost all pixels, while the structure of the cloud was not correct.



(a) $k_1 = 0.01$ and $k_2 = 0.03$ (b) $k_1 = 0.1$ and $k_2 = 0.3$

Figure 6.4: A binary map for the observation and prediction (with the benchmark) for +5 min for Event 1. White indicates that the depicted depicted value (shown in the title) is lower than k_1^2 or k_2^2, for the μ and σ values respectively.

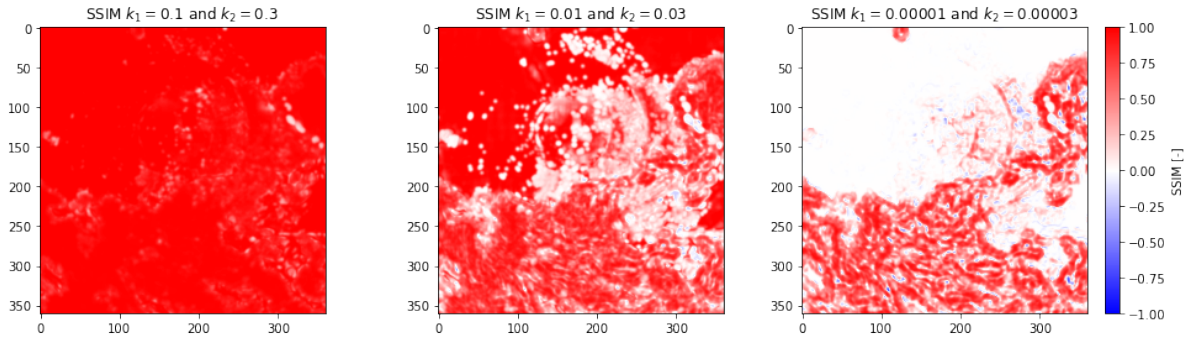


Figure 6.5: The SSIM for every pixel in the case of different values for the constants k_1 and k_2. The SSIM is here depicted as metric and not as loss, so 1 indicates perfect similarity and 0 no similarity.

Conclusion

From this section it can be concluded that it is important to understand the effect of different values of k_1 and k_2, to make sure that the SSIM behaves as intended, instead of getting corrupted by large rewards (too high constants) or penalties (too low constants) arising from the presence of dry areas within the domain. However, the next section shows that this effect becomes even more complicated due to the distribution of the data.

6.2.2. Effect of the data distribution on the SSIM

So far the effect of the constants in the SSIM, on an already decent prediction from the benchmark, were analysed. However, if without such a decent prediction, for example at the beginning of the training, we run into two other problems, caused by the distribution of the data in the NPV.

A first problem occurs when the model predicts homogeneously a little bit of rain (e.g. 0.05 mm/h) for an area without rain (for example the top left corner of Figure 6.3). In terms of NPV, this means that the σ_F is 0, but the μ_F is 0.42. This would mean that from Equation 6.2 we are left with: $SSIM = \frac{k_1^2}{0.42+k_1^2} \cdot \frac{k_2^2}{k_2^2}$, which results in a really low SSIM for the fact that the model only made a prediction error of 0.5 mm/h. Also, taking larger constants does not solve the problem. This effect is the largest when μ_O is equal to 0 exactly, which is true for 21% of the pixels in this event, and for even a larger percentage in other events. Additionally, it also does occur to a lesser extent for all low RR observations.

Therefore the conclusion can be drawn, that due to the distribution of the RR over the NVP (see Figure 4.2), using the SSIM on NPV harshly penalizes the models that fail to correctly predict the dry areas, even if the actual error they make in terms of RR is very low.

On top of that, a second problem occurs. If the model has learned those lower RR (=NPV between 0 and 0.42), due to the emphasis they get, these pixels will obtain an SSIM of 1. Since every image contains many (sometimes the majority) of these lower RR pixels, the reward is huge for predicting close to zero correctly. At the same time, the error for predicting the higher RR can obtain a maximum error of 1, since the SSIM is bound between 0 and 1. Therefore the model might prefer to predict zero for every pixel and fade out the rain. Thus the desired effect (Section 6.2.1) is no longer desired with this distribution.

This second problem is clearly visible for models trained with the SSIM, using different values for k_1 and k_2. The analysis of these models is performed in NPV, so that all the effects are visible (See Figure 6.6). It can be observed, that the colorbar changes with lead time, indicating that the rain fades out over time. The higher the constants, the more this occurs, since this second problem occurs for more pixels (observations up till 0.1 instead of 0.01). Also, the relevance of this problem increases with lead time, since the location of rain at larger lead times is harder to predict, making it saver for the model to simply predict zero.

Training with really small constants shows another effect (see Figure 6.6). The model seems to capture the correct spatial pattern, but it predicts all the values negative. It is suspected that this occurs because the model simply took a path to a different minima, since restarting the model multiple times, shows alternations between the prediction of negative or positive values.

Moreover, this model captures a lot more of the low valued precipitation and clutter, showing that the 'desired' effect (Section 6.2.1) does not occur for these pixels.

(Side-note: The blocky pattern is an indication that the training is not completed yet. However, since this model was not going in a sensible direction, there was no use in training even further.)

Conclusion

Due to the shape of the distribution and that the SSIM is bound between zero and one, a lot of emphasis is put on the lower rainfall regime, just like we saw occurring for the balanced loss, resulting in the dissipation of the rain with lead time. This emphasis cannot be corrected for by altering the values of k_1 and k_2.

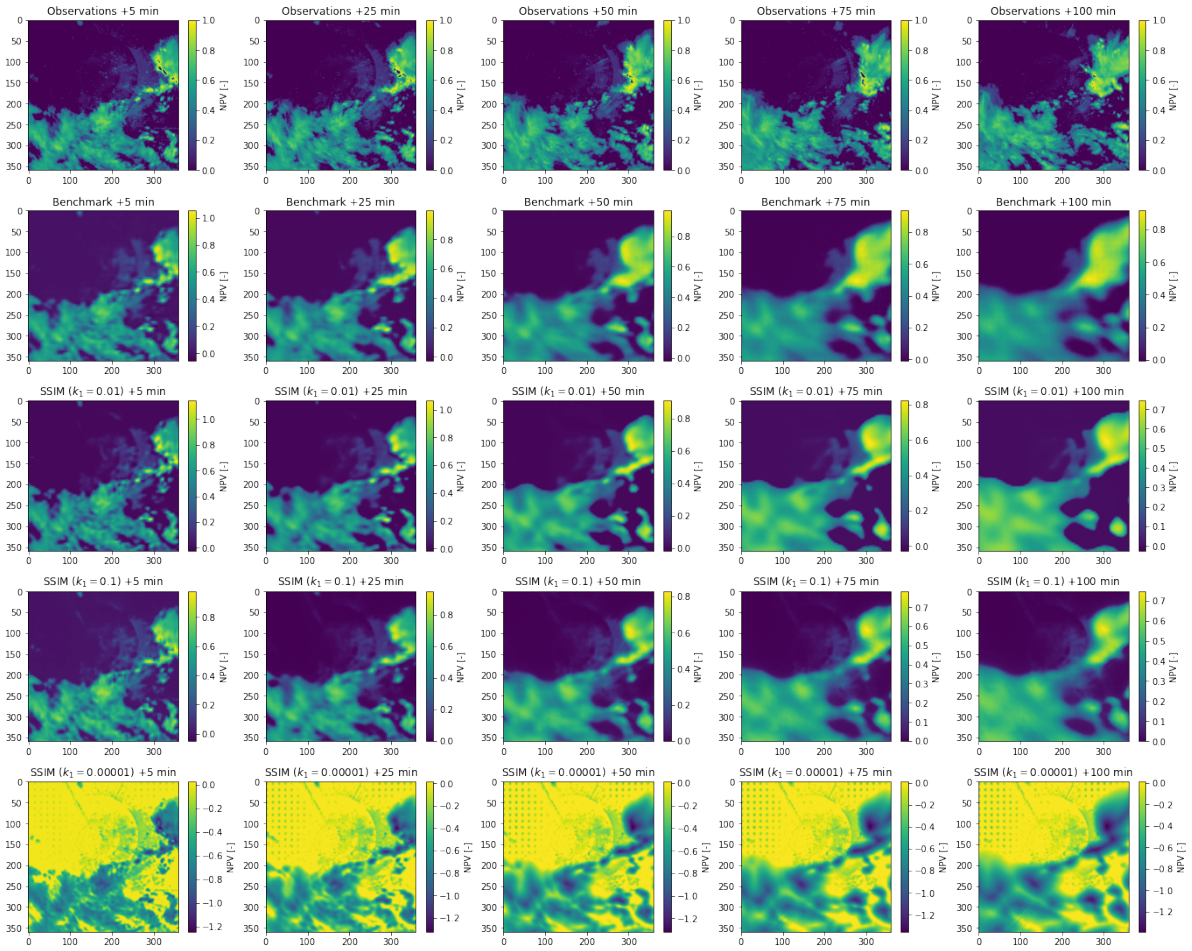


Figure 6.6: This figure shows the he observations in NPV for lead times +5, +25, +50, +75 and +100 min on the first row. The other rows show the predictions in NPV, using the benchmark or one of the 3 different versions of the SSIM with different values for the constants k_1 and k_2.

6.2.3. Two ways to compensate for the effect of the data distribution

Two approaches can be distinguished, to compensate for the effect of the data distribution on the SSIM: 1) Computing the SSIM in the RR domain instead, or 2) Masking out the pixels for which the issue occurs and giving them another penalty.

SSIM on the rainfall rates

By computing the SSIM on the RR, the low RR cover the correct amount of the total value range and therefore do not suffer from the first problem mentioned in section 6.2.2.

In this case, the $max\_value = 48$ instead of 1, and the expression for $C_1 = k_1^2 * 48$ and $C_2 = k_2^2 * 48$. Using this and making a similar binary map, Figure 6.7 shows that for a value of $k_1 = 0.01$ and $k_2 = 0.03$, the clouds of interest are cut out very clearly and all area around it will obtain an SSIM of 1, since the constants dominate there.

Thus, this solution could solve the first problem mentioned in section 6.2.2. Also emphasising the clutter, which occurred when training on the balanced loss in the RR domain, is not expected to occur with the SSIM, since it is bound between 0 and 1. However, instead it suffers more from the second problem mentioned in section 6.2.2. All NPV between 0-0.42 are close to 0 in the RR domain and get dominated by the constants (as shown in Figure 6.7). Therefore the second problem mentioned in section 6.2.2, causes that the model trained on the SSIM in the RR domain will almost completely dissipate the rain with lead time.

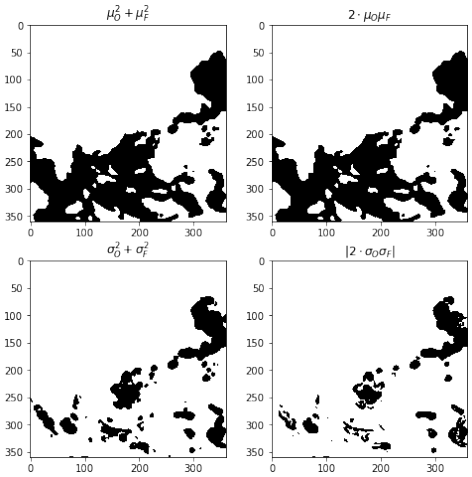


Figure 6.7: A binary map for the observation and prediction (with the benchmark) for +5 min for Event 1. White indicates that the depicted value (shown in the title) is lower than $k_1^2 * max\_value$ or $k_2^2 * max\_value$, for the μ and σ values respectively. This figure depicts the case where $k_1 = 0.01$, $k_2 = 0.03$ and the $max\_value = 48$. The μ and σ values are in terms of RR in mm/h.

Combine the SSIM with a penalty for no-rain

Another idea is to divide the pixels of the observation two groups: wet and dry. The SSIM only gets calculated for the wet pixels, for which the above mentioned problems do not occur. For the dry pixels, we propose to instead calculate a penalty that measures how many pixels are correctly classified as dry. This is important, because without such a penalty, the model would not learn to predict no-rain correctly.

This method is used in combination with training on the RR, since in that domain the σ of the pixels is representative for the actual spread. In the NVP domain, large RR differences are very close together and could therefore easily lead to overprediction.

All pixels that have a RR between 0 and 0.05 in the observations get sorted as dry. For these pixels a penalty is computed: the amount of pixels that is outside of this range in the forecast.

Before the SSIM is calculated for the wet pixels, all the dry pixels are set to zero in both the observation and the prediction, so that they do not influence the SSIM. The SSIM is calculated for every pixel and the SSIM for no-rain pixels is set back to 0. The constants for the SSIM are chosen very low ($k_1 = 0.0001$ and $k_2 = 0.0003$), to make sure that they do not dominate in any of the rain-pixels. The final loss is computed as stated in Equation 6.3.

$$Loss = (\#\text{ no-rain pixels} - \sum SSIM + \text{penalty})/(360 \cdot 360) \tag{6.3}$$

Using this expression for the loss, uses the ratio between the no-rain and rain pixels as the weights for the penalty and SSIM. This loss will be a value between 0 and 1, where 0 indicates a perfect prediction.

However, a model trained with this loss resulted in that the model predicted a small amount of rain at all dry locations. Therefore, a model was trained with 80% of the loss determined by the penalty and only 20% by the SSIM. The result of this model is depicted in Figure 6.8. Also in this case the penalty did not force the model to capture the dry areas.

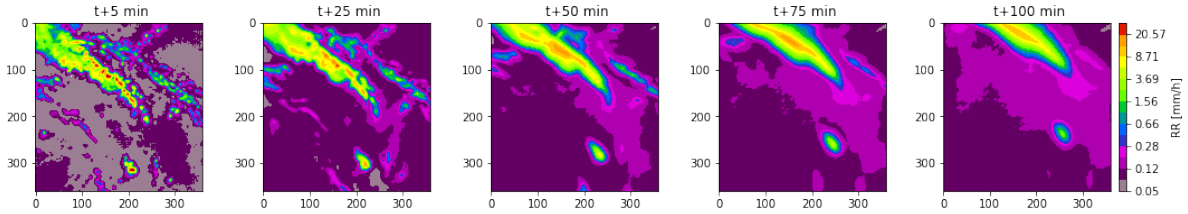


Figure 6.8: The forecast in RR for Event 4 with a model trained on the SSIM combined with a penalty, with weighting 20% and 80% respectively.

So apparently the model is not able to handle such a penalty, which is the same as for version 13 of the proposed balanced losses in Section A.2. Therefore it is suspected that the model does not like the combination of classification and regression, but the exact reason is yet unknown.

Conclusion
This leads to the conclusion that the SSIM cannot be used on its own for this dataset and model architecture, without suffering from any of the problems mentioned in Section 6.2. Therefore, the only option left is to combine it with another loss function, like the balanced loss. These results are presented together with other combinations of losses in Section 6.6.

Discussion
None of the papers Hess and Boers (2022), Li et al. (2021), Tran and Song (2019), Z. Wang et al. (2003), Yin et al. (2021), and Zhao et al. (2016), have discussed the influence of the of the constants in the SSIM, besides that they avoid zero division. Also, Li et al. (2021), Tran and Song (2019), and Yin et al. (2021), show to obtain slight improvements for precipitation nowcasting, when training the model on a combination of the MAE or MSE with the SSIM or MS-SSIM. This without addressing the emphasis the SSIM puts on the zero pixels and therefore contradicting the findings obtained in this research.

Bakkay et al. (2022) published a paper 4 months ago, also presenting the problem of the amount of zero pixels in precipitation data for the SSIM. They propose to solve this problem by weighting the SSIM for every Gaussian window by the standard deviation in that window in the observation, thereby creating the Weighted SSIM (WSSIM). Moreover they show that a Weighted MSE spreads out the rain more than a WSSIM, but that the WSSIM does not obtain the intensity correct. Therefore they argue that a combination is necessary. Thereby presenting similar results to what we have obtained.

6.3. Models trained on the GDL

Before the results of the models trained with the GDL are shown, the difference between training on the GDL with NPV and RR is shown in Figure 6.9. Training in the NPV domain causes more emphasis on the edges of the precipitation fields and low valued clutter, since the change from 0 to 0.1 mm/h of rain is covered by 0-0.42 NPV, creating the largest gradients. Training on the RR instead would ignore these edges, since the gradient from 0-0.1 mm/h is very small. Instead it would focus on details within the precipitation field, from low to high precipitation. Therefore, ideally, to get the details in both the edges and within the precipitation fields, a combination of both domains should be used.

However, it is also expected that training on RR with the GDL will run into the same problem as occurred for the balanced loss. It will start training on the clutter, since it is also an unbound metric. Therefore it is chosen to do the first experiments of the GDL in the NPV domain.

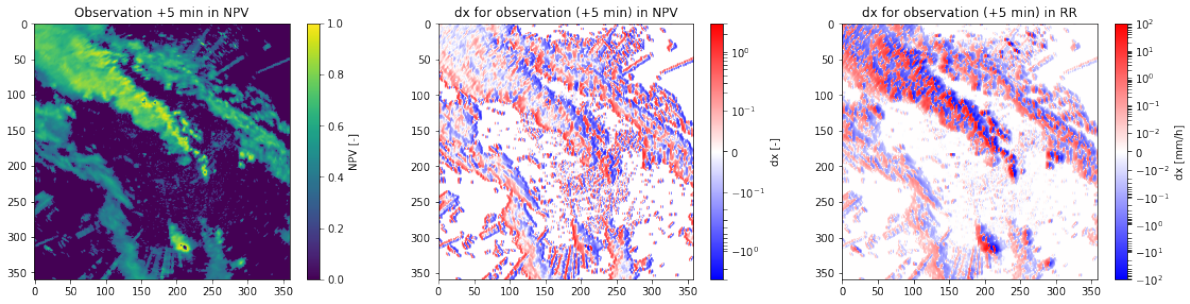


Figure 6.9: The observations for +5 min for Event 4, with its gradients in the x direction, both in the NPV and RR domain.

Figure 6.10 shows the results of training the model with the GDL loss, as proposed in Equation 5.6. The prediction for short lead times looks much sharper than the benchmark. However, the colorbar shows that the prediction is biased for all lead times and that this bias increases with lead time. This is caused by the large homogeneous zero patches in the images. These patches have a gradient of 0, causing that the prediction of lower gradients is safer than trying to predict the high gradients, especially at longer lead times.

It is also visible that the GDL does not change the spatial structure of the rain area over time, it seems to keep the same structure, just more blurry. This could be explained by that the penalty for predicting a larger gradient at the wrong location is big.

A first idea to deal with the bias was to apply a Multi Scale GDL (MS-GDL), where the GDL is calculated at different scales, similar to the MS-SSIM. This would include more global gradients. However, the predictions from the MS-GDL still contained the bias that increases with lead time (just slightly smaller). On top of that, the predictions are slightly less sharp than the GDL, since the finest scale that captures the details is weighted less.

As another approach to prevent this bias from occurring, the GDL was combined with a bias term. It was chosen to use a bias term instead of a combination with e.g. MAE, MSE or Balanced loss, since combining 2 location specific scores often results in one dominating the other. The absolute bias term is computed as stated in Equation 6.4. It is only computed over the wet area, by masking out all the pixels that are dry in the observation. This is done so that the average bias of the image is not decreased too much by the large dry areas. The bias term for all the 20 lead times is averaged.

$$Bias = \left| 1 - \frac{\sum_{\text{wet pixels}} F}{\sum_{\text{wet pixels}} O} \right| \tag{6.4}$$

The result is visible in Figure 6.10 as GDL bias rain. Instead of keeping up the pixel values at longer lead times, the model kept the average bias term of the predictions equal to that of the observations by overpredicting the first lead times. It can also be observed that all the dry pixels now contain rain too. The model still tries to decrease the gradients with lead time, to play it safe, but due to the bias term it cannot lower the wet pixels and therefore it now highers the dry pixels instead.

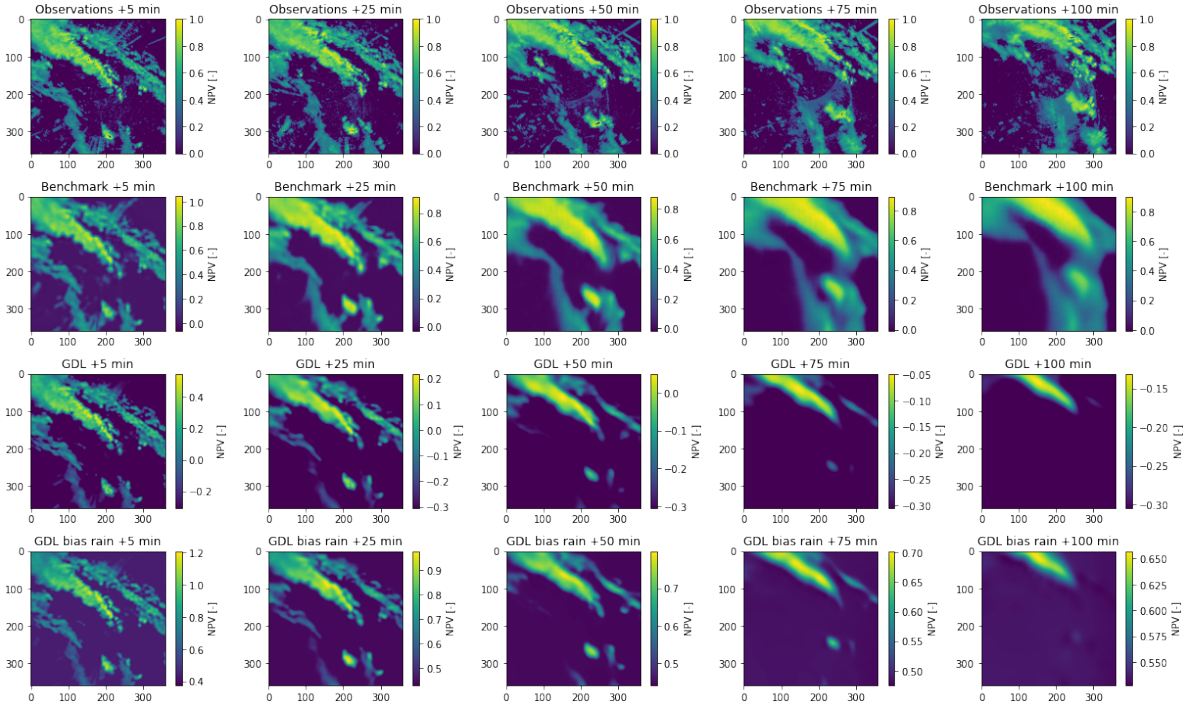


Figure 6.10: The first row shows the observations in NPV for lead times +5, +25, +50, +75 and +100 min for Event 4. The other rows show the predictions in NPV for the benchmark and the models trained on two different versions of the GDL loss. GDL: just the GDL loss. GDL bias rain: The GDL combined with the average over the 20 lead times of the bias over the rain area in the observations.

To prevent the model from overpredicting the first lead times and underpredicting the last, it was tested to use the maximum bias over the 20 lead times instead. However, this gave worse predictions for all lead times.

Conclusion
Due to the large homogeneous zero valued patches in the images, the models trained on the GDL loss tends to predict lower gradients at longer lead times, creating a bias and fading out the rain. Correcting for this with a location-invariant bias term did not show the desired results. Therefore, even though the GDL loss does show sharper predictions for the first lead times, it cannot be used on its own in the tested set ups. In Section 6.6, the result for combining it with the balanced loss is shown.

Discussion
The GDL loss was mostly used in video prediction or image enhancement (Liu et al., 2020; Mathieu et al., 2015; Xu et al., 2018; Zhao & Wildes, 2021). In those cases you do not have equally much uncertainty about the location in the next time frame, or an equally big imbalance in the data, causing them to not face the problems we have observed. However, Yan et al. (2021) have used the GDL in the context of precipitation nowcasting and did obtain improvements of 0.007 on the CSI and HSS. They did combine the GDL with a weighted pixel loss, probably explaining why they did not observe the extreme fading out.

6.4. Model trained on the Wasserstein loss

The result of a model trained with the Wasserstein loss on the NPV with 2 different settings for the parameter p, is depicted in Figure 6.11. It can be seen that these models fade out the rain more than the benchmark. This is caused by the way the RR are distributed over the NPV. The sums of all pixels in the research domain (the total 'mass') for lead time +100 min are displayed in Table 6.1, both in terms of NPV and RR. This clearly shows that in terms of NPV, the total sum for the Wasserstein models are closer to the observation than the benchmark. However, when this comparison is made in RR, the Wasserstein models perform much more poorly.

This analysis ignores the location where the 'masses' are placed, but does show the influence on the total mass when transferred from NPV to RR. It shows that the distribution of the dataset in the NPV domain prevents the Wasserstein loss to make a decent prediction.

Furthermore, it can also be observed that the predictions contain stripes, especially at longer lead times. They might occur due to that the Wasserstein loss values the location of the masses less and therefore allows the model to predict lines with higher values next to lower valued lines, so that together they are correct. This argumentation seems to fit with that it occurs more for p=1 than p=2, since p=2 squares the euclidean distance and therefore punishes an incorrect location more.

	Observation	Benchmark	Wasserstein (p=2)	Wasserstein (p=1)
$\sum$ pixels NPV	29672	32747	**28523**	23814
$\sum$ pixels RR	45828	**47448**	9015	7538

Table 6.1: The sum of the pixels over all the pixels in the research domain of Event 4 for lead time +100 min. This sum is presented for the benchmark and both Wasserstein models, both in terms of NPV and RR.

Since the Wasserstein loss does not perform well on the NPV, it was tried to train the model on the data in RR. However, with current settings for the LR and batchsize, the model was not able to train, meaning that the loss did not go down over the training iterations. Using a higher batchsize and/or a higher or lower learning rate, did result in that the model was able to train. But as can be observed in Figure 6.11, the resulting predictions do not agree well with the observations. And when all the pixels are summed together, the total RR is not close to the observations.

It can also be observed that there is a blocky pattern present in the predictions. This normally indicates that model was not done with training. However, the loss did not reduce anymore for the last few thousand training iterations, so this might indicate that it got stuck in a local minima.

An idea to prevent this is to use an already trained model (for example with the balanced loss) as the starting point for training with the Wasserstein loss. This could be further investigated.

However, it should be kept in mind that the used code is not actually computing the Wasserstein loss itself, but approximates it by using the Sinkhorn divergence. This method is not thoroughly investigated due to a lack of time. It might be that it runs into mathematical problems due to the zeros in the dataset, as occurred for the SSIM.

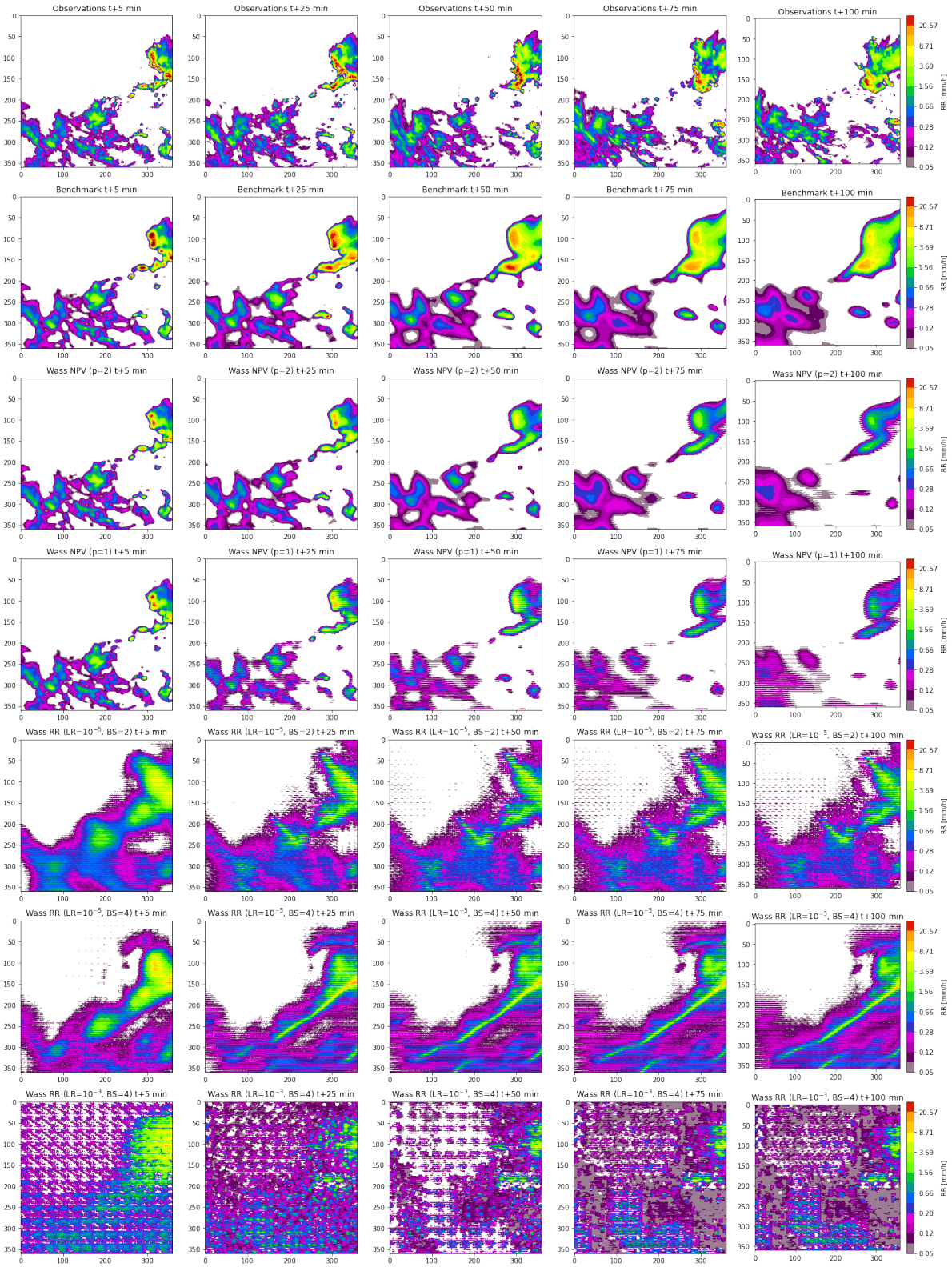


Figure 6.11: The first row shows the observations in RR for lead times +5, +25, +50, +75 and +100 min for Event 1. The 2nd, 3th, 4th row show the predictions in RR for the benchmark and the two models trained on the NPV with the Wasserstein loss, using p=1 or p=2, respectively. The last three rows show predictions of the models trained on the RR with the Wasserstein loss, while varying the batchsize and learning rate.

Conclusion
When the Wasserstein loss is used while training on the NPV, it fades out the rain a lot due to the conversion between NPV and RR. However, training on the RR has not been successful yet and could be further investigated.

Discussion
In general, the Wasserstein loss can mostly be found in literature in the combination with a GAN. Luo et al. (2022) show that their GAN with Wasserstein loss instead of the Jensen-Shannon divergence improves their nowcasting predictions. And that their multi scale Wasserstein GAN can improve them even further. (I could not find in their code which approximation of the Wasserstein loss they use, only that they do include the GDL as well). These results are the same as presented by Lee et al. (2020), who have explored different GAN set ups for radar data augmentation. Both papers work with normalised reflectivity values, in which they also report their results. Therefore they not mention the implications we have experienced. This does form a problem for practical applications, as this research pointed out, which should be kept in mind for further research.

6.5. Models trained with the FFL

The result of training on the FFL is shown in the NPV domain in Figure 6.12. This loss does not spread rain out with lead time, which was observed with the other losses. But the colorbar indicates that it still fades out the rain.

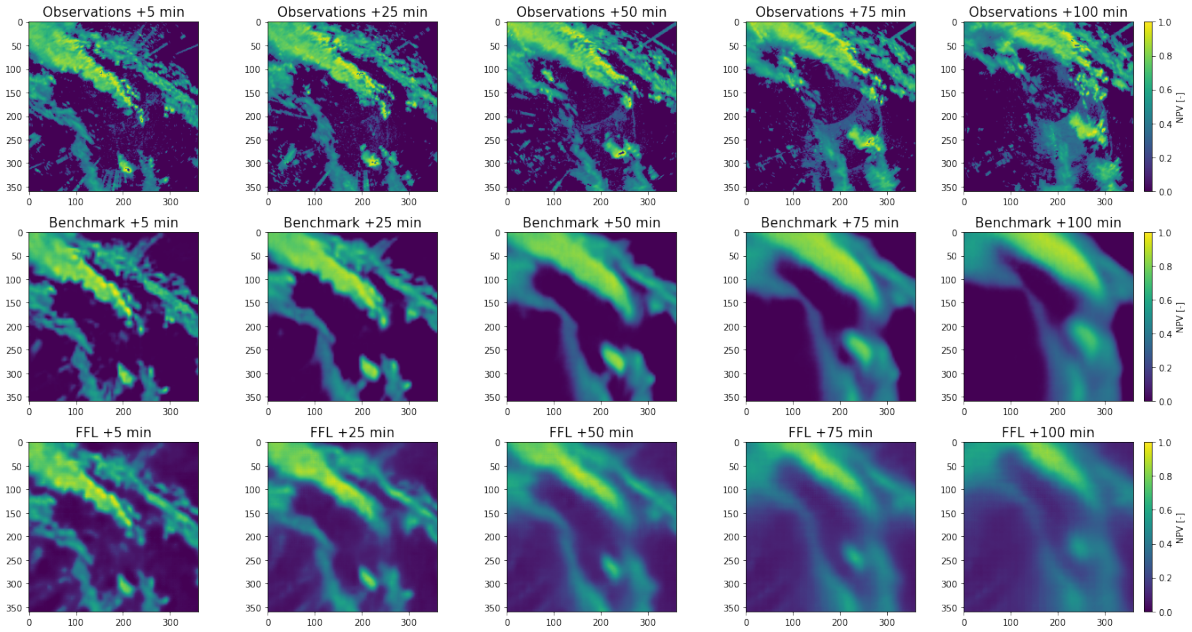


Figure 6.12: The first row shows the observations for lead times +5, +25, +50, +75 and +100 min for Event 4 in NPV. The other rows show the predictions in NPV for the benchmark and the model trained with the FFL.

The FFL loss compares the frequency spectra of the images to each other. Every pixel in the frequency spectrum (see Figure 6.13) has a coordinate (u,v), representing the contribution of the sine wave with frequency u in the x direction and frequency v in the y direction. The brightness of every pixel represents the squared amplitude of the sine wave, which is linked to the RR. The center point is the (0,0) wave, which brightness represents the average RR of the image. The FFL loss compares these spectra with each other.

Figure 6.13 shows that when the rain would get spread out, the frequencies that are present in the image change, leading to a larger FFL (see Table 6.2). When the rain would be predicted at the correct location but only lowered in value, the frequency that are present are the same, just with a lower amplitude. This leads to a smaller FFL compared to when the rain is spread out. Table 6.2 shows that the FFL is the lowest when the model does not spread out the rain, but simply lowers the values, both in case of a high RR event or no rain.

The last image in Figure 6.13 shows that the frequency spectrum becomes exactly the same, when

the same shape is kept but it is just slightly shifted. However, the frequency spectrum is the absolute value of the Fourier transform. The FFL loss compares the Fourier transforms itself and therefore still punishes this shift. An idea for further investigation could be to compute the FFL with the absolute Fourier transform, to make the loss completely location-invariant.

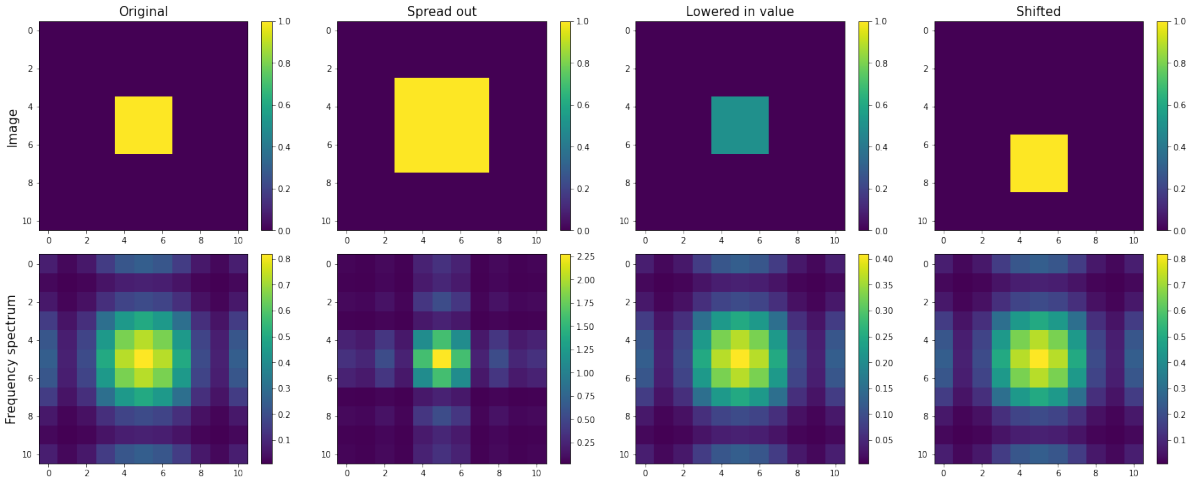


Figure 6.13: Examples of how the frequency spectrum changes as a result of a change in the original image.

	Original	Spread out	Lower value	Shifted	Spread out and lower
FFL wrt Original	0.0000	0.0640	0.0122	0.0707	0.0397
FFL wrt Empty	0.0488	0.1284	0.0122	0.0488	0.0321

Table 6.2: The FFL for the example cases in Figure 6.13. Both with respect to the original image and an empty image (only zeros). There is also a last column added for a case where the original shape would be spread out and lowered in value, showing that this still gives a higher loss than only lowering the value.

Analysing the prediction for the first lead time visually, shows that the sharpness of the prediction did not improve compared to the balanced loss. The FFL loss makes the model train on the frequency that contains the largest error. Lower frequencies have a larger amplitude (Figure 6.13 shows that the middle pixels are brighter), and therefore the absolute error in these frequencies is larger when the RR is decreased (compare the first and the third figure in Figure 6.13). Since the FFL is taken as an average over all the lead times, the error in the lower frequencies dominates, causing the model to train on them and still miss out on the details. In order to solve this problem, a different weighting should be taken per lead time.

Currently this model was trained on the NPV. Training on the RR would theoretically result in the same frequencies, but they will have a different amplitude. This might cause more emphasis on the details in the higher RR. However, training a model with the FFL on the RR for 100.000 iterations, results in a prediction with blocky patterns, indicating unfinished training, even though the loss over iterations had stabilised. So similarly to the model trained with the Wasserstein loss on the RR, some complications occur. Further research would be needed to come to a final conclusion for this loss in combination with the RR.

Conclusion
The FFL loss on the NPV did not lead to any improvements in sharpness for the first lead time, since the errors at the longer lead times dominate the loss. Furthermore, it was observed that the FFL does fade out the rain, but does not spread it out, which is different from the other losses. An interesting experiment could be to try a completely location-invariant FFL loss.

Discussion
As far as I am aware off, the FFL has not been used yet in precipitation nowcasting and therefore these results cannot be compared with other research.

6.6. Combination of losses

As concluded in the previous sections, it is not possible to train a model on only the SSIM, MS-SSIM or GDL. Therefore it was tried to combine them with the balanced loss. In these combinations, the SSIM and MS-SSIM are computed on the RR for only the wet pixels. The GDL is computed over the whole image on the NPV or RR. Since they gave quite similar results, only the GDL on NPV is shown in this section.

Note that the fact that they gave similar results might indicate that for both GDL combinations the balanced loss dominates. The dominance of one of the losses in a combined loss should not occur when the losses are weighted properly. However, as discussed in Section 5.3, combining them is not straight forward. Two methods were used in this research and both were applied to the SSIM loss. Both models turned out very similar, as can be seen in the results below. The model performed slightly better when it was combined with fixed weights, compared to using the AWL scheme. Therefore, the other combinations were made by weighting the losses equally much (except for the MS-SSIM), since there was a lack of time to try both methods for all loss functions.

The combination with the balanced loss was also tested for the Wasserstein and the FFL, but these models did not become better than the ones trained on a single loss, and therefore they are not reported here. This might be due to improper weighting of the losses, or due to that the issues these losses were facing cannot be counteracted by the balanced loss.

Figure 6.14 shows the different metrics over the lead time for the models with different loss combinations. The benchmark gets outperformed on all metrics, except for the balanced loss, the SSIM and the MS-SSIM. However, analysing the scores per RR and visually per event, shows that these better scores are actually not desired.

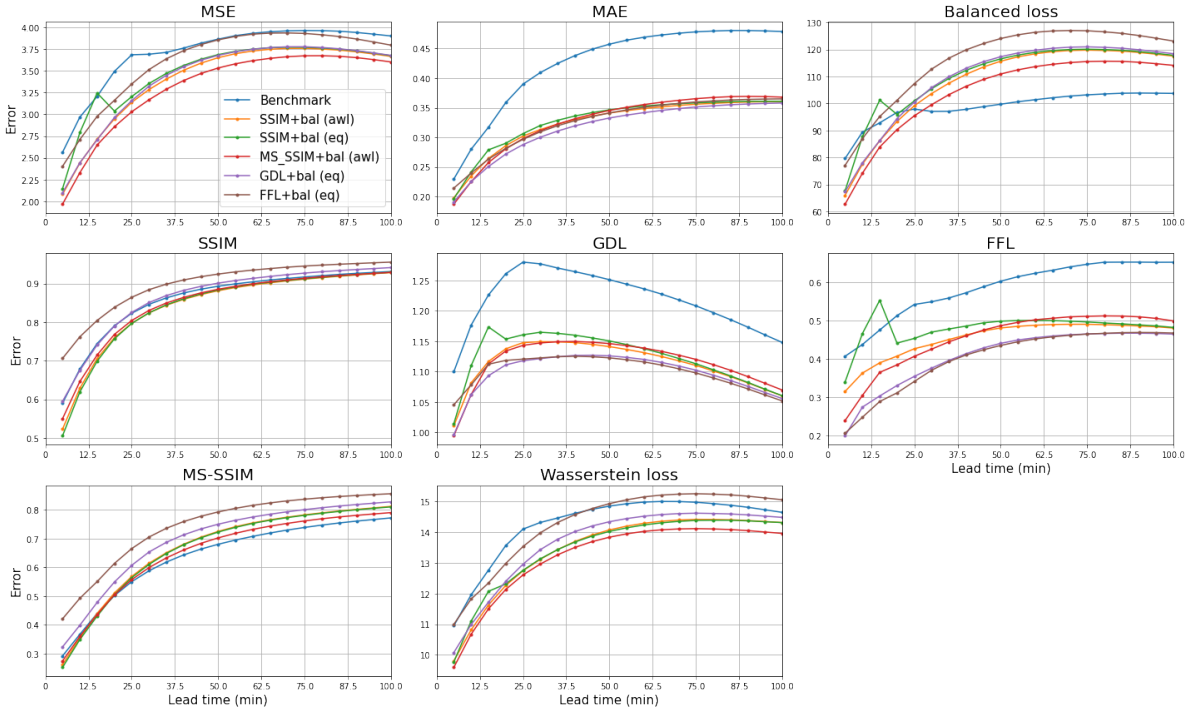


Figure 6.14: The different metrics over the lead times for the different models. They are computed on the testing dataset in RR. A lower value indicates a better score for all metrics (the SSIM and MS-SSIM are depicted as loss, so zero indicates perfect similarity).

The different metrics per threshold of RR, show that the balanced loss (see Table 6.3) for RR up till 5 mm/h is quite a lot worse for the benchmark compared to the others, but above 5 mm/h the benchmark performs better. This same pattern is present in the MSE and MAE. The SSIM is quite comparable for lower RR intensities, but above 5 mm/h the benchmark is slightly better. The GDL does not show much difference between the models for all thresholds. The CSI is comparable for lower RR, but the

benchmark has a better score for >5 mm/h (see Table 6.4). The frequency bias however, is better for the benchmark on all RR (see Table 6.5).

Balanced	[0, 0.1]	(0.1, 0.5]	(0.5, 1]	(1, 5]	(5, 10]	(10, 20]	(20, 30]	> 30
Benchmark	0.43	6.61	79.28	401.34	**1017.07**	**3608.07**	**11667.05**	**32761.06**
SSIM+bal (awl)	0.04	0.96	18.42	177.11	1155.14	5019.87	15647.84	40399.86
SSIM+bal (eq w)	0.04	1.03	20.38	197.23	1212.37	5069.35	15609.17	40214.77
MS-SSIM+bal (awl)	0.06	1.32	21.64	168.55	1049.05	4761.21	15230.59	39815.56
GDL+bal (eq w)	**0.03**	**0.72**	**13.41**	**155.52**	1197.47	5213.51	16140.52	41283.34

Table 6.3: The balanced loss per RR for the final models. The losses are computed on the testing dataset in RR. The best (lowest) value per RR is depicted in bold.

CSI	> 0.1	> 0.5	> 1	> 5	> 10	> 20	> 30
Benchmark	0.573	**0.496**	**0.429**	**0.196**	**0.117**	**0.037**	**0.012**
SSIM+bal (awl)	0.574	0.456	0.379	0.126	0.057	0.016	0.005
SSIM+bal (eq w)	0.574	0.460	0.380	0.132	0.065	0.022	0.007
MS-SSIM+bal (awl)	**0.581**	0.488	0.411	0.133	0.053	0.012	0.003
GDL+bal (eq w)	0.548	0.440	0.358	0.091	0.035	0.008	0.002

Table 6.4: The CSI per RR for the different models. The losses are computed on the testing dataset in RR. The best (lowest) value per RR is depicted in bold.

Frequency Bias	> 0.1	> 0.5	> 1	> 5	> 10	> 20	> 30
Benchmark	**0.954**	**1.058**	**1.213**	**1.413**	**0.814**	**0.290**	**0.237**
SSIM+bal (awl)	0.864	0.704	0.674	0.343	0.145	0.050	0.035
SSIM+bal (eq w)	0.853	0.718	0.678	0.388	0.196	0.084	0.067
MS-SSIM+bal (awl)	0.913	0.862	0.840	0.399	0.129	0.036	0.021
GDL+bal (eq w)	0.743	0.648	0.593	0.180	0.072	0.023	0.014

Table 6.5: The bias per RR for the different models. The losses are computed on the testing dataset in RR. The best value (closest to 1) per RR is depicted in bold.

The visual analysis of the 4 selected heavy rainfall events, of which Event 4 is depicted in Figure 6.15, can explain the better score for the higher RR with the benchmark model: All other models dissipate the rain more than the benchmark, creating larger errors for the higher RR.

The low performance of the benchmark on the low RR (see Table 6.3), combined with the distribution of the data (every image contains more low RR than high), explains the poor overall result per lead time for the MSE and MAE. The better scores for the higher RR by the benchmark, which gets weighted more in the balanced loss, explains its better overall performance for the balanced loss over lead time.

That the overall scores per lead time can give a different impression, shows the importance of the visual analysis. From the visual analysis, the conclusion can be made that for larger lead times, none of the added loss functions actually improved the predictions, but rather worsened the performance.

Moreover, the SSIM and MS-SSIM scores over lead time (Figure 6.14) show no big differences in sharpness. The GDL however, shows to be better for all models compared to the benchmark. But since the visual assessment does not show any improvements at larger lead times, the conclusion can be drawn that the addition of these loss functions did not really sharpen the predictions at larger lead times either.

However, a visual analysis of the prediction at the first lead time (see Figure 6.16) in the domain that it is trained in (NPV), shows that the addition of the perceptual loss functions actually did sharpen this prediction. Especially the combination of the balanced loss with the SSIM, which is also observable in the RR domain in Figure 6.15. This shows that the blurring at the first lead times is mostly caused by the type of loss functions in the model and can therefore be solved by the use of a perceptual loss function. However, the blurring at longer lead times is caused by different problems, namely the imbalance in the dataset and the uncertainty with respect to the location of the rain.

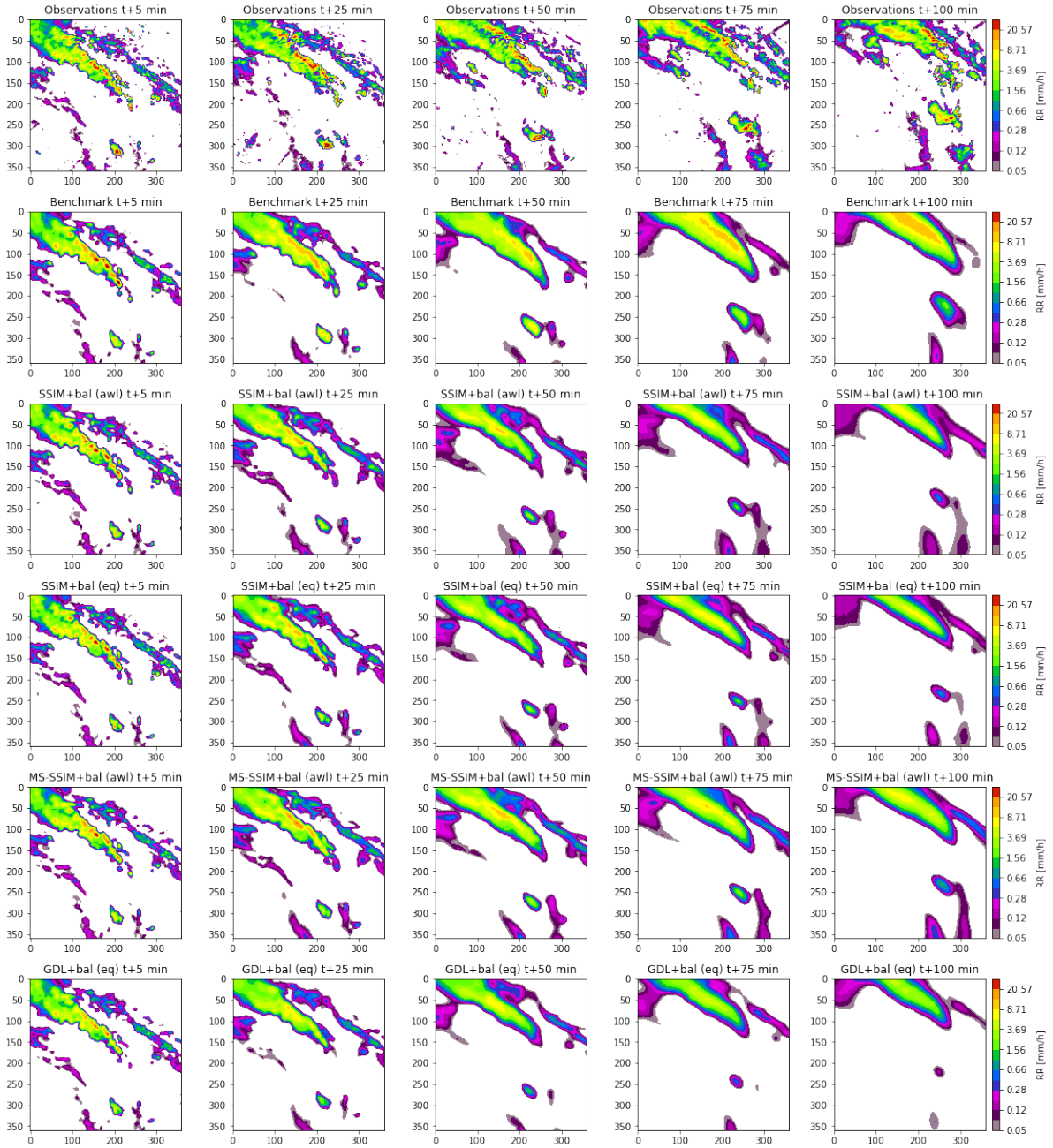


Figure 6.15: The first row shows the observations in RR for lead times +5, +25, +50, +75 and +100 min for Event 4 (from the training dataset). The other rows show the predictions in RR for the benchmark and models trained on a combination of the balanced loss and the SSIM/MS-SSIM/GDL respectively.

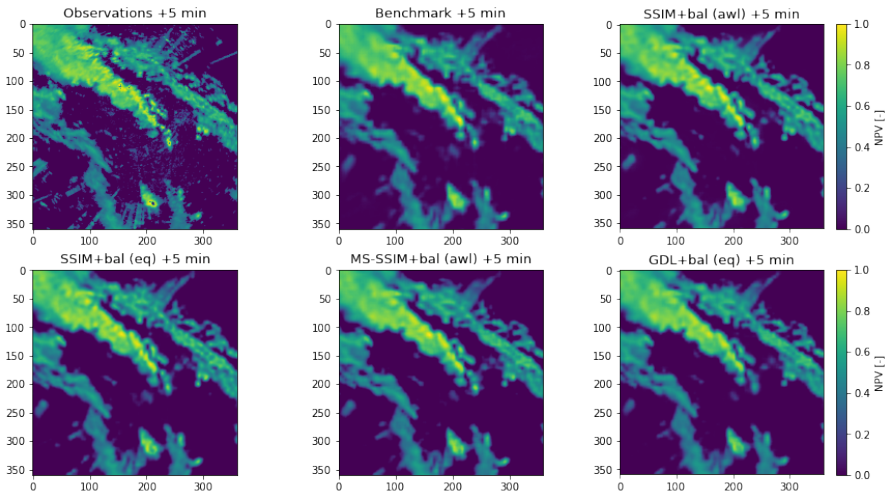


Figure 6.16: The prediction in NPV of Event 4 for the first lead time (+5 min) for all the different models.

Conclusion

This section has shown that the addition of the SSIM, MS-SSIM or GDL to the balanced loss can sharpen the prediction at the first lead time, but did not sharpen the predictions at longer lead times and even worsened the performance. It was also shown that the performance measured in any metric can differ from the visual performance, which shows the importance of the visual analysis.

Discussion

Bakkay et al. (2022) had shown that the combination of their Weighted SSIM and Weighted MSE could lead to improvements, which is in agreement with the results presented here. However, they saw improvement at more lead times. This might be explained by that they did classification of precipitation instead of regression. In such a classification problem, the location dependence becomes smaller, since multiple different valued pixels are grouped together. This might explain why they saw improvement at more lead times.

However, Yin et al. (2021) and Tran and Song (2019) also showed that their combination of the SSIM or MS-SSIM with a MSE and/or MAE loss leads to better results. This contradict our findings. This contradiction is probably not caused by the way the losses are combined, since they also show that the SSIM/MS-SSIM on its own can already improve the results and they do not show any problems with an imbalanced dataset.

Tran and Song (2019) also found that the MS-SSIM is ineffective, which is in contradiction with Yin et al. (2021). This research showed less improvement in sharpness at the first lead time for the MS-SSIM than for the SSIM (Figure 6.16). The scale with the details is weighted less in the MS-SSIM, causing the model to not focus equally much on these details. This observation might be what Tran and Song (2019) observed.

Ayzel et al. (2020) showed that there is a difference between smoothing at short and long lead times. They report that their model smooths out the rain for small scales (up to 16 km) at +5 min and that for longer lead times this smoothing extended to larger scales up to 32 km. They argue that the smoothing at short lead times is a property of the model and one of the solutions they mention is to change the loss function. As is shown here, the smoothing at the first lead time is the type of smoothing that can be corrected for by a perceptual loss function.

In this section it was mentioned that the combination of the losses might not have been optimal. The performance of the model with equal weighting was slightly better than when the combination was made with AWL. This is in agreement with Vandenhende et al. (2021) and Gong et al. (2019), who found that the automatic weighting schemes often perform worse than a simple combination or a single loss. It is also interesting to notice that none of the papers on precipitation nowcasting state how they found their weights that they used to combine the losses. They often just taken one, or otherwise they just mention that they found it experimentally.

7

Conclusion and recommendations

7.1. Conclusion

The objective of this research was to explore how the loss function in TrajGRU can be modified to get sharper and more realistic predictions, without worsening its performance. The contribution of this work is that the different perceptive loss functions are thoroughly analysed, to understand the reasons behind how they function. To this end, the TrajGRU model has been trained with 6 different loss functions, different versions of the balanced loss, the SSIM, MS-SSIM, GDL, Wasserstein loss and FFL. A short, loss specific conclusion is given at the end of each result section in Chapter 6. Overall, the conclusion can be drawn that the performance of the perceptive loss functions strongly depend on the distribution of the dataset and whether the model is trained on normalised pixel values or rainfall rates. Furthermore, it was shown that obtaining an improvement in the metrics, does not indicate a visually better prediction.

This research started with the idea that the blurring in the predictions is mainly caused by the averaging due to the MAE/MSE loss, and could therefore be prevented by using a loss that better represents how we perceive an image. This idea is confirmed by the fact that the predictions for the first lead time can be improved by using a different loss function. However, this research shows that there are multiple other causes for the blurriness at longer lead times.

All combinations of loss functions lead to the fading out of the rain with increasing lead time. Two causes of this dissipation can be distinguished. 1) Most pixels in a radar image contain the value zero. It has been shown that this causes a problem for all loss functions, since the most likely state of a pixel becomes zero (or in case of the GDL a gradient of zero). This causes the model to predict values close to zero to minimise the error. 2) The location specificity of the loss functions, causes that whenever the model is not sure about the location of the rain, it tries to prevent large errors by forecasting the most likely state, which is zero. This gets confirmed by that the scores for models with combined loss functions are better than the benchmark, while the visual assessment shows otherwise (Section 6.6).

These two problems need to be addressed before a perceptual loss function might be able to improve the sharpness of the predictions at longer lead times.

7.2. Recommendations

Multiple perceptive loss functions were investigated in this research. For each of them, their problems were identified and discussion/recommendations were provided on how to cope with these. Two more general recommendations that I want to emphasise are: Switch back to the model settings from X. Shi et al. (2017) (See A.1), and switch to the rainfall rates during the training (Section 6.1). Switching to the rainfall rates is recommended, since the way we perceive an image depends on the relation between the pixels in an image. Training on the normalised pixel values causes you to change this relation between pixels afterwards, causing your perceptive loss function to not be able to actually fulfil its function. However, to make this work, more extended clutter removal would be needed, for example with the wradlib library (Heistermann et al., 2013), clutter removal based on spatial correlation between pixels in time (Sugier et al., 2002), based on additional data sources (Jensen et al., 2006) or based on a machine learning model that separates rain from clutter (Grecu & Krajewski, 2000; Islam et al.,

2012). While exploring these methods, one must keep in mind that the clutter removal should be able to be performed in real-time, in order to obtain the nowcasts.

This section presents ideas to deal with the two main problems that were stated in the previous section. At the end it contains a short discussion on the predictability of the radar frames over lead time, and shows ideas how to implement this knowledge into machine learning models. The discussion will be concluded with my view on the first steps forward.

There are 3 main approaches to deal with the problem of too many zero valued pixels. Firstly, a different model architecture could help, just like it was already shown that the architecture from X. Shi et al. (2017) could improve the predictions for higher rainfall rates. But there are also other model structures designed to deal with this problem. For example Rainformer from Bai et al. (2022), which has a feature balancing fusion mechanism called GFU. It solves the imbalance between features at different scales and effectively extracts the complementary information between them. The GFU uses self-attention, which captures the relation between neighbouring pixels to extract global information from the radar data. They use the GFU instead of stacking more convolution layers, since they argue that convolution leads to fake global features, causing insensitivity to high intensity rainfall.

Secondly, other data transformations could be applied on the rainfall rates, for example the log transform to obtain a more normally distribute dataset. (The normalised pixel values are normally distributed, but due to the non-linear error conversion from normalised pixel values to rainfall rates, this distribution is not normal during the analysis in the rainfall rates.) Other ideas for data transformations on radar data can be found in the review from Erdin et al. (2012). Another idea is to further preprocess the data, to rebalance the dataset, as performed by Ravuri et al. (2021) (See Section A.2). It could also be further investigated what the influence is of using a smaller dataset which only contains higher precipitation events, instead of using the largest dataset possible. A first analysis is presented in Appendix section A.4

Lastly, using a loss function that can correct for the data imbalance could be considered. For example, Ko et al. (2022) recently proposed a new loss inspired by the CSI metric, or Cao et al. (2022) proposed the hybrid weighting (HW) loss as mentioned in section A.2. These losses are designed specifically for precipitation radar data and should therefore be able to handle the unbalanced distribution. This is something that the perceptive losses investigated in this research were not designed to do.

To deal with the problem of location specificity, 3 approaches can be distinguished. Firstly, the losses could be made less location dependent. For the GDL or SSIM you could for example just compare the distributions of the gradients/SSIM values, instead of comparing them pixel per pixel.

Secondly, additional data like wind or other atmospheric variables can be provided to the model. This will probably not help for convective events, since they are chaotic, but it can help for stratiform precipitation. Ideas for data to add are temperature profiles (Bakkay et al., 2022; Mascaro et al., 2013), cloud liquid water content, latent heat release, air pressure, diurnal and annual cycle of net radiation (Germann et al., 2006), wind direction and speed (Foresti et al., 2019).

Lastly, probabilistic machine learning might be able to better access the uncertainty in location and to choose the most likely forecast. Foresti et al. (2019) have investigated this approach to deal with the blurriness. They argue that the MSE can be expressed as $bias^2 + var$, leading to the minimisation of the variance. To avoid this, they combine an AENN with a decision tree that is trained to obtain the certainties per prediction, which is then fed into the AENN.

However, something not discussed in this research is the predictability limit. Seed (2003) argue that an optimal forecast is one that does not attempt to forecast structures beyond their lifetime. Smaller scale features have a shorter lifetime than features at larger scales. Therefore they state that the blurring the predictions from NWP/extrapolation models is needed to obtain better forecasts, which was also stated by Bellon and Zawadzki (1994). Germann and Zawadzki (2002) and Germann et al. (2006) define the lifetime/predictability limit of a scale, as the time when the correlation, between the observation and predication at that scale, falls below 1/e = 0.37.

Germann et al. (2006) found that there are two sources that limit the predictability: mostly the growth and decay of precipitation, and to a lesser extend the evolution in the motion field. Therefore, in a next paper, Radhakrishna et al. (2012) added a growth/decay term to their extrapolation model and thereby improved the predictions. However, a machine learning model should discover these growth and decay

patterns itself. In the appendix section A.3, it is shown that according to a first analysis, the dataset is balanced in terms of growth and decay events. That the models do not correctly capture it and would therefore have a lower predictability limit, is caused by the location uncertainty and the imbalance in intensity in the dataset, as discussed before.

The concept of predictability can be used in 2 ways. First, to realise that such a predictability limit exists based on solely the chaos/randomness of nature, leading to that it should not be expected that the model predicts correctly after this limit. A way this could be implemented in the machine learning models, is by varying the scales at which the loss is computed with lead time, by using a convolution to downscale the image. Or by keeping all the scales but varying the weights with lead time. In this way, the model is not requested to learn patterns after their lifetime, which possibly improves the overall model performance and allows the perceptual loss functions to sharpen the images at lead times within the predictability limit. Do note, that in this case it is not attempted to sharpen predictions at longer lead times anymore. If this is desired, the predictability limit needs to be increased. This can be done by solving the two problems mentioned in this research.

Secondly, the way predictability is defined, makes it dependent on the model that is used to determine the lifetime and is therefore not a set timescale. Therefore it could be used as a metric or loss function to determine the skill/sharpness of the model, by looking at the life time for different scales.

To continue forward after this research, I think that it is most important to focus on solving the two problems. I do not think that simply using another perceptual loss can deal with these. To solve the problem of too many zero valued pixels, my preference would go to rebalancing the dataset. As shown in this thesis, using weights in the loss (balanced loss) or using masks (classify pixels as wet or dry) leads to several problems. Therefore I think that improving the dataset is a better option.

My preference for solving the problem of location specificity goes to the addition of other datasets. The other two approaches might be able to predict the higher rainfall rates, but they value the correctness of the forecast less. I think that the addition of other atmospheric variables might provide the model with more information about the location of the rain and therefore improve the model. However, I do not expect this to provide enough information to give complete certainty. Therefore I do think that we should keep the predictability limit into account and apply different weighting for the different lead times.

A Appendix

A.1. The effect of the TrajGRU architecture

As presented in Section 3.2.1, the TrajGRU architecture in the benchmark deviates from the one reported in the paper by X. Shi et al. (2017). To investigate the effect of this difference in architecture (see Figure 3.2), a model is trained, for which all the settings are set to the table of X. Shi et al. (2017). The only difference between this implementation and the settings from X. Shi et al. (2017), is that the first input is set to 1 instead of 4, to be able to run it without errors. This model is then compared with the benchmark model from van der Kooij (2021).

In Figure A.1, the scores over the lead times, for both of the models on the testing dataset, are visualised. Most metrics do not show much difference. Only the MSE (and therefore the balanced loss) show clearly that the model with the settings from X. Shi et al. (2017) performs slightly better for longer lead times.

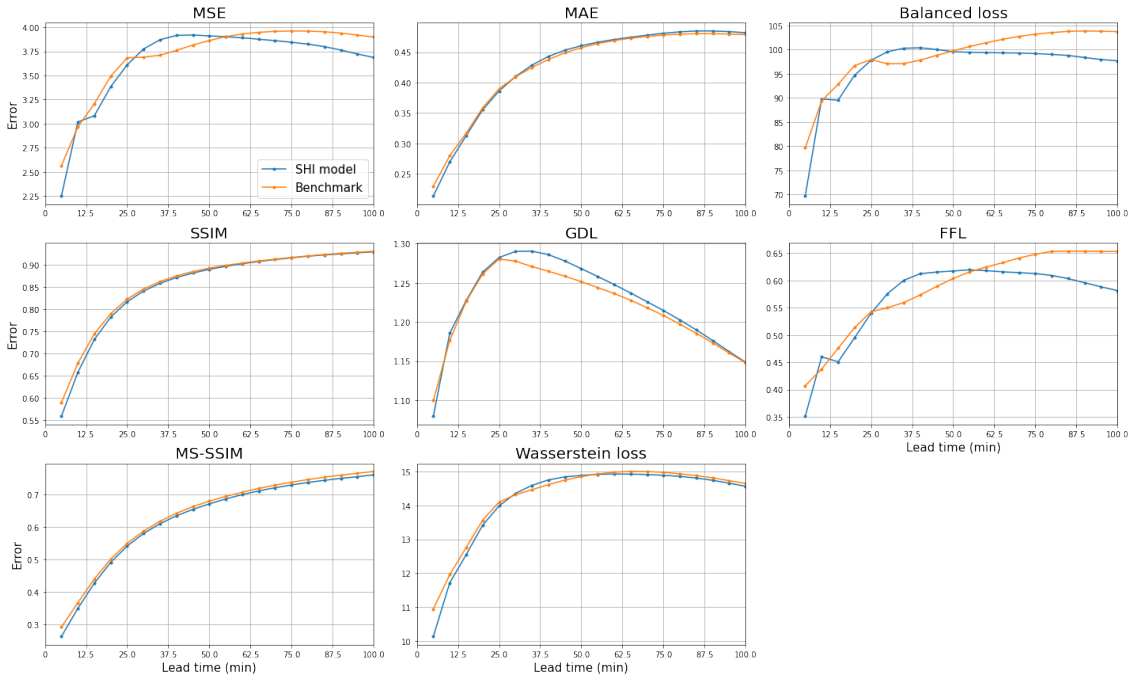


Figure A.1: The metrics over lead time, computed on the testing dataset in RR. They are depicted for the benchmark and the model trained with the architecture from X. Shi et al. (2017). A lower value indicates a better score for all metrics (the SSIM and MS-SSIM are depicted as loss, so zero indicates perfect similarity).

Analysing a single event visually, does not show these improvements for the longer lead times. It only shows a difference when a colormap is used that reserves more colors for low RR, as shown in

Figure A.2. The original model architecture by X. Shi et al. (2017) produces more low RR. This might be due to the extra convolution layer in Eva's architecture (indicated with blue in Figure 3.2), which can have filtered out these lower value. Normally a convolution layer is used to resize the image, but this extra convolution layer keeps the size the same, since the stride is 1. In that case, the kernel determines how the 3x3 pixels together determine the new value of the pixel, resulting in some kind of averaging. Therefore it can explain why the benchmark model has less low RR: these lower values occur on the sides of the precipitation areas, and get lower due to the averaging with the 0 values around it in this extra convolution layer.

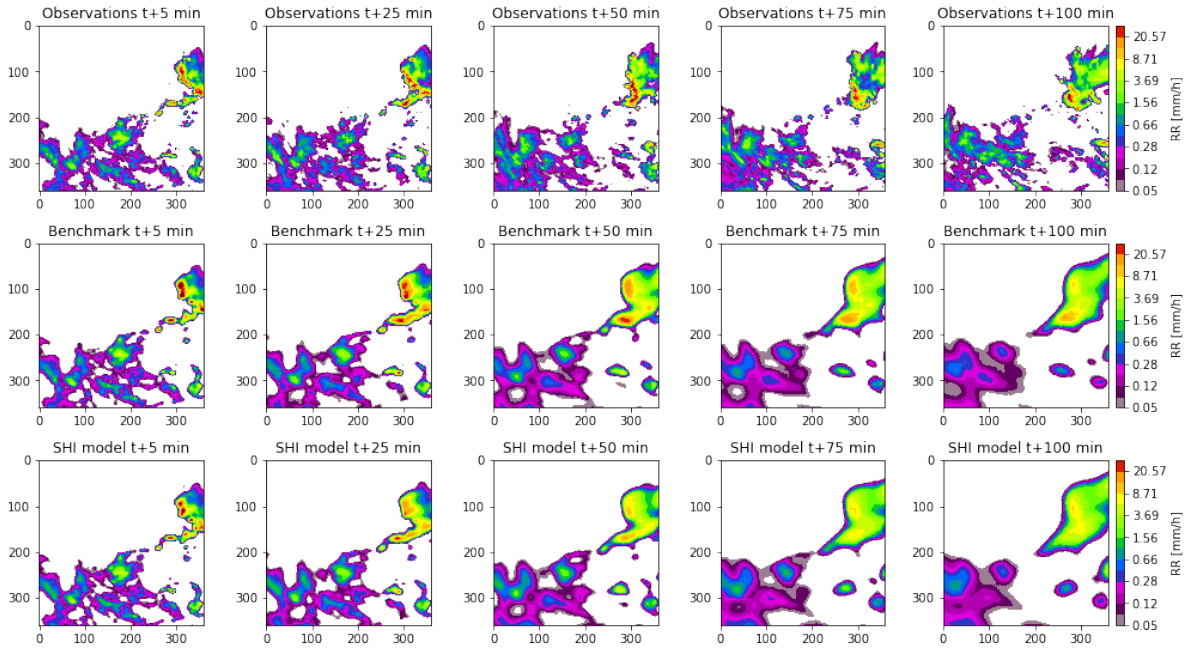


Figure A.2: The effect of the difference in TrajGRU architecture for high precipitation event 4, with a color map that reserves more colors for the lower RR. The first row shows the observations for lead times +5, +25, +50, +75 and +100 min. The other rows show the predictions for the benchmark and the model with the architecture from X. Shi et al. (2017), respectively.

However, the scores per rainfall intensity show a clearly better score for the rainfall group $0 \leq 0.1$ for any of the metrics. The scores for the balanced loss are depicted in Table A.1, showing that the architecture from X. Shi et al. (2017) obtains better results for intensities ≥ 1 mm/h. This same pattern is present in the MAE and to a lesser extend in the MSE. The SSIM, GDL and CSI scores show really small to no difference. The bias in Table A.2, shows that they are quite comparable for the lower rainfall intensities, but that for rain of > 30 mm/h, the bias of the model with the architecture from X. Shi et al. (2017) is lower. These better scores for the higher RR fit together with that the model with the architecture from X. Shi et al. (2017) performed slightly better at longer lead times, which is the moment where most of the errors for higher RR are present.

Balanced	[0, 0.1]	(0.1, 0.5]	(0.5, 1]	(1, 5]	(5, 10]	(10, 20]	(20, 30]	> 30
SHI model	**0.43**	6.72	79.87	**397.63**	**983.30**	**3413.20**	**11285.52**	**32173.54**
Benchmark	0.43	**6.61**	**79.28**	401.34	1017.07	3608.07	11667.05	32761.06
Improvement	1.1%	-1.68%	-0.74%	0.92%	3.32%	5.4%	3.27%	1.79%

Table A.1: The balanced loss per rain intensity for the two different architectures. The losses are computed on the testing dataset in RR. The best (lowest) value per rainfall intensity is depicted in bold. The last row shows the relative improvement of the SHI model compared to the benchmark.

Bias	> 0.1	> 0.5	> 1	> 5	> 10	> 20	> 30
SHI model	0.961	1.086	1.262	1.504	**0.788**	**0.258**	**0.152**
Benchmark	**0.954**	**1.058**	**1.213**	**1.413**	0.814	0.290	0.237

Table A.2: The bias per rain intensity for the two different architectures. The bias is computed on the testing dataset in RR. The best (lowest) value per rainfall intensity is depicted in bold.

Conclusion
It can be concluded that visually not many differences appeared, based on which the decision was made to continue with the model from van der Kooij (2021) as benchmark for this thesis. However, the analysis of the different scores on the testing dataset show quite some differences that indicate that the model settings from X. Shi et al. (2017) do produce better forecasts for longer lead times and higher RR. Therefore I would recommend to switch to the model settings from X. Shi et al. (2017) in the future.

A.2. Different balanced loss functions

As mentioned in Section 5.1.1, different weighting schemes are tested in order to actually emphasise the higher RR when training on the NPV. They are presented below and their effect on the distribution is shown in Figure A.3. The results of training on these balanced losses are visible in Figure A.4.

To actually compensate for the imbalance in the distribution and the difference in error size for small and low RR, the first balanced loss that was proposed gives 10x more weight to the high valued pixels, and 1000x less weight to the low valued pixels, as shown in Equation A.1. This is done in Balanced Version 2, and the effect of this weighting scheme on the distribution is shown in Figure A.3.

$$w_{n,i,j} = \begin{cases} R_{n,i,j} \cdot 10^{-3} & \text{if } R_{n,i,j} < 0.1 \\ R_{n,i,j} & \text{if } 0.1 \leq R_{n,i,j} \leq 10 \\ R_{n,i,j} \cdot 10 & \text{if } 10 < R_{n,i,j} \leq 30 \\ 30 \cdot 10 & \text{if } R_{n,i,j} > 30 \end{cases} \tag{A.1}$$

As can be seen in the Figure A.4, this results in too much rain. Therefore, it was tried to stimulate underprediction for lower RR values and stimulate overprediction for higher RR. This is done in Version 3 by multiplying the weight with 0.5 when you underpredict for RR under 20 mm/h, and multiplying with 0.5 when you overpredict for RR over 20 mm/h. Besides that, the same weighting scheme is used as in Version 2.

As shown in Figure A.4, such an asymmetric weighting scheme did not prevent the overprediction of rain enough and therefore we need to give some weight back to the low pixel values. Version 4 allows the weights to go up with the RR and only sets the weights for the 0 mm/h pixels to 10^{-4}. See Equation A.2.

$$w_{n,i,j} = \begin{cases} 10^{-4} & \text{if } R_{n,i,j} = 0 \\ R_{n,i,j} & \text{if } 0 < R_{n,i,j} \leq 10 \\ R_{n,i,j} \cdot 10 & \text{if } 10 < R_{n,i,j} \leq 30 \\ 30 \cdot 10 & \text{if } R_{n,i,j} > 30 \end{cases} \tag{A.2}$$

Version 4 still overpredicts the areas without rain. From the result of Balanced 2,3,4, it can be concluded that the emphasis on the zero and low pixel values is needed to prevent predicting rain everywhere.

Therefore Version 5 proposes to keep the weighting from the benchmark for the low rainfall rate values and just put some extra emphasis on the higher precipitation values, by multiplying the weights with a factor if it is above a certain threshold, as shown in Equation A.3.

$$w_{n,i,j} = \begin{cases} 0.1 & \text{if } R_{n,i,j} \leq 0.1 \\ R_{n,i,j} & \text{if } 0.1 < R_{n,i,j} \leq 5 \\ R_{n,i,j} \cdot 5 & \text{if } 5 < R_{n,i,j} \leq 20 \\ R_{n,i,j} \cdot 10 & \text{if } 20 < R_{n,i,j} \leq 30 \\ 30 \cdot 10 & \text{if } R_{n,i,j} > 30 \end{cases} \tag{A.3}$$

Figure A.4 shows that Version 5 results in the same problem as was observed when training on the RR. The model starts to train on clutter (the line in the top right corner) and it spreads out the rain as an attempt to not make a huge error for the higher intensity rainfall.

Since all these versions keep overpredicting the amount of rain, the next two explored options give less weight to the high precipitation values, to see what effect this would have on the model. Version 6 sets all the weights equal to 1, so that the original distribution is kept. Version 7 uses the opposite of the weighting scheme from the benchmark, as shown in Figure A.3. As shown in Figure A.4, both these options result in dissipation of the rain.

As a last option it was explored what the effect would be, of using higher weight on the high precipitation values in combination with an extra penalty for predicting the amount of wet pixels wrong. This extra penalty should then prevent the model from predicting precipitation everywhere. To this end, the weighting scheme of Version 5 is used and the penalty according to Equation A.4.

$$wet\_penalty = |(\#\text{ pixels} < 0.001\text{ mm/h rain in target}) - (\#\text{ pixels} < 0.001\text{ mm/h rain in prediction})| \tag{A.4}$$

Combining version 5 with a penalty for the wet area does not seem to cause any improvements (Version 8). Even though the penalty is up to 200x as large as the balanced loss, it seem to not really train on it. Not training on such a penalty term has also been observed in other models and is further discussed in Section 6.2.3.

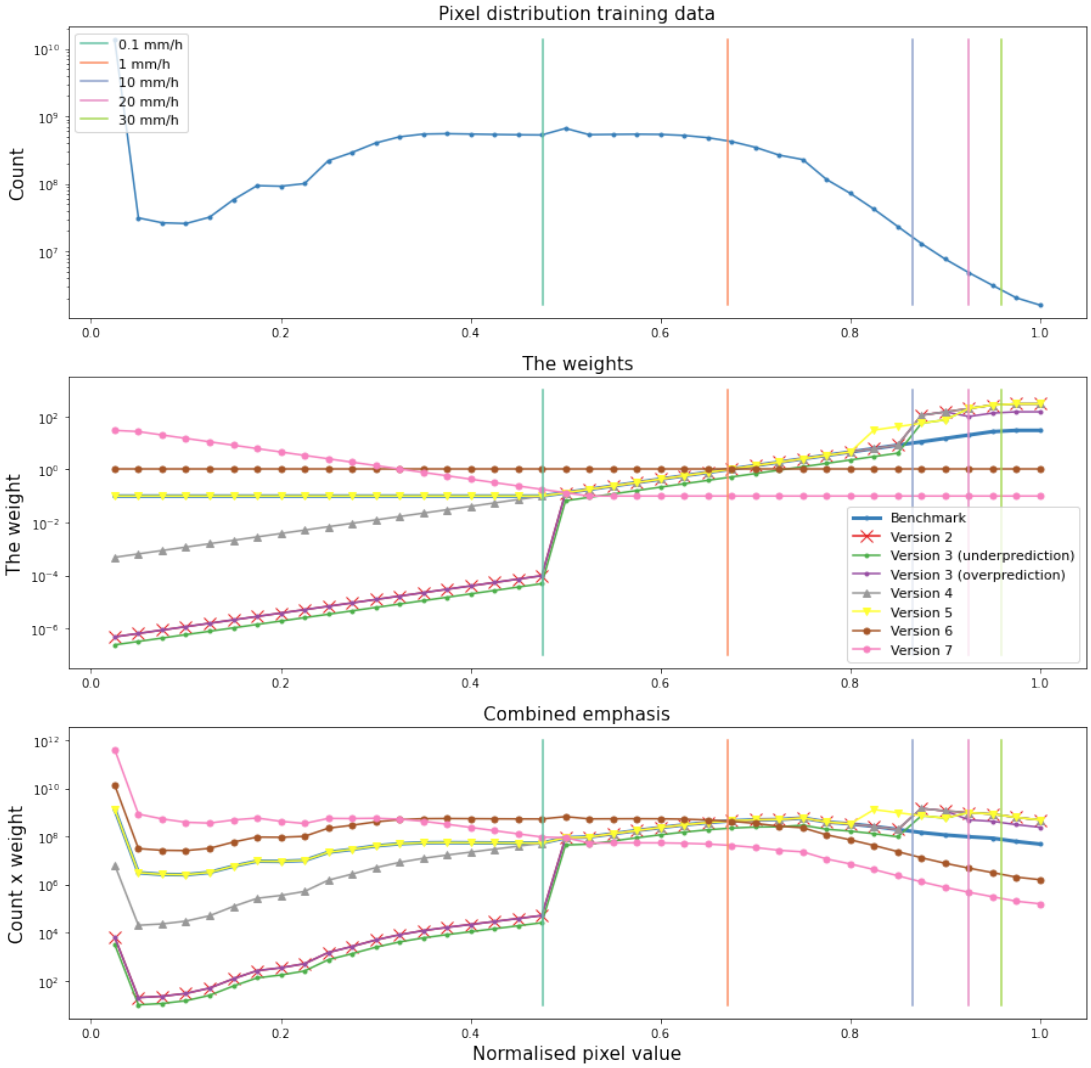


Figure A.3: The effect of the different proposed balanced loss functions on the data distribution.

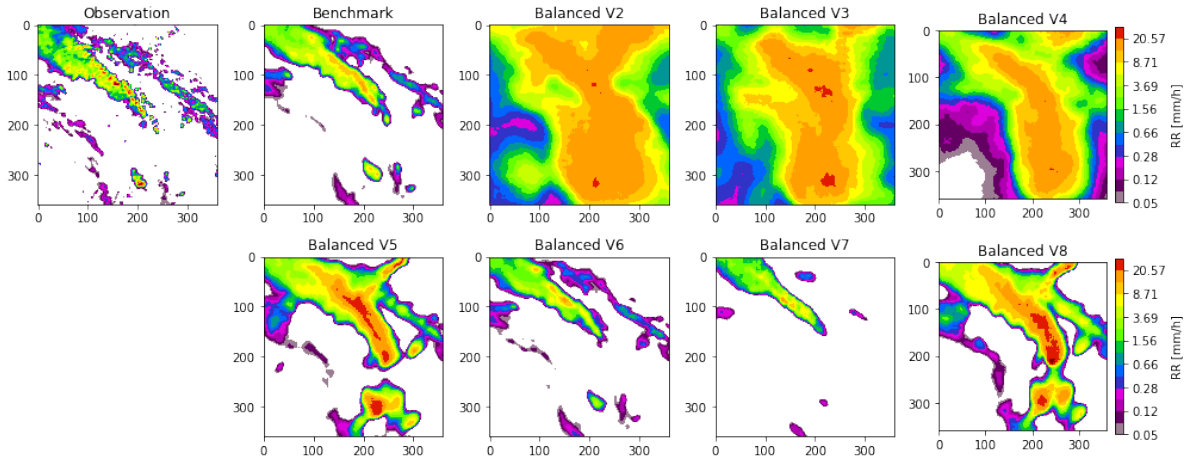


Figure A.4: The observation and predictions in RR for +25 min for event 4. The predictions are from the different proposed versions of the balanced loss.

Conclusion

From these results it can be concluded, that even though the balanced loss from the benchmark model did not compensate for the imbalance in the distribution of the data as was expected, it does seem to be an optimum between fading and spreading out the rain, and therefore perform the best compared to the other versions. The emphasis on the zero pixel values is needed in order to predict dry areas correctly.

Discussion

As concluded, it was found that emphasising the higher rainfall rates more, to correct for the incorrect error, leads to overprediction. This is in agreement with the findings from Cao et al. (2022), who argue that reweighting strategies lead to the boosting of the performance on heavy rainfall pixels, but compromise the performance on low rainfall rates. In order to prevent this, they propose the hybrid weighting (HW) loss and show that it improves the higher rainfall rates without degenerating the accuracy on the lower rainfall rates. This would be interesting to look at in future research. Ravuri et al. (2021) also mention the uneven distribution, but they solve this by rebalancing the dataset to include more data with heavier precipitation. They do this with importance sampling which they describe in their Supplementary Information section A.1. This could be interesting to look at.

A.3. Growth and decay in the dataset

The results from this research show that all models fade out the rain, for which 2 reasons are identified. However, a third possible reason could be that most events in the dataset actually do fade out the rain. Therefore, a first step is made with further analysis of the dataset, using 2 definitions of growth: 1) An increase in total intensity over lead time, 2) an increase in the amount of wet pixels over lead time.

Analysing all the events by simply comparing the last frame (frame 25) with the first frame, gives the distributions shown in Figure A.5. These distributions in the NPV and RR domain look normally distributed and centered around zero. Also the amount of decay (<0) and growth (>0) events only differ slightly, indicating that this probably does not cause the dissipation of the rain by the models.

Comparing the events that would be categorised as growth or decay according to the different definitions, shows that 6.76% (NPV) or 18.59% (RR) of the events disagree. But so far, this categorisation did not take the amount of growth/decay into account, and did not categorise events that did not grow/decay. Analysing single events, as shown in A.6, shows that growth and decay can occur within a single event, causing the total event to almost not grow/decay. Therefore, more in depth analysis on the growth and decay is needed to get a more accurate classification into growth and decay and to be able to understand how this might influence the training of the model. Thing that could be considered is a threshold for when something is seen as growth or decay. Something can be a certain percentage of the amount of rain in the first time frame, or just a fixed value.

However, as a first impression, the analysis presented here does not show any evidence for an imbalance in the dataset for growth/decay that can explain the predicted dissipation by the models.

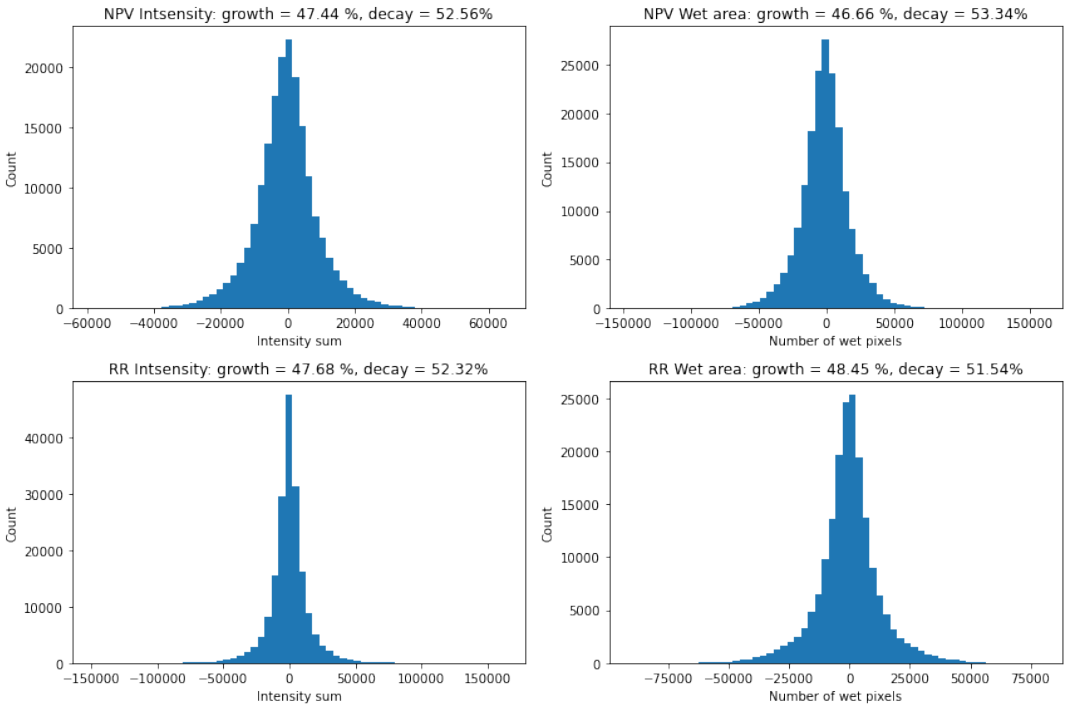


Figure A.5: The distributions of the difference between the first and last time frame in an event. Both in terms of total intensity or the amount of wet pixels, for the data in NPV and RR.

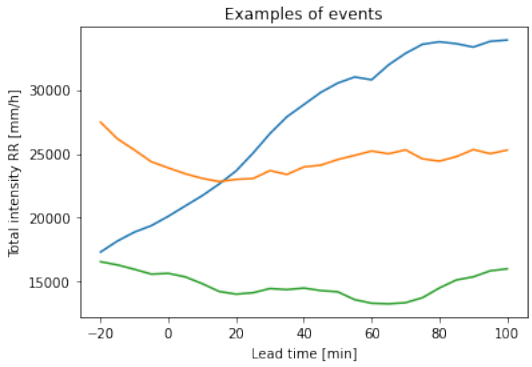


Figure A.6: The total intensity in RR per lead time for different events.

A.4. Smaller dataset

van der Kooij (2021) had shown that the use of the largest possible dataset (all_c1 in thesis van der Kooij (2021)), slightly improved the predictions in terms of MAE and RMSE, while keeping other scores approximately the same. However, in Section 6.6 it was shown that the best metric values do not need to give the best visual results. Therefore, switching to a smaller but more specific dataset for heavy precipitation (all_c4 in thesis van der Kooij (2021)), might improve the predictions. The datasets are selected the same, only using a threshold of 2% for the minimum area that should contain >1 mm/h of precipitation (all_c4), or by setting the minimum peak intensity to 10 mm/h (heavy_c1). More details can be found in van der Kooij (2021). A first analysis of this dataset is presented below. Figure A.7 shows the distributions in terms of RR and NPV. When only the wet area is changed in the selection of the events, the distributions become quite similar. Putting a higher request on the peak intensity shifts the distribution more. In terms of RR it gets slightly tilted, but in terms of NPV we can observe that this is not caused by a decrease in zero valued pixels, but by a decrease in RR values between approximately 0.05 and 5 mm/h. This gets confirmed by A.3, which shows the percentage of pixels in the first dot and in the last dot.

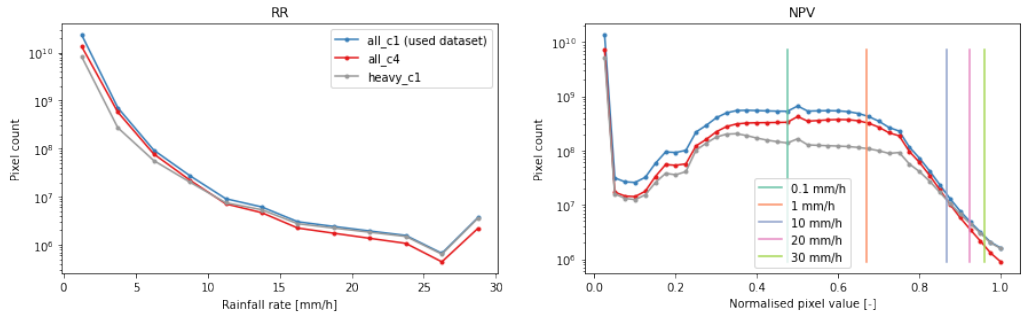


Figure A.7: The data distribution for the used and a smaller dataset, both in terms of RR and NPV.

	NPV % [0-0.025]	NPV % [0.975-1]	RR % [0-1.25 mm/h]	RR % [>28.75 mm/h]
all_c1	56.26	0.0067	96.41	0.0152
all_c4	51.23	0.0064	95.01	0.0157
heavy_c1	61.58	0.0184	95.59	0.0418

Table A.3: The amount of pixels compared to the total amount of pixels in the first and last dot in Figure A.7.

Thus, the percentage of zero and very low valued pixels do not change much in a smaller dataset, where the events are selected with more constraints. Therefore, it is not expected that changing dataset can solve the problem of the imbalanced dataset. Comparing the predictions from models trained on these 3 datasets at longer lead times (See Figure A.8) shows, that the the model trained with heavy_c1 does keeps more rain. So it can contribute to the solution, only still blurs out the higher intensity rainfall events. Therefore it would be interesting to investigate how the other perceptual losses perform with this smaller dataset.

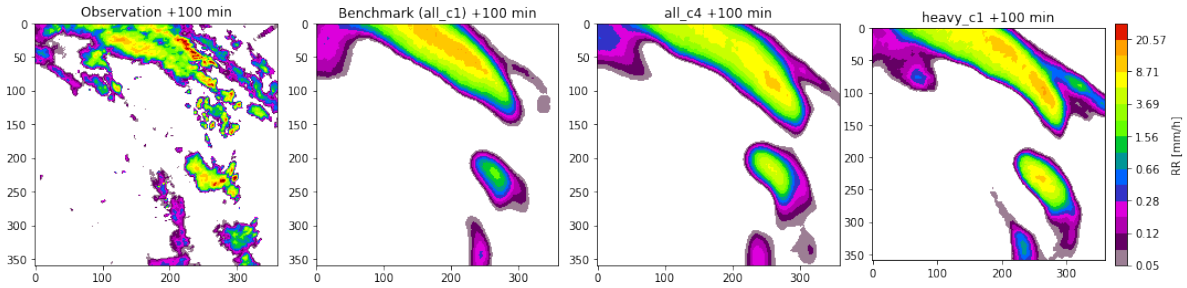


Figure A.8: The predictions in RR of the models trained with the balanced loss but with different datasets. For Event 4 at lead time + 100 min.

A.5. Guide for setting up the computational environments

For this thesis, 2 different computer facilities were used. A computer from Riccardo with a fast GPU, and the VR-Lab from GRS. (The option of using Lisa was explored as well, and environments were set up, but I did not manage to actually use the GPU instead of CPU there. Since the other 2 facilities were working, I have not solved this issue. Just know that this could be a possibility for your thesis.)

GitHub was used to easily transfer code between. To make sure that the results from different facilities are comparable, anaconda and pip environments were created, containing the same packages. These packages and their versions can be found in the file `requirements.txt`, which is compatible with both anaconda and pip.

To make the environment file you need to have the admin rights. If you do, go into your environment and execute: `pip list -format=freeze > requirements.tx` or `conda list -e > requirements.txt`.

So the idea is to create an environment on one computer/facility in which you install everything you need. Afterwards, you make the requirements.txt file and copy everything to the next environment.

Information on how to get everything ready at the 2 facilities I have used, can be found in the next subsections.

To keep track of the training process, tensorboard was used. To open this, you activate your environment in anaconda and type `tensorboard –logdir=<folder with experiment in the save folder>`. Then you copy the link to the browser and it will show the tensorboard interface. It is not possible to use the tensorboard in a linux environment at the VR-Lab, since this would require more rights. Instead, you can transfer the log files to your own computer and open tensorboard there.

To transfer the data from computer to computer, I used the psftp frogram. Instructions can be found on:
https://documentation.help/PuTTY/psftp.html#:~:text=PSFTP%2C%20the%20PuTTY%20SFTP%20client%2C%20is%20a%20tool,protocol%2C%20which%20is%20a%20feature%20of%20SSH-2%20only.

Riccardo's workstation:
This facility was a single computer with windows.
Connect remotely:

1. Connect to VPN from TU Delft
2. Open the Citrix app on your computer (or use weblogin.tudelft.nl). The webversion had some bugs for me. The app worked perfectly and you can install it for free.
3. Go to `Apps` and then to `remote desktop connection`
4. Select TUD1003202
5. Log in with your NetID

Create an anaconda environment

1. Open the anaconda prompt
2. Type: `conda create <name environment>`
3. `conda init bash`
4. To activate: `conda activate <name environment>`
5. To install packages: use `conda install <name package>`
6. To install pytorch, use the command that fits to your computer, which can be found on the website: https://pytorch.org/

Run your code:

1. Activate your anaconda environment
2. Install spyder
3. Type `where spyder`
4. Copy this location and execute it. This opens the normal spyder interface, from which you can directly run your .py files.

VR-Lab:
The VR-Lab uses Linux.
Connecting:

1. Connect to VPN from TU Delft
2. Use Putty and connect to: bastion-grs.vrlab.tudleft.nl, using SSH, open
3. Log in with your account

Setting up the environment:

1. Make a personal copy of Python available to you as a module:
   - Log in into shell-1 or shell 2 by executing `ssh shell-1`
   - Install the version of python you want. If your Python version is one of: 3.8.3, 3.8.5, 3.8.8, 3.9.6 or 3.9.7 you can install a personal copy of Python in your home directory by running the following command on shell-1 or shell-2 on the VRLab: `install-python-<version>-in-user-home-dir.sh`. If you normally have another version, try a higher python version. If you need a higher version, you need to contact the people from the VR-lab.
   - Add the code below to your .bashrc file. This file can be found with the command `ls -la / | more`
   - Log out and log back into shell-1
   - Check if it worked by typing `module avail`. This should show the installed python version.
   - Now your sbatch file will be able to load the python version you are using, which it does in the line: `module load Python/<version>`
2. Create a virtual environment with pip (virtualenv)
   - Create the enrionment with `virtualenv <name project>`
   - Activate the environment with `source <name project>/bin/activate`. This also gets done when you execute the .sbatch file to run your runs.
   - Install all the packages into your environment from the `requirements.txt` list, by executing: `pip3 -r < requirements.txt`
     Note, for me this gave some errors since some dependencies for python packages are different for a Windows or Linux environment, Therefore, I ended up to install all the packages manually anyway. Do make sure to have the same versions.

```
# Setup private modules
if [ ! -z "${MODULESHOME}" ]; then
     if [ -f /etc/lsb-release ]; then
          source /etc/lsb-release
          PRIVATE_MODULEPATH="${HOME}/Modules/${DISTRIB_ID}/${DISTRIB_RELEASE}/
                                                    Modulefiles"
          [[ -d "${PRIVATE_MODULEPATH}" ]] && module use --append "${PRIVATE_MODULEPATH}
                                                    "
     fi
fi
```

I had the data stored at a separate location: `/net/labdata/<username>`.

Run your code:

1. Log in into shell-1 or shell-2, depending on where you created your environment (`ssh shell-1`)
2. Run an sbatch file with: `sbatch <file name>.sbatch`
   See below on how to create the .sbatch file
3. Monitor the job with `squeue`
4. Cancel a job with `scancel <jobid>`

Creating the sbatch file:

Based on how the vr-lab is set up, some extra commands are needed in the .sbatch file. An example of a working .sbatch file is `Run_TrajGRU_final.sbatch` which can be found on GitHub and is also shown below. I would advice to take this one as template and only alter the python version, the source and the .py file you want to run.

After editing a .sbatch file in a text editor, you will get the error: `sbatch: error: Batch script contains DOS line breaks instead of expected UNIX line breaks.` To solve this, use `dos2unix <sbatch file name>`.

```
    #!/bin/bash
#SBATCH --job-name=SSIM_run1_VR_lab
#SBATCH --output=save_name-%j.out
#SBATCH --error=save_name-%j.err
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --nodes=1
#SBATCH --mem=50G

# Load cuda toolkit
module load cuda/10.2.89
module load Python/3.9.7

export NCCL_IB_DISABLE=1
export NCCL_SOCKET_IFNAME=eno1

# Run cuda kernel
source ~/.bashrc
source ${HOME}/thesis_diewertje/bin/activate
# the commented lines below are just to print information
# which python3
# pip3 list
# nvcc --version
# echo "Number of CUDA devices (CUDA_VISIBLE_DEVICES): ${CUDA_VISIBLE_DEVICES}"
srun python3 Test_if_model_can_run_on_workstation.py

# This sbatch file works!
```

Bibliography

Ayzel, G., Scheffer, T., & Heistermann, M. (2020). Rainnet v1. 0: A convolutional neural network for radar-based precipitation nowcasting. *Geoscientific Model Development*, *13*(6), 2631–2644.

Bai, C., Sun, F., Zhang, J., Song, Y., & Chen, S. (2022). Rainformer: Features extraction balanced network for radar-based precipitation nowcasting. *IEEE Geoscience and Remote Sensing Letters*, *19*, 1–5.

Bakkay, M. C., Serrurier, M., Burda, V. K., Dupuy, F., Cabrera-Gutierrez, N. C., Zamo, M., Mader, M.-A., Mestre, O., Oller, G., Jouhaud, J.-C., et al. (2022). Precipitaion nowcasting using deep neural network. *arXiv preprint arXiv:2203.13263*.

Beekhuis, H., & Holleman, I. (2008). From pulse to product, highlights of the digital-if upgrade of the dutch national radar network. *Proceedings of the 5th European Conference on Radar in Meteorology and Hydrology, Helsinki, Finland*, *30*.

Bellon, A., & Zawadzki, I. (1994). Forecasting of hourly accumulations of precipitation by optimal extrapolation of radar maps. *Journal of Hydrology*, *157*(1-4), 211–233.

Buishand, T. A., De Martino, G., Spreeuw, J., & Brandsma, T. (2013). Homogeneity of precipitation series in the netherlands and their trends in the past century. *International journal of climatology*, *33*(4), 815–833.

Cao, Y., Chen, L., Zhang, D., Ma, L., & Shan, H. (2022). Hybrid weighting loss for precipitation nowcasting from radar images. *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3738–3742.

Chen, Z., Badrinarayanan, V., Lee, C.-Y., & Rabinovich, A. (2018). Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *International conference on machine learning*, 794–803.

De Luca, D. L. (2013). *Rainfall nowcasting models for early warning systems*. Nova Science Publishers, Incorporated.

Erdin, R., Frei, C., & Künsch, H. R. (2012). Data transformation and uncertainty in geostatistical combination of radar and rain gauges. *Journal of Hydrometeorology*, *13*(4), 1332–1346.

Feydy, J. (2020). *Geometric data analysis, beyond convolutions* (Doctoral dissertation). Université Paris-Saclay Gif-sur-Yvette, France.

Feydy, J., Séjourné, T., Vialard, F.-X., Amari, S.-i., Trouve, A., & Peyré, G. (2019). Interpolating between optimal transport and mmd using sinkhorn divergences. *The 22nd International Conference on Artificial Intelligence and Statistics*, 2681–2690.

Flamary, R., Courty, N., Gramfort, A., Alaya, M. Z., Boisbunon, A., Chambon, S., Chapel, L., Corenflos, A., Fatras, K., Fournier, N., Gautheron, L., Gayraud, N. T., Janati, H., Rakotomamonjy, A., Redko, I., Rolet, A., Schutz, A., Seguy, V., Sutherland, D. J., … Vayer, T. (2021). Pot: Python optimal transport. *Journal of Machine Learning Research*, *22*(78), 1–8. http://jmlr.org/papers/v22/20-451.html

Foresti, L., Sideris, I. V., Nerini, D., Beusch, L., & Germann, U. (2019). Using a 10-year radar archive for nowcasting precipitation growth and decay: A probabilistic machine learning approach. *Weather and Forecasting*, *34*(5), 1547–1569.

Franch, G., Nerini, D., Pendesini, M., Coviello, L., Jurman, G., & Furlanello, C. (2020). Precipitation nowcasting with orographic enhanced stacked generalization: Improving deep learning predictions on extreme events. *Atmosphere*, *11*(3), 267.

Gao, S., Huang, Y., Zhang, S., Han, J., Wang, G., Zhang, M., & Lin, Q. (2020). Short-term runoff prediction with gru and lstm networks without requiring time step optimization during sample generation. *Journal of Hydrology*, *589*, 125188.

Germann, U., & Zawadzki, I. (2002). Scale-Dependence of the Predictability of Precipitation from Continental Radar Images. Part I: Description of the Methodology. *130*(12), 2859–2873. https://doi.org/10.1175/1520-0493(2002)130<2859:SDOTPO>2.0.CO;2

Germann, U., Zawadzki, I., & Turner, B. (2006). Predictability of Precipitation from Continental Radar Images. Part IV: Limits to Prediction. *63*(8), 2092–2108. https://doi.org/10.1175/JAS3735.1

Gong, T., Lee, T., Stephenson, C., Renduchintala, V., Padhy, S., Ndirango, A., Keskin, G., & Elibol, O. H. (2019). A comparison of loss weighting strategies for multi task learning in deep neural networks. *IEEE Access*, *7*, 141627–141632.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, *27*.

Grecu, M., & Krajewski, W. F. (2000). An efficient methodology for detection of anomalous propagation echoes in radar reflectivity data using neural networks. *Journal of atmospheric and oceanic technology*, *17*(2), 121–129.

Guo, M., Haque, A., Huang, D.-A., Yeung, S., & Fei-Fei, L. (2018). Dynamic task prioritization for multitask learning. *Proceedings of the European conference on computer vision (ECCV)*, 270–287.

Han, L., Liang, H., Chen, H., Zhang, W., & Ge, Y. (2021). Convective precipitation nowcasting using u-net model. *IEEE Transactions on Geoscience and Remote Sensing*.

Heistermann, M., Jacobi, S., & Pfaff, T. (2013). An open source library for processing weather radar data (wradlib). *Hydrology and Earth System Sciences*, *17*(2), 863–871.

Helmis, C. G., & Nastos, P. T. (2012). *Advances in meteorology, climatology and atmospheric physics*. Springer Science & Business Media.

Hess, P., & Boers, N. (2022). Deep learning for improving numerical weather prediction of heavy rainfall. *Journal of Advances in Modeling Earth Systems*, *14*(3), e2021MS002765.

Hu, Y., Chen, L., Wang, Z., Pan, X., & Li, H. (2021). Towards a more realistic and detailed deep-learning-based radar echo extrapolation method. *Remote Sensing*, *14*(1), 24.

Imhoff, R., Brauer, C., Overeem, A., Weerts, A., & Uijlenhoet, R. (2020). Spatial and temporal evaluation of radar rainfall nowcasting techniques on 1,533 events. *Water Resources Research*, *56*(8), e2019WR026723.

Islam, T., Rico-Ramirez, M. A., Han, D., & Srivastava, P. K. (2012). Artificial intelligence techniques for clutter identification with polarimetric radar signatures. *Atmospheric Research*, *109*, 95–113.

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1125–1134.

Jensen, T. B., Gill, R. S., Overgaard, S., Hansen, L. K., & Nielsen, A. A. (2006). Detecting weather radar clutter using satellite-based nowcasting products. *Proceedings of the Fourth European Conference on Radar in Meteorology and Hydrology (ERAD) 2006*.

Jiang, L., Dai, B., Wu, W., & Loy, C. C. (2021). Focal frequency loss for image reconstruction and synthesis. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 13919–13929.

Jing, J., Li, Q., Ding, X., Sun, N., Tang, R., & Cai, Y. (2019). Aenn: A generative adversarial neural network for weather radar echo extrapolation. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *42*, 89–94.

Jo, Y., Yang, S., & Kim, S. J. (2020). Investigating loss functions for extreme super-resolution. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 424–425.

Kantorovich, L. V. (1960). Mathematical methods of organizing and planning production. *Management science*, *6*(4), 366–422.

Kendall, A., Gal, Y., & Cipolla, R. (2018). Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7482–7491.

KNMI, K. (n.d.). '14 climate scenarios for the netherlands; a guide for professionals in climate adaptation (knmi, de bilt, the netherlands, 2014). *http://www.klimaatscenarios.nl/brochures/images/Brochure_KNMI14_EN_2015.pdf*.

Ko, J., Lee, K., Hwang, H., Oh, S.-G., Son, S.-W., & Shin, K. (2022). Effective training strategies for deep-learning-based precipitation nowcasting and estimation. *Computers & Geosciences*, *161*, 105072.

Lee, H., Kim, J., Kim, E. K., & Kim, S. (2020). Wasserstein generative adversarial networks based data augmentation for radar data analysis. *Applied Sciences*, *10*(4), 1449.

Lenderink, G., Mok, H., Lee, T., & Van Oldenborgh, G. (2011). Scaling and trends of hourly precipitation extremes in two different climate zones–hong kong and the netherlands. *Hydrology and Earth System Sciences*, *15*(9), 3033–3041.

Li, D., Liu, Y., & Chen, C. (2021). Msdm v1. 0: A machine learning model for precipitation nowcasting over eastern china using multisource data. *Geoscientific Model Development*, *14*(6), 4019–4034.

Liebel, L., & Körner, M. (2018). Auxiliary tasks in multi-task learning. *arXiv preprint arXiv:1805.06334*.

Liu, X., Yu, A., Wei, X., Pan, Z., & Tang, J. (2020). Multimodal mr image synthesis using gradient prior and adversarial learning. *IEEE Journal of Selected Topics in Signal Processing*, *14*(6), 1176–1188.

Lu, C., Hirsch, M., & Scholkopf, B. (2017). Flexible spatio-temporal networks for video prediction. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6523–6531.

Luo, C., Li, X., Ye, Y., Feng, S., & Ng, M. K. (2022). Experimental study on generative adversarial network for precipitation nowcasting. *IEEE Transactions on Geoscience and Remote Sensing*.

Marshall, J., Hitschfeld, W., & Gunn, K. (1955). Advances in radar weather. *Advances in geophysics* (pp. 1–56). Elsevier.

Mascaro, G., Deidda, R., & Hellies, M. (2013). On the nature of rainfall intermittency as revealed by different metrics and sampling approaches. *17*(1), 355–369. https://doi.org/10.5194/hess-17-355-2013

Mathieu, M., Couprie, C., & LeCun, Y. (2015). Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*.

Mu, B., Cui, Y., Yuan, S., & Qin, B. (2022). Simulation, precursor analysis and targeted observation sensitive area identification for two types of enso using enso-mc v1. 0. *Geoscientific Model Development Discussions*, 1–21.

Nie, D., Trullo, R., Lian, J., Petitjean, C., Ruan, S., Wang, Q., & Shen, D. (2017). Medical image synthesis with context-aware generative adversarial networks. *International conference on medical image computing and computer-assisted intervention*, 417–425.

Prudden, R., Adams, S., Kangin, D., Robinson, N., Ravuri, S., Mohamed, S., & Arribas, A. (2020). A review of radar-based nowcasting of precipitation and applicable machine learning techniques. *arXiv preprint arXiv:2005.04988*.

Radhakrishna, B., Zawadzki, I., & Fabry, F. (2012). Predictability of precipitation from continental radar images. part v: Growth and decay. *Journal of the Atmospheric Sciences*, *69*(11), 3336–3349.

Ravuri, S., Lenc, K., Willson, M., Kangin, D., Lam, R., Mirowski, P., Fitzsimons, M., Athanassiadou, M., Kashem, S., Madge, S., et al. (2021). Skillful precipitation nowcasting using deep generative models of radar. *arXiv preprint arXiv:2104.00954*.

Sato, R., Kashima, H., & Yamamoto, T. (2018). Short-term precipitation prediction with skip-connected prednet. *International Conference on Artificial Neural Networks*, 373–382.

Schmid, F., Wang, Y., & Harou, A. (2019). Nowcasting guidelines–a summary. *Bulletin nº*, *68*, 2.

Schumann, U. (2012). *Atmospheric physics: Background–methods–trends*. Springer Science & Business Media.

Seed, A. (2003). A dynamic and spatial scaling approach to advection forecasting. *Journal of Applied Meteorology*, *42*(3), 381–388.

Shi, E., Li, Q., Gu, D., & Zhao, Z. (2018). A method of weather radar echo extrapolation based on convolutional neural networks. *International Conference on Multimedia Modeling*, 16–28.

Shi, X., Gao, Z., Lausen, L., Wang, H., Yeung, D.-Y., Wong, W.-k., & Woo, W.-c. (2017). Deep learning for precipitation nowcasting: A benchmark and a new model. *arXiv preprint arXiv:1706.03458*.

Singh, S., Sarkar, S., & Mitra, P. (2017). A deep learning based approach with adversarial regularization for doppler weather radar echo prediction. *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 5205–5208.

Song, L., Wang, Q., Liu, T., Li, H., Fan, J., Yang, J., & Hu, B. (2022). Deep robust residual network for super-resolution of 2d fetal brain mri. *Scientific reports*, *12*(1), 1–8.

Sugier, J., du Chatelet, J. P., Roquain, P., & Smith, A. (2002). Detection and removal of clutter and anaprop in radar data using a statistical scheme based on echo fluctuation. *Proceedings of ERAD (2002)*, 17–24.

Tian, L., Li, X., Ye, Y., Xie, P., & Li, Y. (2019). A generative adversarial gated recurrent unit model for precipitation nowcasting. *IEEE Geoscience and Remote Sensing Letters*, *17*(4), 601–605.

Tran, Q.-K., & Song, S.-k. (2019). Computer vision in precipitation nowcasting: Applying image quality assessment metrics for training deep neural networks. *Atmosphere*, *10*(5), 244.

Tuyen, D. N., Tuan, T. M., Le, X.-H., Tung, N. T., Chau, T. K., Van Hai, P., Gerogiannis, V. C., & Son, L. H. (2022). Rainpredrnn: A new approach for precipitation nowcasting with weather radar echo images based on deep learning. *Axioms*, *11*(3), 107.

van der Kooij, E. (2021). Nowcasting heavy precipitation in the netherlands: A deep learning approach [Accessed: 15-6-2022].

Vandenhende, S., Georgoulis, S., Van Gansbeke, W., Proesmans, M., Dai, D., & Van Gool, L. (2021). Multi-task learning for dense prediction tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*.

Veillette, M., Samsi, S., & Mattioli, C. (2020). Sevir: A storm event imagery dataset for deep learning applications in radar and satellite meteorology. *Advances in Neural Information Processing Systems*, *33*, 22009–22019.

Wang, C., Wang, P., Wang, P., Xue, B., & Wang, D. (2021). Using conditional generative adversarial 3-d convolutional neural network for precise radar extrapolation. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *14*, 5735–5749.

Wang, S., Zhao, J., Shao, C., Dong, C., & Yin, C. (2020). Truck traffic flow prediction based on lstm and gru methods with sampled gps data. *IEEE Access*, *8*, 208158–208169.

Wang, Y., Coning, E., Harou, A., Jacobs, W., Joe, P., Nikitina, L., Roberts, R., Wang, J., Wilson, J., & Atencia, A. (2017). Guidelines for nowcasting techniques. *WMO publication, published online: https://library. wmo. int/opac/doc_num. php*.

Wang, Z., Simoncelli, E. P., & Bovik, A. C. (2003). Multiscale structural similarity for image quality assessment. *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, *2*, 1398–1402.

Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., & Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting, 802–810.

Xu, K., Li, G., Xu, H., Zhang, W., & Huang, Q. (2018). Edge guided generation network for video prediction. *2018 IEEE International Conference on Multimedia and Expo (ICME)*, 1–6.

Yan, B.-Y., Yang, C., Chen, F., Takeda, K., & Wang, C. (2021). Fdnet: A deep learning approach with two parallel cross encoding pathways for precipitation nowcasting. *arXiv preprint arXiv:2105.02585*.

Yin, J., Gao, Z., & Han, W. (2021). Application of a radar echo extrapolation-based deep learning method in strong convection nowcasting. *Earth and Space Science*, *8*(8), e2020EA001621.

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning. *arXiv preprint arXiv:2106.11342*.

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., & Wang, O. (2018). The unreasonable effectiveness of deep features as a perceptual metric. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 586–595.

Zhao, H., Gallo, O., Frosio, I., & Kautz, J. (2016). Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*, *3*(1), 47–57.

Zhao, H., & Wildes, R. P. (2021). Interpretable deep feature propagation for early action recognition. *arXiv preprint arXiv:2107.05122*.