Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

DEEP LEARNING APPROACH FOR PROCESSING RADAR DATA

SUBMITTED BY

Tobias Machnitzki

MASTER'S THESIS

FAKULTÄT FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

UNIVERSITÄT HAMBURG

2019

1st Supervisor: Dr. Marco Clemens
Meteorologisches Institut, Universität Hamburg

2nd Supervisor: Prof. Dr. Felix Ament
Meteorologisches Institut, Universität Hamburg

SUBJECT

Processing Local Area Weather Radar Data with Neural Networks

Abstract

In recent years one buzzword can be found more and more along all scientific fields: neural networks. With steadily increasing computational power, deep learning is capable of solving many problems and especially image processing is gaining momentum in terms of applicability. Weather radar measurements are not very different from any other images despite being in polar coordinates. The question is whether it is possible to use modern neural network technology to process these measurements.

Within this thesis a neural network for processing radar data based on the U-net structure and the convolutional neural network technology is derived. The network is able to use raw reflectivity measurements as input and produce images which are free of noise and non-meteorological echoes (clutter). The aim is to use the network on single-polarized radars which have reflectivity as only measured quantity, otherwise clutter could be removed with the linear depolarization ratio or the Doppler velocity.

State-of-the-art concepts are presented and implemented into the neural network, as for example batch normalization. Not only the network itself, but also the generation of applicable datasets is presented, since the training dataset is one of the most important parts for the quality of the network predictions. Therefore, two approaches are compared, one using a Gaussian random field generator to produce artificial radar images, the other using the Noise2Noise (N2N) approach by Lehtinen et al. (2018). The neural network is optimized using hyper parameter optimization with a random search procedure.

The predictions of the network are radar images with no noise and clutter left. Using the Structural Similarity Index Measurement (SSIM) as metric for comparing the results, the Convolutional Neural Network for Radar Data (CNNR) achieves a mean score of SSIM = 0.98 over 3000 artificially generated radar images. The operational software achieves a SSIM = 0.96 at the same dataset and takes about 100 times longer for the computation.

Additionally, the trained neural network has a complete size of less than 10 MB. This is why inference with the trained neural network is computational inexpensive and hundreds of images can be processed within seconds, due to the optimization of the Network on running on Graphics Processing Units (GPUs). Therefore, the derived neural network is a good addition to existing algorithms and it is superior especially when the results are used only for displaying the data.

Zusammenfassung

Ein Schlagwort das in den letzten Jahren mehr und mehr in der Welt der Wissenschaft gefunden werden kann, sind neuronale Netzwerke. Dank stetigem Leistungswachstum moderner Computer sowohl im privaten, als auch im Super-Computing Bereich, sind neuronale Netzwerke mittlerweile fähig auf viele verschiedene Fragestellungen angewendet zu werden. Besonders in der industriellen Bildverarbeitung gibt es kaum noch andere Anwendungen. Da Radarmessungen als Bilder in Polar-Koordinaten betrachtet werden können, liegt es nahe, dass neuronale Netzwerke auch bei der Verarbeitung dieser Messdaten erfolgreich sein können.

Im Rahmen dieser Arbeit wurde ein neuronales Netzwerk konstruiert, welches die U-Net Architektur und die Convolutional Neural Network Technologie benutzt. Das Netzwerk ist in der Lage die rohen Radarmessungen zu verarbeiten, sodass ein Störungsfreies Ergebnis entsteht. Das neuronale Netzwerk wird auf Messungen von einfach polarisierten Radaren angewendet, deren einzige Messgröße die Reflektivität ist.

Zur Optimierung des neuronalen Netzwerkes werden modernste Technologien, wie z.B. Batch-Normalization verwendet. Für das Erstellen der Trainingsdaten wird ein Regenfeld-Generator konstruiert, welcher auf der Basis von Gausschen zufälligen Fäldern künstliche Radarmessungen generieren kann. Zudem wird eine alternative Methode zum Erstellen der Eingangsdaten gezeigt, bei welcher zeitlich Mittel benutzt werden, um Rauschen aus Eingangsdaten zu entfernen (Lehtinen et al., 2018). Mittels eines Random-Search Verfahren wird die bestmögliche Kombination an nicht trainierbaren Parametern ermittelt, sodass ein optimales Netzwerksetup entsteht.

Die vom Netzwerk ausgegebenen Ergebnisse weisen keinerlei Rauschen mehr auf und nicht meteorologisches Echo (Clutter) ist zum größten Teil entfernt. Zur Quantifizierung der Ergebnisse wird der Structural Similarity Index Measurement (SSIM) verwendet, welcher zwei Bilder A und B vergleicht und genau eins ist, wenn $A = B$. Das in dieser Arbeit konstruierte Netzwerk erreicht für 3000 Testbilder einen mittleren SSIM $= 0.98$, wobei die vom Netzwerk produzierten Ergebnisse mit einer zuvor erstellten Wahrheit verglichen werden. Die bisher operationell benutzte Methode erreicht einen SSIM $= 0.96$, bei einer 100-fach längeren Rechenzeit.

Das Netzwerk ist kleiner als 10 MB groß, sodass es auf kleinen Endgeräten wie Raspberry Pis oder Smartphones laufen kann. All dies macht die Prozessierung von Radarmessungen mittels neuronaler Netzwerke zu einer guten Ergänzung zu bereits bestehenden Algorithmen.

Contents

List of Figures

List of Tables

Acronyms

CL	Custom Loss
clutter	non-meteorological echoes
CNN	Convolutional Neural Network
CNNR	Convolutional Neural Network for Radar Data
CPU	Central Processing Unit
DSSIM	Difference of Structural Similarity Index Measurement
GPU	Graphics Processing Unit
LAWR	Local Area Weather Radar
MAE	Mean Absolute Error
MSE	Mean Squared Error
N2N	Noise2Noise
NN	Neural Network
pylawr	Python Package for Weather Radar Processing
SSIM	Structural Similarity Index Measurement

1 Introduction

Once upon a time, in the year 1943 McCulloch and Pitts proposed an idea about how the brain might work. Using electric circuits they built the very first Neural Network (NN). It took another twelve years for computers to be able to simulate a NN, so that a small group at IBM research laboratories built the first abstract neural network in 1955. This was the first NN on software technologies. Their results were not very promising at that time, though (Goyal and Benjamin, 2014). But shortly after this the first results with a model called Multilayer Adaptive Linear Neuron (MADELINE) were achieved in Stanford by filtering out echoes from a telephone line (Widrow and Hoff, 1960). After this the computational resources were exhausted for many years. Only 1990 LeCun et al. proposed the backpropagation method to adjust weights inside neural networks. This was now possible due to the increase in processing power. A specialization of NNs came up, which was designed especially for processing image data (LeCun et al., 1995). This new technology was called Convolutional Neural Network (CNN) and became the state of the art in image and language processing (Krizhevsky et al., 2012; Collobert and Weston, 2008).

In recent years neural networks got some momentum inside the science community and finally even in nature- and particularly atmospheric science (Reichstein et al., 2019; SHI et al., 2015). But so far the usage is sparse and there are only a few publications on neural networks for meteorological tasks. There even have been first steps in using artificial intelligence for clutter detection in polarimetric radar images with the use of support vector machines (Islam et al., 2012). Contrarily to the nature science community, the industry uses NNs heavily. One of the most attention-getting sectors is the automobile industry. They are instrumental in not only applying, but also advancing deep learning techniques. Their principle purpose with this is to bring self-driving cars on the market (Bojarski et al., 2016). Another huge area of interest is the recognition of faces in images (Parkhi et al., 2015). All social media platforms are using it by now for gaining contextual information (Stone et al., 2008). Advertisement can be customised to each person with these information, for example. The reason that the industry is using deep learning progressively is that the results of their NNs are getting increasingly more accurate, as the frameworks

are getting easier to use.

Radar images are in principle not very different from images taken with a photo camera. They distinguish only in two facts. First, camera images have three color dimensions while radar images only have the reflectivity. Second, due to the measuring technique of a revolving instrument, radar images are stored in polar coordinates while most other images are available in Cartesian coordinates. Finding the ground truth of measured radar reflectivities is nearly impossible, especially when reflectivity is the only measured parameter by the radar. An ideally set up rain radar at the perfect location would in theory only measure back scattered signals by water particles of the size of rain drops. In reality the electric components of the radar are not perfect though and therefore induce noise to the measurements. The noise is not even constant but varies with the meteorological conditions. Furthermore, the field of view for the beam is usually not completely free of objects like houses, trees or other similar structures. These reflect the radar beam and thus induce static clutter into the radar image. Another source of not wanted signals are other radars. If a radar detects the electromagnetic wave from another radar with the same frequency, this signal will be most likely interpreted as a back scattered signal from the first radar and thus a spike of high reflectivity appears in the measurements of that radar. This happens outstandingly often with X-Band radars, when there is water with sea traffic close to the radar location, as most ships have their own X-Band radar for traffic monitoring. A different reason for those spikes can be caused by running multiple radars at the same frequency bandwidth with overlapping field of view in a network setup. In section 2 the measuring techniques of the used radars and their operational processing is being presented in more detail.

The guiding question this thesis likes to investigates is: When deep learning is deployed so successfully in the industry on classic images, is it possible to apply it on radar images for getting noise- and clutter-free results?

When speaking about building a neural network the structuring of the layers is meant, as well as which activations are used and how the network loss is measured. An introduction to neural networks is given in section 3. The focus there is set on the terminology used in NN-literature. As already stated, one of the main differences of camera images to radar images is the underlying coordinate system. As most CNNs are used on Cartesian input data, a simple test scenario is presented to investigate whether it is possible to use polar gridded input data as well (Sec. 4).

Filtering all the non-meteorological signals from the measured reflectivities is a complex challenge. Usually mathematical concepts as well as image processing techniques are used together to achieve best results. Nevertheless some clutter and noise still remain in most results and the processing is a time consuming and computa-

tional challenge. For a supervised learning approach with NNs the network would best need perfect clean radar images as targets to train on. No analytic algorithm yet exists to produce those and therefore it has to be developed. Two methods for this are shown in section 5.

The network structure and technologies are described in detail in section 6. Not only the used technologies are shown there, but also the custom implementations are presented. In section 7 the results of the NN are investigated by comparing its result over 3000 images to the results of the Python Package for Weather Radar Processing (pylawr). Additionally, two cases from that study are investigated in more detail. At the end some conclusions are drawn from the investigations and a few suggestions on how to use the derived results are proposed (Sec. 8).

2 The Operational Radar Processing

All the radar measurements used in this thesis were produced by a self developed weather radar from the University of Hamburg. The radar is based on a nautical radar technology and has a transmission power of 25 kW. Its wavelength is in the microwave frequency band (X-Band) which makes it perfect for rain detection on a high temporal and spatial resolution. The downside of the high frequency is a shorter range than with lower frequencies (Skolnik, 1970). Together with the relatively low transmission power a range of 20 km is reached. The radar is placed in the center of Hamburg, on the rooftop of the Geomatikum building (53.568° E, 9.974° N, 90 m above sea level) so that the 20 km radius covers all of Hamburgs city center and its vicinity. The radar rotates with 24 revolutions per minute and the measurements are stored every 30 seconds as averages. The spatial resolution is composed of 333 range gates and 360 azimuth angles. Each range gate has a length of 60 m. The angular resolution is 1° (Lengfeld et al., 2013).

The radar only measures reflectivities and no additional parameters. Reflectivities are measured by emitting a microwave signal which gets reflected by an object and the reflected wave gets detected by the radar again. The time between emission and receiving of the signal is measured and with an assumed velocity for the microwave the distance can be calculated. The time has to be halved because it was measured once for the path from the antenna to the object and once from the object back to the antenna.

The problem is that not only wanted objects like rain drops reflect the microwave signal but also unwanted objects like buildings, planes and birds. These non-meteorological echoes are called clutter. Other phenomena like interference can produce not wanted signals when two radars are pointed at each other or when other devices transmit at the same frequency (Saltikoff et al., 2016). This can cause a spike of high reflectivity in the radar images, because signals sent from another device are interpreted as strong back scattering of a small object. In general clutter is split into two categories, static and dynamic clutter. More advanced radars like

dual polarized radars do not only measure the reflectivity, but also parameters like the depolarization ratio and Doppler velocity. This gives additional information of a detected object (Doviak et al., 2006). With such information a large amount of clutter can be easily excluded from the measurements (Lee, 2003). The single polarized X-Band radar from the University of Hamburg was aimed to be an affordable radar, so that a whole network of these radars could be built. On the downside these radars do not measure those additional information and thus clutter and noise have to be filtered out by software solutions.

For filtering out static clutter usually clutter-maps are used. These are maps which hold information about where a static clutter point inside the radar image usually occurs and than removes this measurement. With this technique the clutter of buildings, trees and wind power plants can be easily removed. Other phenomena such as the previously described spikes can be removed by using algorithms originally developed for image processing. The removed data has to be filled again, otherwise artificial gaps would occur in the radar images. Therefore level-1 data can not be called true measurements anymore, because for filling the gaps lots of assumptions have to be made.

Another unwanted signal, which is included in every radar image, is noise. Most noise comes from the electric and moving parts of the radar, but also atmospheric noise plays an important role (Spaulding and Washburn, 1985). Since conductors are temperature dependent the noise is temperature dependent as well. And not only temperature, but also humidity and other factors influence the noise level. Therefore, the noise can not simply be subtracted like a static number. On the other hand, a common way of removing the noise is by estimating a noise level (Lengfeld et al., 2014) and than subtracting 103 % of it. The estimated noise level usually does not account for peaks in the noise and therefore the extra 3 % are subtracted to make sure removing all of it. The downside of this method is that some of the wanted signal get canceled out as well.

The primary objective for producing noise- and clutter-free radar images is to convert the reflectivities into rain rates for displaying it to the public. The conversion is usually done using the distribution by Marshall and Palmer (1948). The quality of this transformation is heavily dependent on the quality of the underlying reflectivity field, though (Lee and Zawadzki, 2005). When the reflectivity field includes clutter in form of spikes for example, those are converted to rain rates when in reality there is no rain. Empirical all those errors are easy to detect, but practical they are very hard to eliminate. This is where neural networks outshine the operational processing.

3 | Neural Networks

Often NNs are denounced as their results are not fully traceable, because the underlying model is not based on physical and conservation laws. For this statement NNs are not in line for scientific questions by many researchers. On the other hand, analytic and numeric algorithms for detecting objects in images fall far behind modern CNNs. So NNs achieve reasonable accuracy at tasks which often are intuitive for humans but not really accessible with classical computational approaches.

The field of machine learning belongs to the huge research area of artificial intelligence that among others try to close the gap on how computers can further help humans in constantly recurring tasks, like recognizing and labeling objects in images (Karpathy and Fei-Fei, 2015). It is further divided into areas like regression trees (Quinlan, 1986), clustering (MacQueen et al., 1967) and regression learning (Loh, 2011). Another of these areas are neural networks, and especially when these networks become larger and larger the common term used to address this theme is "deep learning". There is no global definition on when a neural network falls into the section of deep learning, but as elaborated by Goodfellow et al. (2016): The terminology deep learning is just another historically grown name for the same thing which once was addressed by the term artificial neural network. Thus those two phrases will be used in equal meaning in this thesis.

The aim of this section is not to explain every detail on how neural networks work but to give an overview on the most used terminology. The necessary fundamentals are provided to understand the methodology and results of this thesis.

3.1 What is a CNN?

Usually when referring to Neural Networks (NNs) fully connected networks are proposed. The term *fully connected* means that every node from one hidden layer is connected to every node of the next layer (Fig. 3.1). The weight of each connection causes the input data to be adjusted and this way the NN can learn to represent the target data.

A special form of neural networks are Convolutional Neural Networks (CNNs),

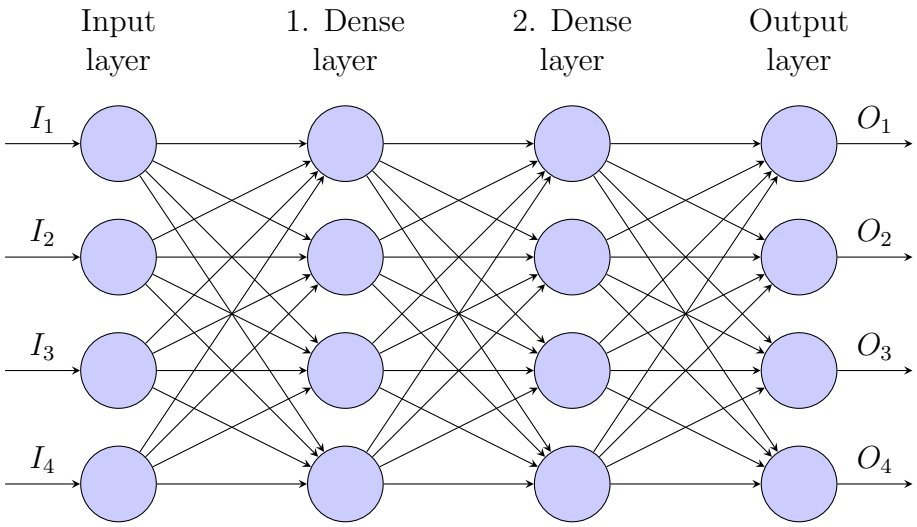


Figure 3.1: *In the fully connected neural network, each node from one layer is connected to every node of the previous layer. A fully connected layer is also called Dense layer.*

which were first successfully implemented by LeCun et al. (1998) to recognize hand written characters. Already at this time they outperformed all previous established algorithms. More recently CNNs got more attention, because they are preeminent in image recognition tasks (Krizhevsky et al., 2012) and are used by many social networks to identify persons in images (Ahmed et al., 2015) for gaining information about relations. But with research going on CNNs were discovered to be able to do a lot more then only image recognition. The right structuring of the convolution layers allows that the information gained by the convolutions can be scaled upwards again and used to combine the information of the recognition with the exact positional information in the original input data (see section 6.1).

The huge advantage of CNNs over fully connected NNs is using the assumption that the input data is an image. As a result, information about neighbouring pixels are known. Such information are for example that pixel being far away from another do not correlate as strong as adjoining ones. Therefore, a connection of each node with every other node from the previous layer is not necessary. Instead, only a connection to some nodes is built (Fig. 3.2) (LeCun et al., 1995). The reduction of connections between the nodes lowers the computational costs enormously and thus it is possible to pass larger input data to the network, like full scaled images. In most cases, even with todays computers, this would otherwise not be possible.

CNNs can be further divided into the two kinds of regression and categorical networks. The former predicts continuous numbers or even multidimensional fields while the latter predicts only a single natural number per input. Usually a human readable label belongs to each natural number. This separation into regression

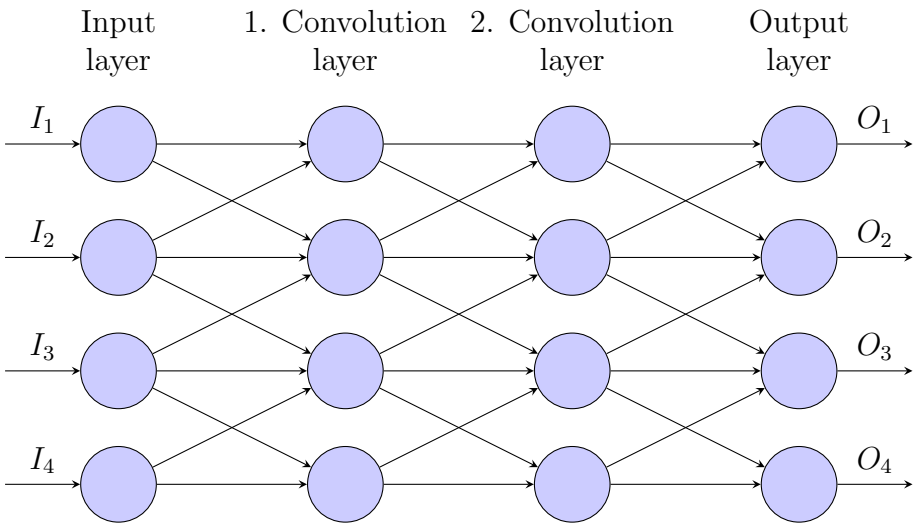


Figure 3.2: *The connection of the nodes within a convolutional neural network. The output layer has the same shape as the input layer and Nodes are only connected to neighbouring nodes of the previous layer.*

and categorical networks is important to know, because it determines which loss functions can be used (Gunn et al., 1998). Since the CNNR belongs to the first class, only loss functions for regression networks will be described.

3.2 Neural Network Terminology

When talking about neural networks a few terms need to be explained. Some have already been used in this thesis without further illustration like *loss function* or *hidden layer*. The following listing is a clarification to the most used terms. Since there is not a general definition for most of these phrases, it is as well a definition on how these items are used in this thesis. For deeper information on many of the following terms have a look at the paper by Kröse and van der Smagt (1993) or the book DEEP LEARNING by Goodfellow et al. (2016).

Datasets

Deep learning can be referred to as a data driven science (Strasser, 2012). Therefore, to train a NN a dataset with a sufficient number of samples has to be available. This dataset is usually split into a training dataset, a validation dataset and a testing dataset (Sigal and Black, 2006; Fergus et al., 2005). The fractions which are used for dividing the dataset differ from problem to problem but in general the training dataset should be the largest part.

The training dataset is used for training the model and adjusting its weights. After each epoch the validation dataset is used for checking the generalization of

the training. Only after the model is completely trained and tuned the testing dataset is used to gather the overall accuracy of the NN.

There is no general definition on when a dataset has enough samples. As rule of thumb more samples are needed if the model converges to a minimum on the training data but the accuracy on the validation data is low (Bengio, 2012). Since the model achieves good accuracy on the training data its architecture seems to work. On the other hand, the accuracy on the validation data is low, which means it does not generalize enough. More training data could solve this.

Epoch

An epoch consists of all the training images. After one epoch of training the network has seen the complete training dataset once. For the reason that not all the data of one epoch fit in the memory of a computer at once it is usually divided into several batches of fixed size. These batches are small enough to fit in the memory.

Batch

The number of images which the neural network gets to see before adjusting its weights is summarized as one batch. When training the network on multiple processing units the batch size should be chosen as a multiple of the number of processing units. This is due to the way most neural networks have implemented the multiprocessing. The number of batches is divided by the number of processing units, so that each processing unit gets the same number of batches. The model itself is copied to each processing unit as well, and each copied model then computes its gradients. The results are gathered after every batch and the gradients are updated simultaneously for all the models. This form of parallelism is called data parallelism.

Weight

Every connection between two nodes has its own weight. The weight is a value to determine how strong that connection is. The stronger a connection (higher weight), the more likely it is for the target node to be activated. The weights are typically initialized at the beginning of the training with random values and special criteria such as the Xavier uniform initializer (Glorot and Bengio, 2010) and then get adjusted during the training. Therefore they are the parameters that effectively get learned when training a NN. There are other parameters that get trained too, but weights supply the majority of trainable parameters in every neural network.

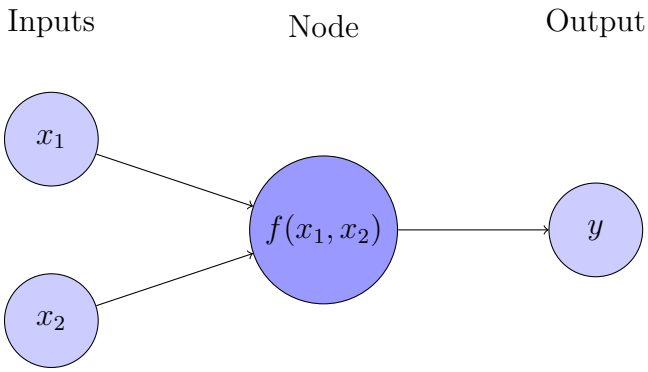


Figure 3.3: *Schematic image of a node. The inputs to the node are multiplied by their weights before entering the function $f(x_1, x_2)$. In a real NN, one node usually has lots more inputs and outputs.*

Node

Every layer of a NN has many nodes. Sometimes they are called neurons. They can be thought of as the computational unit of the neural network. Each node of a hidden layer has many inputs which are being processed and then the same result is distributed to the different outputs of the node. Let a node have two inputs x_1 and x_2. These are the outputs from a previous layer. To connect x_1 and x_2 with the actual node they are multiplied by weights w_1 and w_2. The node itself takes the inputs, applies some function to them and passes the result to the next layer. The function of a node first sums all the incoming results, adds a bias b to this sum and then applies the activation function to the result (Fig. 3.3). If the activation function was just a linear function then the previous defined example node would result in

$$f(x_1, x_2) = f(x_1 \cdot w_1 + x_2 \cdot w_2 + b) = x_1 \cdot w_1 + x_2 \cdot w_2 + b. \tag{3.1}$$

The bias is a trainable parameter for the neural network as well as the weights. If the node has more than one output (Fig. 3.2) the same result is distributed to all outgoing connections.

Layer

A collection of nodes is clustered in a layer. In a fully connected network each node from one layer is connected to each node of the next layer. How strong each interconnection is being used, is determined by the weight and the activation function. Every layer which is not an input or output layer is called hidden layer.

Depth

Depth is a term used for measuring the complexity of a neural network. The more layers a network has, the deeper it gets. A deeper network can represent more complex functions and therefore it can solve more complex tasks. But if the network is too deep for a certain problem the risk of overfitting increases. The reason for this is that the network is complex enough to represent the training data completely and therefore all generality is lost.

Overfitting and Underfitting

When training a neural network only the training dataset is used for adjusting the model weights. After each epoch the validation data is used to check if the adjustments did increase the generality of the model. When the network is too deep it can happen that the models accuracy on the training data still increases while the accuracy on the validation data decreases. This is because the neural network weights are adjusted so precisely on the training data samples that it looses the ability to make more general predictions. This phenomena is called overfitting.

On the opposite a model might not be deep enough to represent the complexity of the training data sufficiently. Then the training loss is not converging to smaller numbers or even diverging. This would be addressed by the phrase underfitting.

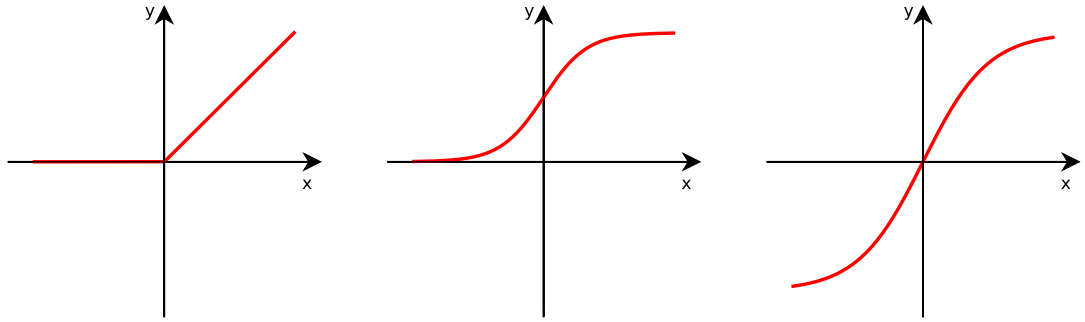
Learning Rate

The factor determining how much the weights of the network are being adjusted is referred to as learning rate. A value of 0 would result in a network that does not adjust its weights at all. A common order of magnitude is around 0.1 to 0.001. The learning rate does not have to be static and can be adjusted during the training. In the beginning usually a higher learning rate is chosen to faster converge the training in the right direction and later the learning rate decreases to make fine adjustments (Bengio, 2012).

Activation Function

After the data went through any layer that actually changes the values within the NN, like the convolution layer or the dense layer, the activation function is applied to this data. The most simple activation function is just a linear function mapping all inputs to the same outputs:

$$f(x) = x. \tag{3.2}$$



(a) ReLU *(b) Sigmoid* *(c) tanh*

Figure 3.4: *The ReLU (left), Sigmoid (center) and the tangens hyperbolicus (right) activation functions in comparison.*

This activation function is commonly used in the last layer to gather the combined results of the previous layers. Within the network a non-linear activation function is used to represent a non-linear relationship between nodes. The problem to be solved is non-linear in the very most use cases of NNs and therefore only a non-linear connection of the nodes is able to produce the requested results. Common functions to use particularly together with convolution layers are the Rectified Linear Unit (ReLU) activation function (Hahnloser et al., 2000), the Sigmoid function (Sibi et al., 2013) or the tangens hyperbolicus (Fig. 3.4). The second one brings the benefit that all input values, no matter how small or large are mapped to values between zero and one which increases the stability of the NN. On the other hand, it reduces the effective range and so can cause results that always look too smooth due to the vanishing gradients inside the network.

Loss Function

In the training phase of the neural network each prediction by the network y has to be compared to the ground truth $\hat{y}$ to evaluate the model after every batch. Dependent on this evaluation the network weights are being adjusted. The loss function $\mathcal{L}$ is the metric for the comparison between y and $\hat{y}$ and should usually be defined to result in zero if y and $\hat{y}$ are the same, so that

$$\mathcal{L}(y, \hat{y}) = 0 \qquad \text{if} \qquad y = \hat{y}. \tag{3.3}$$

Often used loss functions are the Mean Squared Error (MSE) and the Mean Absolute Error (MAE) (Murata et al., 1994), as they are intuitive and already implemented in every machine learning framework.

Hyper Parameter

Parameters which can not be trained by the network and therefore have to be set by the user are called hyper parameters. Examples for these are the batch size, the learning rate or layer specific parameters.

3.3 The Layers and Their Functionality

The art of creating neural networks that accomplish a job to a sufficient accuracy, is to place the right layers in the right order and to tune their parameters in the way that is most advantageous for the problem to be solved. To do this the knowledge about how every single available layer works has to be gained first. In the following paragraphs the most common layers are described. They will be used to create the structure of the CNN to predict clean radar images (Sec. 6.1).

In general, a layer I_l defines how the nodes of the previous layer I_{l-1} are connected to I_l and what kind of operation is done to the values. This means that not every node from I_{l-1} must have a connection to every other node in layer I_l. Furthermore, a layer has a certain size and shape. Size and shape are distinguished as different things. Size are the two dimensions of a tensor that defines its height and width. Shape can be used equal to size or mean the complete dimensional outline of a layer, including its depth. The depth of a layer in a CNN determines the number of convolutions with different kernels to this layer. The word tensor can be used equally to layer but usually when speaking about a tensor its values are addressed. The layer can additionally mean the functions that are applied to the values.

Input Layer

The input layer is the first layer of every NN. Here the shape of the input data has to be provided. The input data will always have at least two dimensions, because one dimension is preserved for the batch size. For training a network with radar images which have a dimension of 360 angle values, 320 range gates and for the training a batch size of 64 is used, then the input shape would be (64, 360, 320, d). The last dimension d changes with the number of filters within the convolution layers of the network. Since the input data has just one dimension and not 3, as it would be in a coloured image for the channels red, green and blue (rgb), the last number initially will be 1 and so the input dimension to the network is (64, 360, 320, 1). A neural network can have more than one input layer and other input layers do not have to be at the beginning of the network.

Output Layer

The output layer usually is the last layer of a NN where the data is returned to the user. While inside the network the data is represented in form of tensors, at the output layer the data gets transformed into an array or other common data format. A neural network can have more than one output layer and they can produce data in different shapes. As an example in the case of radar image processing one output layer could return the processed radar data and another output layer could return only the background noise. It has to be considered though that in a supervised modeling approach for each output layer separate target data needs to be provided during the training. This is because at the output layers the loss function is applied to calculate the accuracy of the training results which is further necessary to adjust the weights of the NN.

Convolution Layer

The center piece of every CNN is the convolution layer, because this layer is where the Model actually learns. For understanding the convolution layer a few key words need to be explained beforehand. First, every convolution layer has at least one kernel. The kernel has a size which is usually way smaller than the tensor it is applied to. It furthermore has a stride width which determines how many values of the incoming tensor are skipped before applying the convolution with the same kernel again. An often used alias for the kernel is the filter. Its size is referred to as receptive field, because it determines to how many nodes of the previous layer a node of the actual convolution layer is connected. The filter count stands for how many convolutions with different filters are applied. The last keyword is the padding. Due to its nature, a convolution always results in a tensor which is smaller than the incoming one. But there is a way to circumvent that by simply filling the edges of the incoming tensor with any number. Generally the number zero is used, because adding zeros does not change the outcome of the convolution. More details on padding are described later in section 6.4.

The so called kernel-convolution is a simple but effective way to filter out features from an array. Let the kernel K be defined as

$$K = \begin{bmatrix} 0 & 1 & -1 \\ 2 & 0 & -2 \\ -1 & 0 & 1 \end{bmatrix}. \tag{3.4}$$

The kernel often is randomly filled with numbers at the beginning of the training and each convolution layer has more then one. Now let the incoming tensor T_{in} be

defined as

$$T_{in} = \begin{bmatrix} 11 & 12 & 18 & 9 & 7 & 19 \\ 9 & 1 & 5 & 4 & 9 & 13 \\ 8 & 16 & 18 & 18 & 9 & 8 \\ 12 & 0 & 6 & 14 & 13 & 17 \\ 11 & 15 & 13 & 10 & 0 & 0 \\ 5 & 5 & 7 & 11 & 17 & 1 \end{bmatrix}. \tag{3.5}$$

The resulting tensor of the convolution is called feature map G. Its values are calculated by summing up all multiplications of each kernel value with the corresponding value in the incoming tensor

$$G(x, y) = (T_{in} * K)(x, y) = \sum_i \sum_j K(j, k) \cdot T_{in}(x - i, y - j). \tag{3.6}$$

Here x and y are the indices of the feature map and i, j are the indices of the kernel. For the example values from 3.4 and 3.5 the resulting feature map G would be

$$G = \begin{bmatrix} 12 & 5 & -15 & -40 \\ -30 & 11 & 20 & 19 \\ 12 & -33 & -18 & -15 \\ -8 & 8 & 37 & 6 \end{bmatrix}. \tag{3.7}$$

The calculation is explained more intuitively in figure 3.5. In the given example the kernel is applied directly, but usually some none linear function is mapped to the resulting value of the convolution, called activation function. If the *tanh* activation function is applied for example, the values of the feature map would be capped between minus one and one:

$$tanh(G) = \begin{bmatrix} 1 & 0.9999 & -1 & -1 \\ -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & -1 \\ -1 & 1 & 1 & 1 \end{bmatrix}. \tag{3.8}$$

Last, there is one more parameter to each convolution called bias. The bias is added to each value even after the activation function was applied and the bias is randomly set when starting the training.

It was said at the beginning of this section that the convolution layer is the place where the learning happens. As mentioned above, the kernel values are filled with random numbers at the beginning but during the training these numbers are adjusted. Therefore, these numbers are called the weights of that neural network.

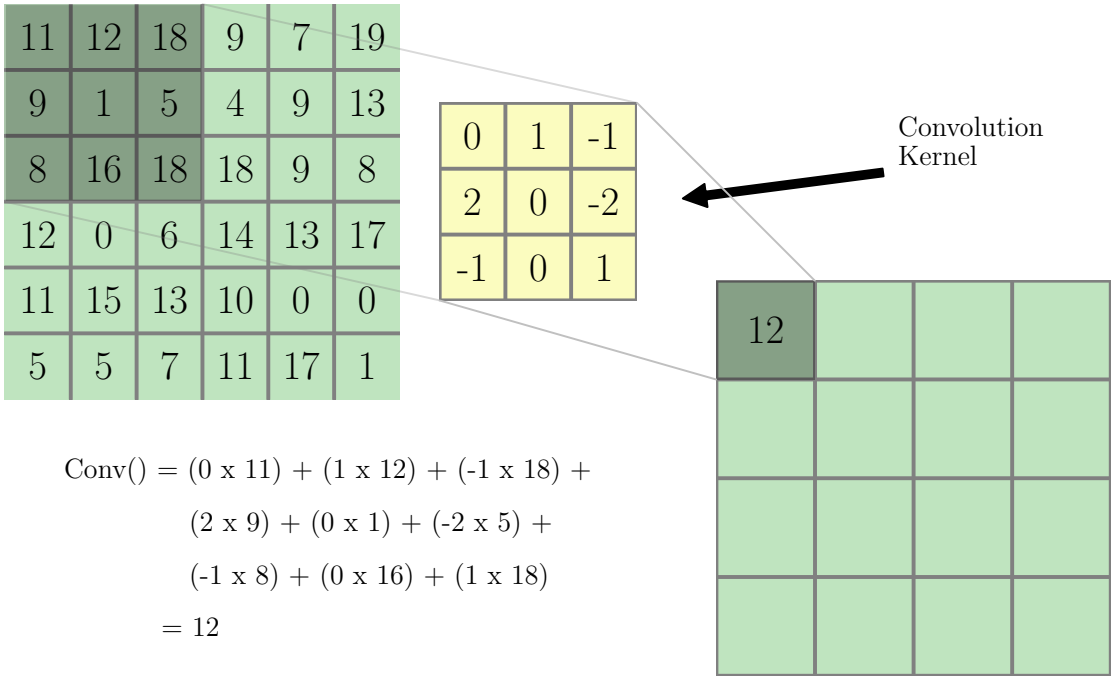


Figure 3.5: *Two-dimensional convolution with a kernel size of (3, 3) and a stride width of one. Since there is no padding applied before the convolution the resulting tensor is smaller than the incoming one. The activation applied here is linear.*

They determine how strong a certain feature map is taken into account, because higher values in the feature map mean higher activation chances for the next convolution. One kernel of a layer could for example be adjusted to enhance only the vertical edges of its inputs and another could be adjusted to enhance the horizontal edges. The more kernels a layer has, the more features can be extracted.

Increasing the filter count raises the number of trainable parameters of the CNN. With this rises the computational effort and the time for training the model. Furthermore, the bias of each feature map is a trainable parameter and will be adjusted during the training phase too.

Pooling Layer

For some network structures it is necessary to change the size of the tensor, meaning that the tensor entering the layer has another shape than the one leaving the layer. An example is the Max-Pooling layer (Zhou and Chellappa, 1988). Max-Pooling reduces the size of the input tensor by taking only the maximum values over a defined field of view (Fig. 3.6). The window size determines the number of values over which the maximum norm is applied and the stride size tells how many steps the window moves before applying the maximum norm again. Due to the reduction of the tensor size the trainable parameters in the next layer are less than in the layer

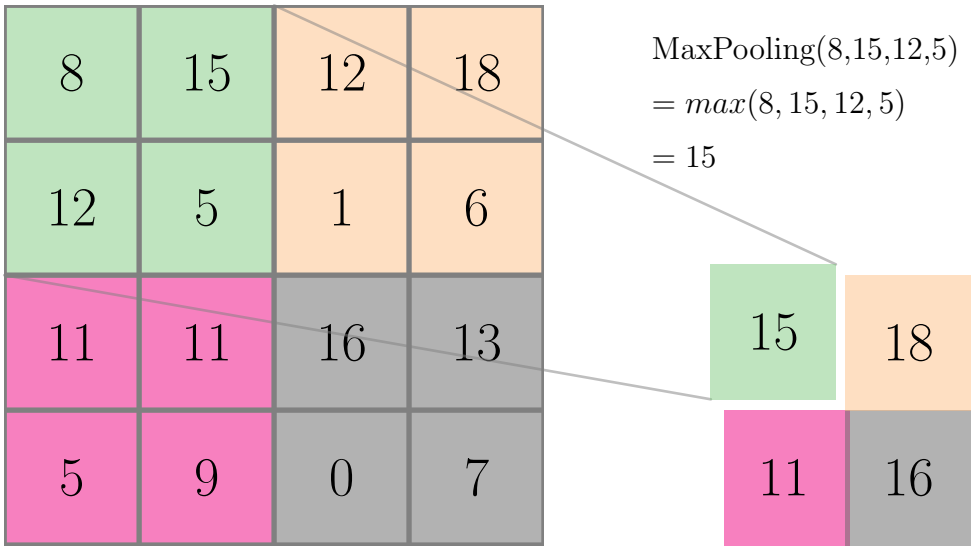


Figure 3.6: *The original array is reduced in size by the Max-Pooling layer. The window size of the Max-Pooling shown is (2, 2) with a stride of 2.*

before. This means that using Max-Pooling layers reduces the computational cost.

Max-Pooling has not only the purpose to reduce the computational effort, it further serves as a signal enhancer. Since just the largest value is taken, only the strongest features get transported to the next layer. Therefore, it is a core component for the feature extraction in CNN structures.

Up-Sampling Layer

Contrary to the Max-Pooling layer, the Up-Sampling layer increases the size of the tensor. Usually nearest neighbor Up-Sampling is used. Each field in the input layer is taken $k \cdot l$ times with k being the up-scaling in the first dimension and l the up-scaling in the second dimension (Fig. 3.7). Unlike Max-Pooling it does not serve any other reason but changing the tensor shape.

Concatenation Layer

Every layer can output to as many layers as the programmer likes. The same output is then send to each of the connecting layers. The other way round on the other hand is not so easy. When many layers have to be joined into a single layer their outputs have to be processed in some way to result in a layer with the same dimensionality as all inputs. This could be done by taking the mean over all incoming connections for example. Another way would be to change the dimensionality of the resulting layer. The concatenation layer takes the output from two other layers and merges them into one. The layers therefore must have the same shape with exception to

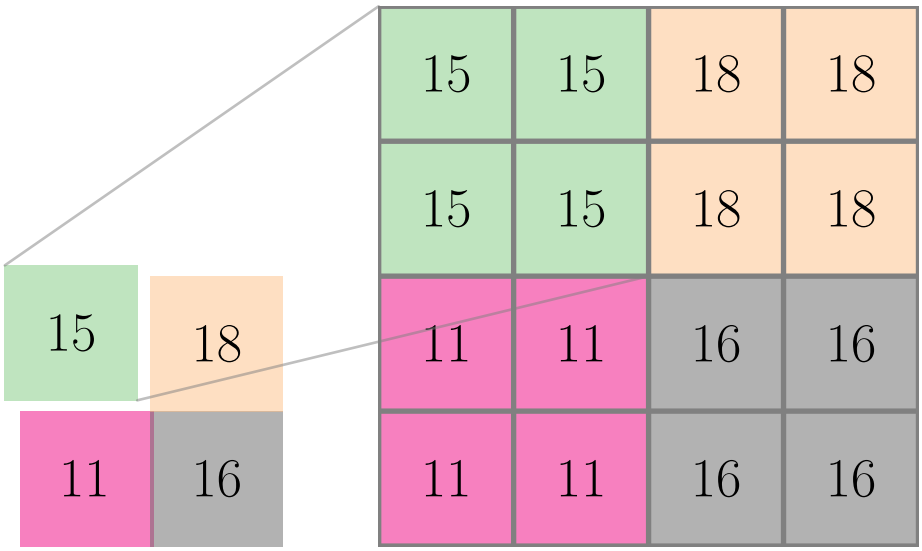


Figure 3.7: *Due to Up-Sampling the array size is artificially increased by spreading the same value over a defined area. In the shown example the window size for the Up-Sampling is (2, 2).*

the dimension at which the layers are added. Usually, this dimension where the incoming layers are glued together is called the filter dimension. The resulting layer has the feature maps of both preceding layers. Therefore, its depth is the sum of the depth from both incoming tensors.

Dropout Layer

A general problem with training neural networks is the possibility of overfitting. In a perfect case scenario the training should be stopped before overfitting starts taking overhand, but in reality it is sometimes hard to detect it during the training phase. Using dropout layers can help to diminish this effect. In such a layer a defined ratio of the incoming connections is not activated. The disabled connections are chosen randomly and change every batch. Due to the random dropouts the remaining connections have to step in. The idea is that the detection of one feature in the data is not done by a single layer or a single path through the network, but rather by many different paths and so the network is less sensitive to overfitting (Srivastava et al., 2014).

Batch Normalization Layer

Mentioned by veterans of deep learning Goodfellow et al. (2016): Batch normalization is *one of the most exciting recent innovations in optimizing deep neural networks.* It was first introduced by Ioffe and Szegedy (2015) and is an approach for

regularizing the values in hidden layers by introducing two new trainable parameters for each hidden layer.

Before any value gets passed to the input layer of a NN it gets normalized either between zero and one, or between minus one and one (Sec. 6.2). This helps a lot for learning since the activation functions are most effective between those values. For example with the *tanh* function the weights can not grow infinitely. But if the NN is very deep and an activation function like *ReLU* is used the weights can grow further in the deeper layers. This is called exploding gradients. Like in any numerical model small perturbations can lead to such an explosion of the model. This means that no useful results can be produced anymore.

The normalization of input data is done using a fix threshold. This means if a radar field with no rain is normalized between minus one and one its values just fill a very small range with a maximum of about -0.5. When a heavy rain event is normalized using the same threshold its maximum might be around 0.9. The change of this input distribution is called covariance shift and it can happen inside the network as well. In the deeper layers it happens due to changing weights and is called internal covariance shift (Goodfellow et al., 2016).

Normalizing helps at the input layer so it could help in the hidden layers as well. This is why Ioffe and Szegedy developed batch normalization which exactly does this by reducing the internal covariance shift. The clue is that for the normalization of the input data the exact conversion from the raw data to the normalized values is known. For every conversion inside the NN this simply has to be known as well. But since the input data, once converted, does not change its range anymore this might not be true for the values inside the hidden layers. Therefore no static parameters for such a conversion can be used. The solution to this are trainable parameters to each conversion. With such, the right normalization can not only reduce amplifying of weights inside of the network but further contribute to learn activations which suit best for the given problem.

Batch normalization is done for each batch ξ separately. Every batch consists of several input fields $\xi = \{x_1, x_2, ..., x_n\}$. The mean μ and variance σ^2 of this batch is calculated with

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{3.9}$$

and

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2. \tag{3.10}$$

With these elements each field x_i in the batch can be normalized to

$$x_i' = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \tag{3.11}$$

where ϵ is a hyper parameter that can be adjusted for each batch normalization layer and usually $0 \leq \epsilon << 1$. The parameter ϵ introduces some noise to each normalization and thus reducing the possibility of overfitting of the NN. Now the trainable parameters come into play, because the normalized values x_i' still have to be scaled and shifted. The batch normalization can then be defined by

$$\tilde{x} = BN_{\gamma,\beta}(x_i) = \gamma \cdot x_i' + \beta. \tag{3.12}$$

The parameters γ and β are trainable by the network and the resulting field $\tilde{x}$ is the normalized value of the batch normalization.

The presented terms, layers and methods are not nearly complete in regard of all the literature about NNs. It is just an insight on the necessary minimum to understand the methodology and results of this thesis. For using NNs on radar images not only the terminology has to be defined. It also has to be checked if the input data is compatible with the chosen network setup.

4 | Testing a Neural Network on Polar Represented Data

For nearly all use scenarios of CNNs the data provided to the CNN is represented in Cartesian coordinates, meaning the data is mapped to a grid which axes are orthogonal and its values are equidistant. For exactly these kind of data the built-in functions of CNN-frameworks are designed. On the other hand some data might not come in Cartesian coordinates, like the data by a weather radar, and thus would have to be remapped before applying the CNN algorithms to it. A remapping usually increases the data size if the transformed image aims to have the same level of precision as the original data, though. This leads to rising computational costs for the CNN. Additional the remapping itself needs computing power. The question therefore is: Can a CNN be applied to data in polar coordinates without significantly loosing accuracy?

To address this question a test scenario is created by running the same neural network on the same dataset. Once the dataset is presented to the neural network in Cartesian coordinates and once in polar coordinates. For this test case the CIFAR10 dataset (Krizhevsky et al., 2009) is used which consists of 60000 images with a resolution of 32 x 32 pixel (Fig. 4.1). To each image there is a corresponding label and each label belongs to one of ten classes: *airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck*, so that there are 6000 images to each label. The aim of the test-CNN built for this scenario is to predict the label from the input images. If the network achieves the same accuracy on the polar dataset as on the Cartesian dataset, the assumption can be made that CNNs can indeed be used on polar gridded input data.

4.1 Creating a Polar CIFAR10 Dataset

To compare the results of a CNN on the same dataset in polar and Cartesian coordinates these datasets first need to be created. As already mentioned, the CIFAR10 dataset will be used once as it is and once converted into polar coordinates to get a

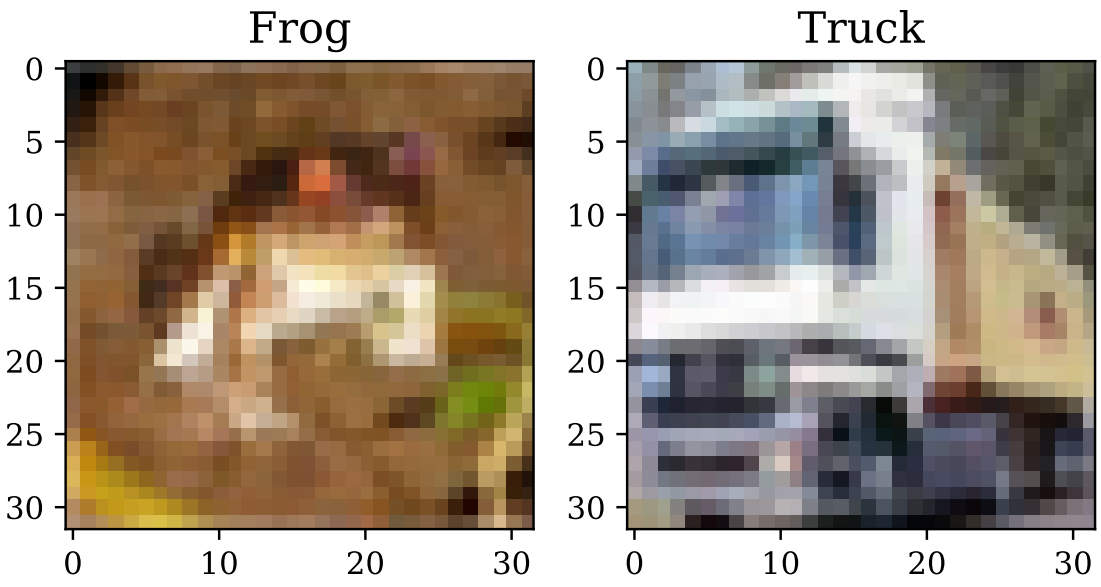


Figure 4.1: *Example images from the CIFAR10 dataset. The images shown are from the classes frog (left) and truck (right).*

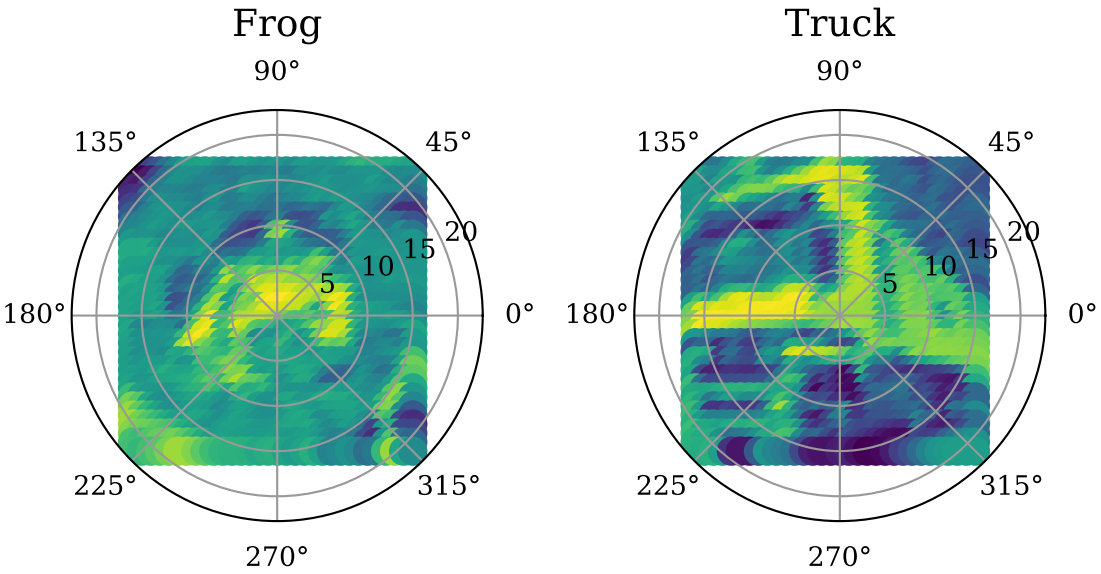


Figure 4.2: *Images from the CIFAR10 dataset in irregular polar coordinates. Displayed are a frog (left) and a truck (right).*

second dataset.

To convert an image from Cartesian to polar coordinates each pixel has to be transformed individually to the new coordinate system. Let x be the horizontal and y be the vertical position of the pixel in the Cartesian image. Further is X the width of the image in pixels and Y its height. First, the coordinate systems center is placed at the image center, so that $\hat{x}$ and $\hat{y}$ are the pixel coordinates relative to the center (Lange, 2016):

$$\hat{x} = x - \frac{X}{2}, \qquad \hat{y} = y - \frac{Y}{2}. \tag{4.1}$$

Then the radius r of the pixel can be calculated with

$$r = \sqrt{\hat{x}^2 + \hat{y}^2} \tag{4.2}$$

and the azimuth angle α of each pixel simply is

$$\alpha = \arctan \frac{\hat{x}}{\hat{y}}. \tag{4.3}$$

The resulting image plotted in polar coordinates (Fig. 4.2) looks the same as the image in Cartesian coordinates (Fig. 4.1). The problem is that the polar coordinate image is not in a regular grid and plotted on a Cartesian grid each pixel would have different distances to each other. Therefore, the polar images need to be regridded to a regular polar grid. This is done by using nearest neighbour interpolation to a 32 x 32 grid, this time meaning 32 range gates and 32 azimuth angles. The result is an images which look way too smooth, especially at the higher radius values (Fig. 4.3). The reason for this is that the density of grid points reduces with the radius on a polar grid. The regridded images have no data at around 90°, because the used nearest neighbor interpolation does not support cyclic boundary conditions which would be necessary to fill these gaps (Sec. 6.4). The data can now be forwarded to a neural network which would see the data as an angle and area preserving field again (Fig. 4.4).

4.2 Results of the CNN on Cartesian and Polar Images

The NN used for this test scenario is a CNN. It consists of four convolution layers, each followed by a Max-Pooling layer and at the and a dense layer collecting the results. The idea for this model structure for predicting labels from the CIFAR10

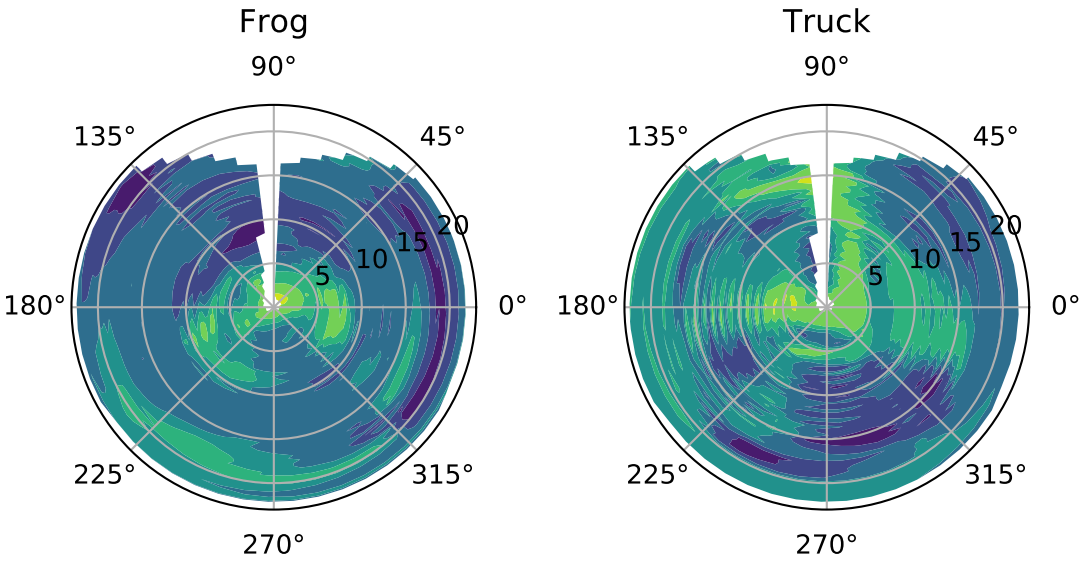


Figure 4.3: *Images from the CIFAR10 dataset in regular polar coordinates. Displayed are a frog (left) and a truck (right).*

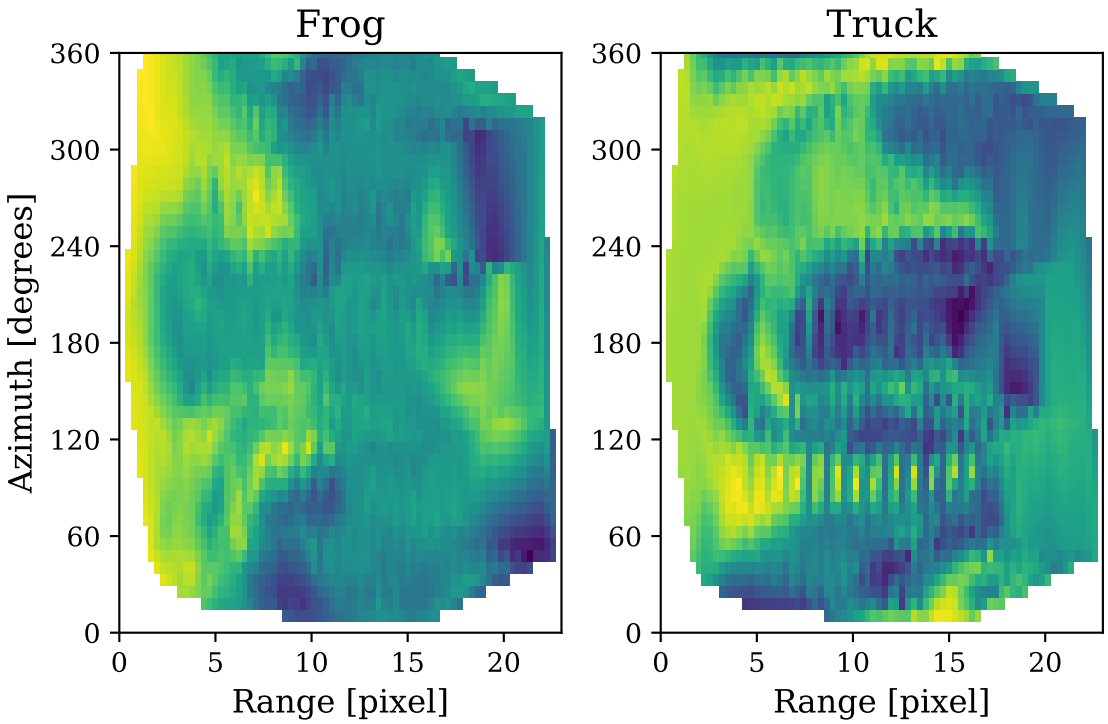


Figure 4.4: *An image of a frog and a truck converted from Cartesian into regular polar coordinates and displayed on a cartesian grid.*

dataset is taken from the Keras homepage (Chollet et al., 2015). For activation the *ReLU* function is used and the loss is calculated by the categorical crossentropy (Zhang and Sabuncu, 2018). Other than manual hyper parameter tuning (Sec. 6.7), no further optimizations were applied to the network.

For the Cartesian represented data a mean accuracy of 0.69 was achieved. The number can be interpreted as the relative accuracy, meaning that 69 % of the images were predicted with the right label. The accuracy of the same network for predicting the labels on polar represented data achieves an accuracy of 0.64. Both scores are comparably low to the 0.93 highly tuned models achieve on the same dataset (Recht et al., 2018). The reason that the produced scores fall far behind what others achieved was already mentioned. The used model is not optimized. If batch normalization and dropout were used together with hyper parameter tuning the results would be more similar. Furthermore, such high accuracies are usually achieved only if the training dataset was increased artificially. This is done for example by rotating the input images randomly (Engstrom et al., 2017).

For this thesis it is more important that the scores on the polar dataset are close to the scores on the Cartesian dataset. The difference of 0.05 can be attributed to two major reasons. The first one is that polar padding was nor implemented on the convolutions of this network, neither for the regridding (Sec. 6.4). This can already be seen by the missing data on the regridded polar images around 90° (Fig. 4.3). Together this results in the network not knowing that the pixels $< 90°$ and the pixels $> 90°$ are next to each other. The second reason for the lower scores on the polar input is that the remapping is adapted to a polar grid with the same size as the Cartesian grid. In terms of the CIFAR10 dataset this results in a polar grid with only 32 range values and 32 angles. Especially at the outer range values this resolution is clearly not large enough to display the images at sufficient detail (Fig. 4.3).

Both of the problems could have been solved with more time to work on this part. The aim of this experiment on the other hand was only to investigate whether the CNN technology can be used on circular input data. Since the compared results are very close, even for an obviously disadvantaged setup for the polar input data, the conclusion can be made that CNNs can indeed be used on polar input data.

5 | Generating Training Datasets

The creation of a dataset containing the input variable X and the output variable y is one of the key elements when working with neural networks. The quality of this dataset determines the highest possible accuracy the network can achieve on real data. If a network is built to predict a cat inside an image, but the input data X does not contain cat images or the output label y does not include the word "Cat", then the network will never be able to fulfill its task. For radar images this concludes in a dataset containing each possible scenario which could occur in real measurements. On the other hand, the NN should be trained to work on the most common cases. The problem is that extreme case scenarios such as tornadoes in a city like Hamburg sometimes do occur, but they are very rare. It is hard to generate a dataset containing such cases and not overfitting the network on it. Therefore in this thesis extreme case scenarios are not being addressed in the training data generation.

Two approaches for generating the training dataset are tested. The first one is proposed by Lehtinen et al. (2018) and is called the Noise2Noise (N2N) method. The second approach uses generated gaussian random fields to create artificial radar images. The big difference between both methods lies in the fact that the N2N approach needs pre-cleaned radar images. In this case pre-cleaned means that the best possible results were already produced by using the Python Package for Weather Radar Processing (pylawr). Preferably, these contain only a small rest of clutter and noise. Then again the radar image generator approach can handle the raw, unfiltered and uncorrected data.

5.1 Noise2Noise Approach

The N2N method was introduced by Lehtinen et al. (2018) and is based on the idea that the mean over enough samples of a random distribution cancels itself out. In the original way this can be used to first take a clean photograph as target y and then add random noise to it. This is done multiple times leading to different input values containing different noise X but all heaving the same clean target y. If a

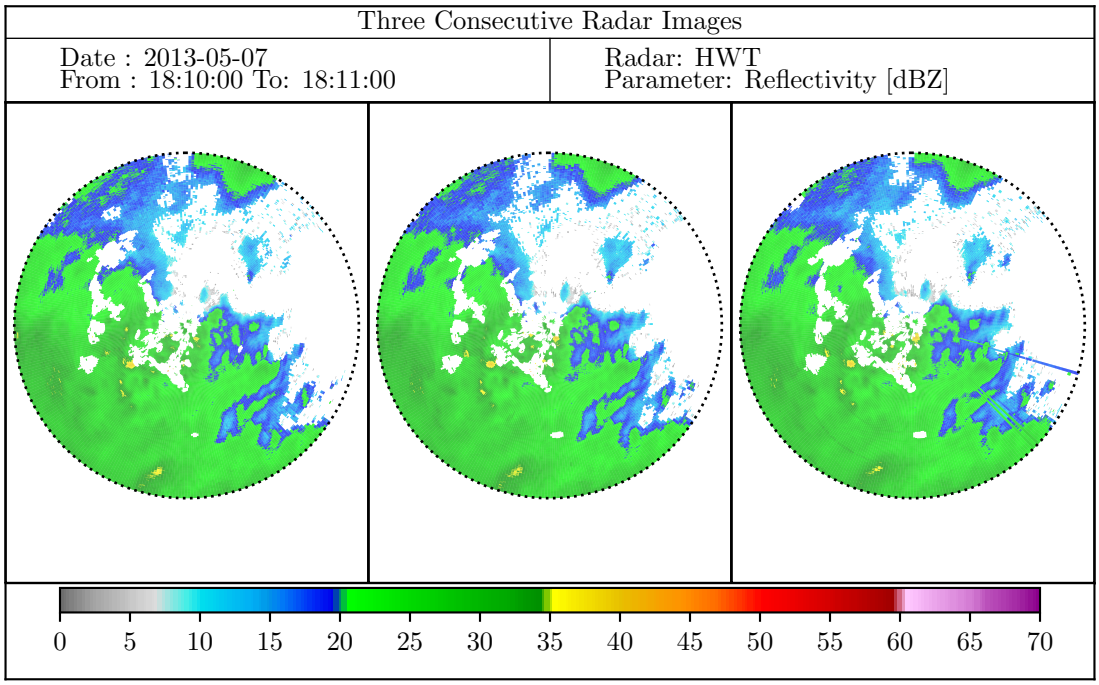


Figure 5.1: *Sequence of three consecutive radar images, which have been cleaned from most noise and clutter by the pylawr.*

dataset contains of 1000 clean images and for each image ten noisy samples X are generated, than the training dataset would have a final size of 10000 X and y pairs. So with the N2N technique the size of the training dataset increases. Since for deep learning the general rule the more data the better applies this is a well-received by-product.

For using the N2N method on radar data already pre-cleaned images are needed. For this pre-processing the pylawr is used. It already removes most of the noise and clutter of the raw radar measurements (Fig. 5.1). The task for the neural network is to learn removing the remaining artefacts.

To use the N2N concept for creating clean targets y for the CNNR a few assumptions have to be made. The first premise is that the scene over three consecutive radar images does only change in a minimal way. Since the temporal resolution of the radar images is 30 seconds this means that over 90 seconds the scene has to stay about the same. The next assumption is that the noise and clutter vary with each radar image. If now the temporal mean is calculated pixel-wise over three consecutive radar images the noise and clutter will be canceled out, because as previously assumed they occur only in one of those three images. On the other hand, the wanted radar signal is assumed to stay about the same over the three images and therefore the mean over the three images represents a noise- and clutter-free image (Fig. 5.2). This mean image is the target y for the CNNR to train on. But a NN

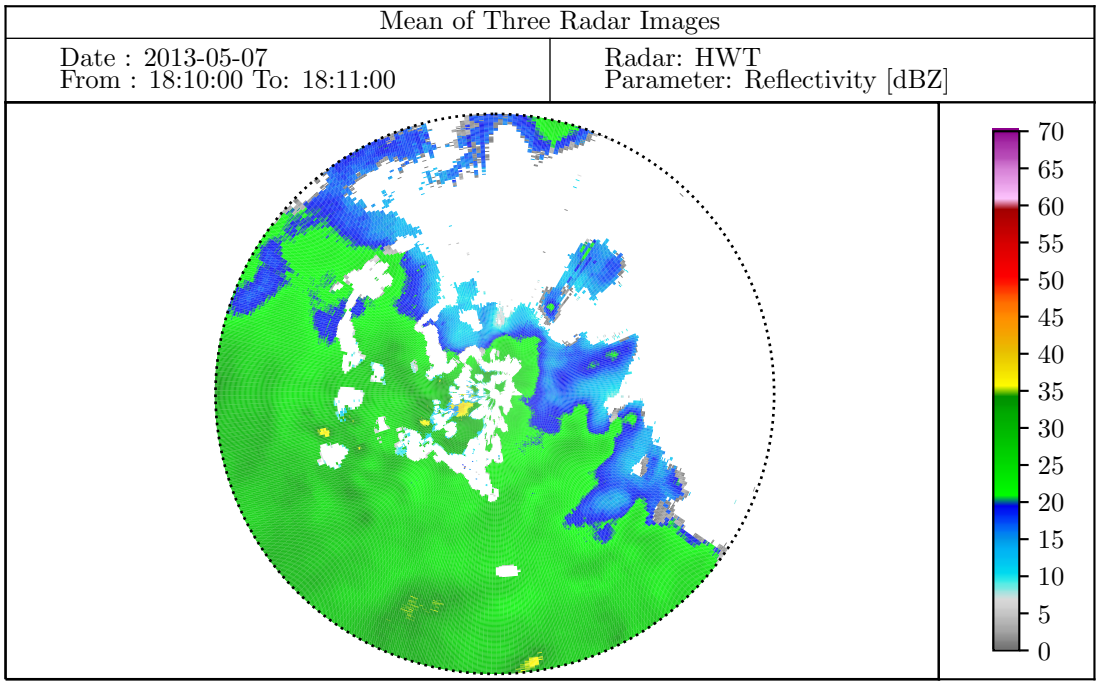


Figure 5.2: *Taking the mean of each pixel by pixel of the three images from Fig. 5.1 results in this radar image.*

in a supervised setup always needs pairs of input data X and targets y to train on. Therefore, X is the radar image in the middle of the used radar images for creating y. As always three consecutive images are taken to create y, X will always be the second image. It is the one theoretically closest to the mean image y.

For the shown series of images (Fig. 5.1) the noise in the upper right part and the spokes of the last radar image are for the greater part removed in the mean image (Fig. 5.2). The question that now can be asked is: Why is a NN still necessary if theoretically with this approach clean images can be produced?

The aim of this method is not to produce a perfect target y for the model to train on. In practise the averaged image will never be completely free of noise and clutter. The NN has to be set up in a way that it does not completely learn to produce images that are the same as y but to produce images that are only close to y. Since the remaining noise and clutter are more randomly distributed in the targets but the wanted signal is not, the NN will only be able to learn the patterns of the wanted signal. To achieve this a high learning rate and batch size are necessary for the model setup.

The N2N approach was implemented with the pylawr package at its version of January 2019. At this time filling gaps of removed clutter was not yet implemented and therefore all targets contain the same clutter gaps in the same spaces. To trick the network into not knowing that these gaps are static, X and y can be rotated

around the center. Due to being in polar coordinates this is fairly easy done by shifting the axis of the azimuth angles.

The benefit of the N2N approach is that it is easy to implement and does not take a lot of computational power. It can be used to train a neural network live in an semi-supervised manner, because the input data can be generated on the fly. On the downside, already pre-processed radar images have to be available and the better those are the higher is the possible quality of the predictions by the CNN.

5.2 Radar Image Generator Approach

As seen in the N2N approach, getting radar images which are clean of noise and clutter and simultaneously are not missing information at areas where just mentioned artefacts were deleted, is not a trivial task. But when it is not possible to clean the Local Area Weather Radar (LAWR) images enough, maybe it is possible to generate artificial clean radar images. These could be taken as targets y and just some clutter and noise had to be added for serving as input X to the CNNR. Even more, not only the rest of the clutter of pre-processed radar images could be added to a generated target y, but all the noise and clutter from the original measurement. Since they are additive to a genuine signal (Doviak et al., 2006) a generated, clean radar image could be transformed into an image which had the same properties as a raw measurement retrospectively.

For creating synthetic radar images containing rain-like structures the assumption is made that reflectivities from rain events in the unit decibel relative to Z (dBZ) can be described by Gaussian fields. Therefore, a random Gaussian field can be generated (Abrahamsen, 1997; Powell et al., 2014) which values θ are scaled to be in the range of radar rain reflectivities $-32.5\,\mathrm{dBZ} \leq \theta \leq 70\,\mathrm{dBZ}$ (Fig. 5.3, upper left). The upper boundary of 70 dBZ was chosen, because it covers most rain events and extreme case scenarios should not be included in the training dataset, as mentioned before.

Now noise and clutter need to be added to the generated rain field for creating the input data X. For this, the raw radar image from a rain-free measurement is taken (Fig. 5.3, lower left), because it will contain all the clutter and noise the CNNR is meant to learn to filter out. In section 2 it was stated that the noise and clutter vary in time. Therefore, for best prediction quality of the network for a specific point in time, the training data should contain artificial rain fields with noise and clutter not older than two month to that certain point. So if predictions should be made for measurements from July 2019 then the training dataset would best contain the rain-free clutter and noise from June 2019. Since noise and clutter are only additive

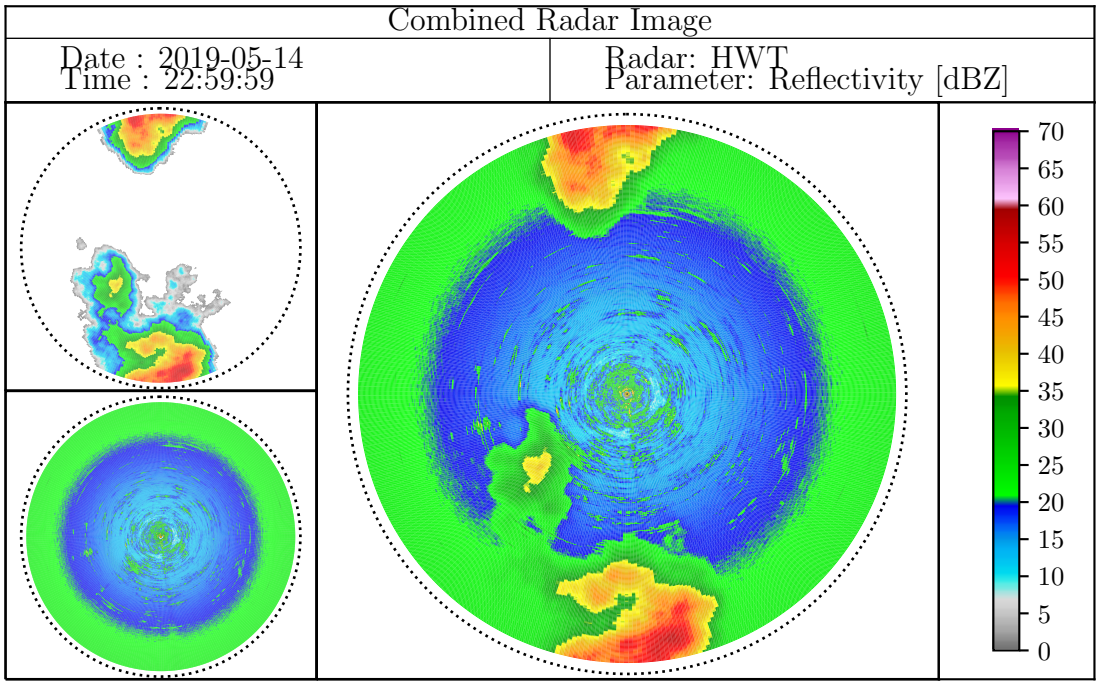


Figure 5.3: *The generated radar image (upper left) and the raw rain-free measurement from the 14th of May 2019 10:59:59 pm (lower left) are combined to serve as input data to the CNNR (right).*

in radar reflectivity units $\mathrm{mm}^6\mathrm{mm}^{-3}$, but the generated radar image and the raw measurement are on hand in dBZ, they have to be transformed to $\mathrm{mm}^6\mathrm{mm}^{-3}$ where they are added. The result has to be retransformed into dBZ again (Fig. 5.3, right).

The radar image generator was developed to not only produce single random radar images, but also to produce consecutive scenes where multiple images are correlated with each other (Fig. 5.4). Developing and moving rain events can be imitated with the generator. The amount of change happening within two consecutive images can be controlled as well. This makes the produced training and testing scenarios more realistic and prevents temporal non-continuous noise and clutter from occurring in the prediction results. The training generator can be set up to produce scenes with a length of n images. If the training dataset should consist of 1000 images and each scene has a length of $n = 100$ images then 10 scenes are included in the dataset to train on. The noise that gets added to each consecutive image of a scene is from consecutive rain-free measurements. This makes the artificial dataset as close to realistic measurements as possible. In contrast to real radar measurements the truth is known as well as the measured quantity.

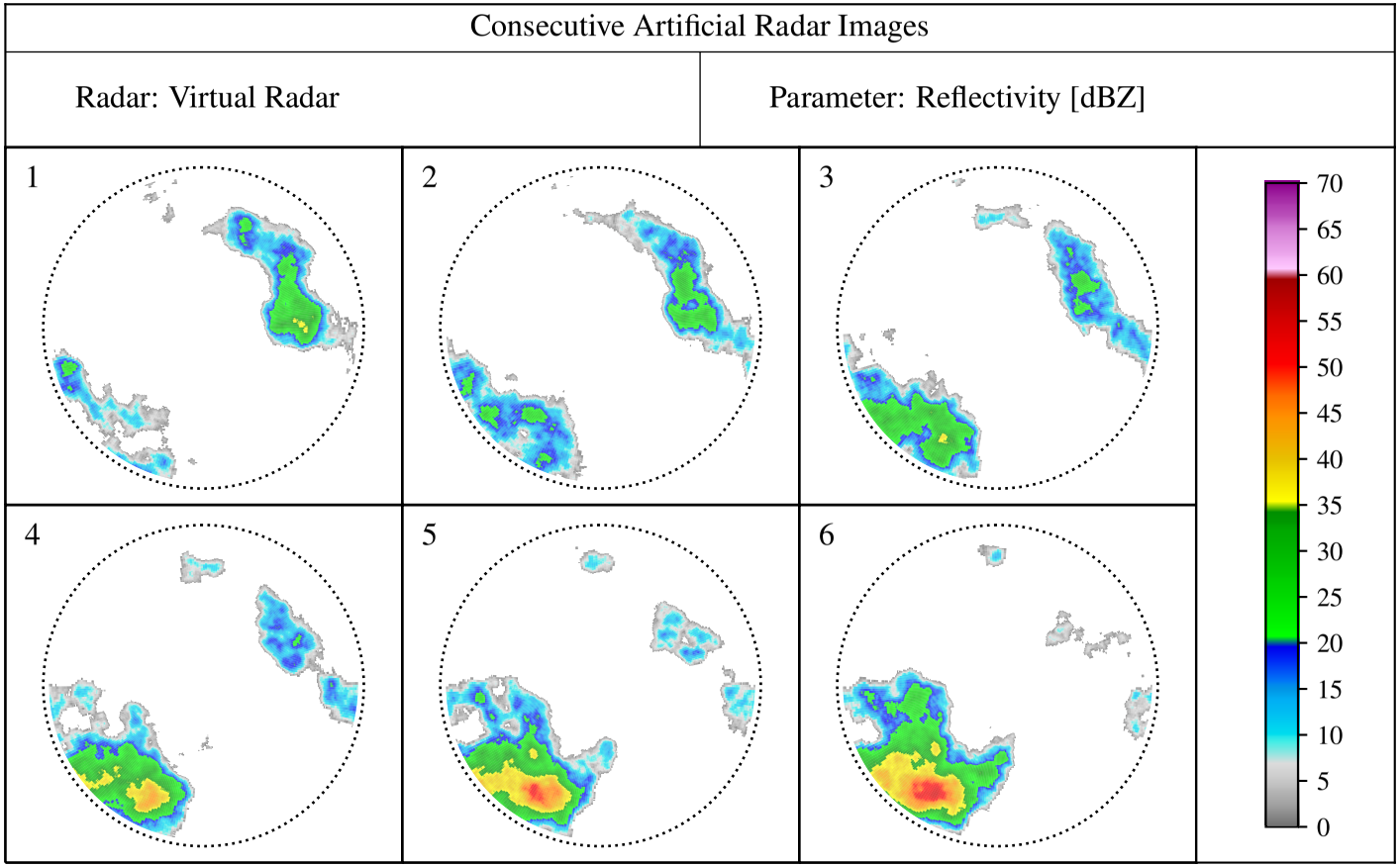


Figure 5.4: *Six consecutive generated images. The scene represents a vanishing precipitation area in the right part of the images. Simultaneously a new new convective area is developing in the lower left part.*

5.3 Conclusion and Decision for one Method

The two tested methods for generating training datasets for the NN differ not only in the targets y they produce, but also in the inputs X that are generated. The N2N approach can only be as good as the pre-processed radar images. But since exactly this pre-processing is not yet good enough it can not be implemented up to now. It would have the advantage that an unsupervised learning setup could be established by fitting the model live on the last three received images. Unsupervised in this case would mean that the model could train forever and always adjust on the last available data. Without creating a new dataset for training, this method could be applied to all radars with identical spatial and temporal resolution.

Since the pre-processed radar images with the pylawr package at its version of January 2019 did not clean the images of enough artefacts to be used as target data for a neural network, this approach was pursued only at the beginning of this research. It was used to start building the network structure and receive first results but with the acquired knowledge from this experiment the idea of another approach to create training data came into play.

The data generator approach has the disadvantage that a new dataset has to be created for every radar. This is mainly because of the static clutter which differs completely for every radar location. Nevertheless, the created dataset results in input data X for the NN which has the same properties as raw radar measurements. The created targets y are free of all clutter and noise. With this dataset it is possible to train a NN for predicting noise- and clutter-free images from raw radar measurements without additional dependencies. All together it makes the data generator approach the best choice.

6 | Using CNNs for Radar Data Processing

After declaring and defining all necessary methods in section 3 and with the knowledge about how the datasets are created (Sec. 5), the structure of the CNNR can be presented. First the definition of the network structure is shown, followed by custom changes to the python frameworks tensorflow (Abadi et al., 2015) and keras (Chollet et al., 2015). Those are the two major python frameworks for deep learning. Implementations which are not custom, but unusual or very specific are explained in this section too. Last but not least, the optimizations of the network will be shown.

6.1 The Layer Structure

To create a NN the ordering of the layers is the essential part. The structure, together with the training data, define the ability of the NN to learn. In section 3.3 already all used layers and their functionalities were described. Now their alignment and adjustment will be presented.

As the underlying blueprint the U-Net Structure (Ronneberger et al., 2015) will be used. This structure defines a CNN of two parts. First being the encoder part, where Max-Pooling is used to downscale the tensor shape while the number of filter increases with each convolution. The other part is the decoder, where Up-Sampling is used to regain the original size of the tensors step by step. There are the same amount of Max-Pooling steps as Up-Sampling steps so that each tensor size exists twice. The clue of the U-Net structure are shortcuts through the network. They are created by concatenating the layer after each Max-Pooling with the layer after each Up-Sampling. Therefore the filter count after the Up-Sampling is the sum of the filter count of the layer before the Up-Sampling and the number of filters which the layer had when first Max-Pooling was applied to its actual size (Fig. 6.1). The input radar image is one dimensional and has 360 azimuth angles and 333 range gates. For the U-Net structure to work, the input dimensions have to be divisible multiple times by two or by three. With the azimuth angles this is possible, but

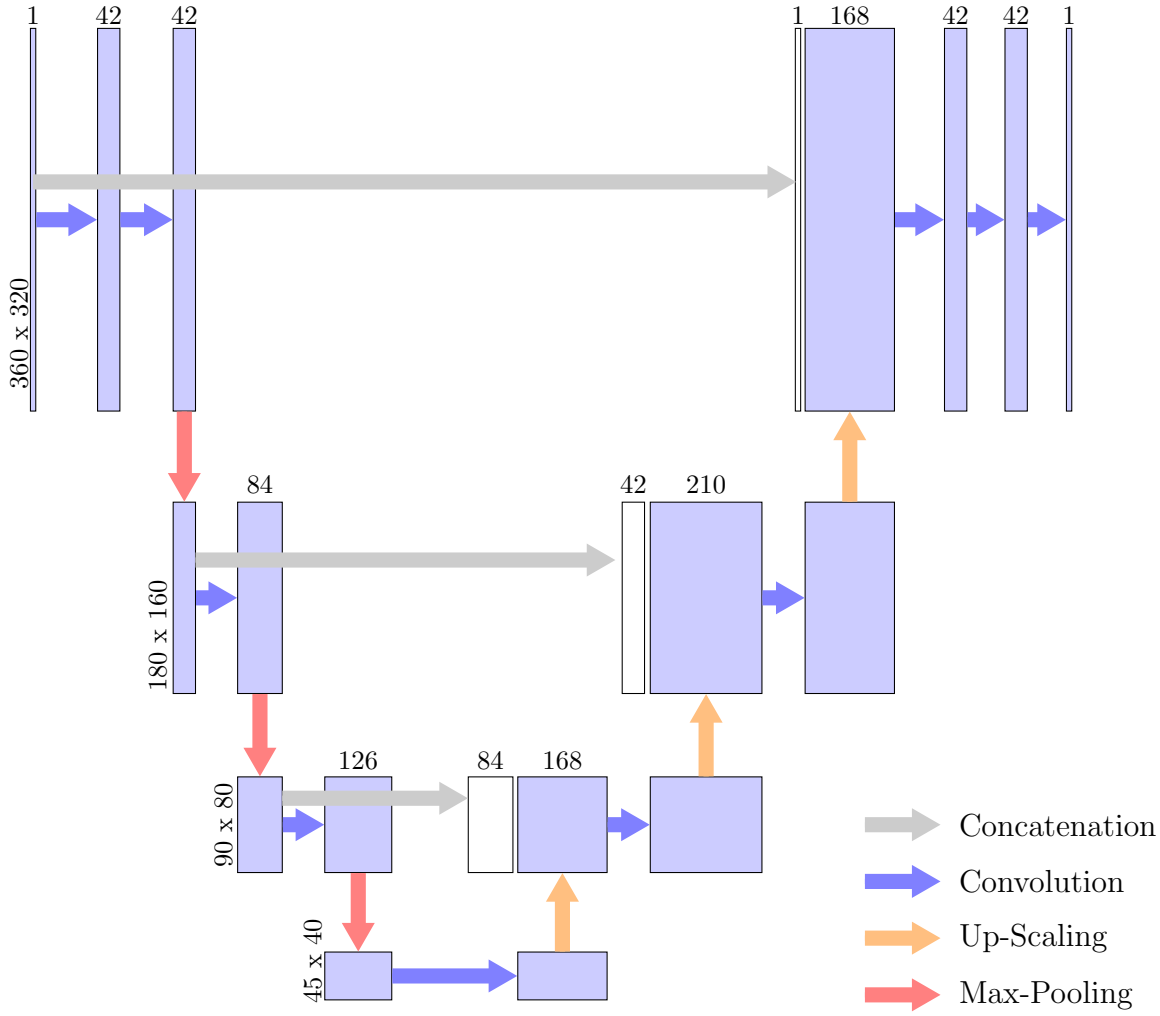


Figure 6.1: *A CNN with layers ordered according to the U-Net structure. The horizontal numbers indicate the number of filters for each level, the vertical numbers indicate the tensor shape. Blue filled areas stand for the tensors and white filled areas indicate a copied tensor. (Adapted from Ronneberger et al., 2015)*

for the range dimension only the first 320 range gates are used (Tab. 6.1). This results in the first tensor having the shape (360, 320, 1). The first convolution uses 42 filters and therefore the tensor depth increases so that the new tensor has the size (360, 320, 42). The next convolution uses the same amount of filters preserving the tensor shape. With the Max-Pooling the first two dimensions of the tensor are halved and it now has the shape (180, 160, 42). This procedure is repeated until the deepest point of the network is reached. The decoder part works in the same manner as the encoder part, but vice versa. At the deepest point of the network the tensors have a size of (45, 40, 168) which then get increased in size by Up-Sampling to (90, 80, 168). The output from the tensor which first had this size gets concatenated to the up-scaled tensor. This means that their filters are added resulting in a shape of (90, 80, 252). In the decoder part convolutions are used with decreasing filter numbers to reduce the computational effort while increasing the tensor size again. At the end, the output of the network has the same shape as the input.

The depth of this presented U-Net is three, because three times a Max-Pooling layer was used until the deepest point of the network was reached. The deepest point is defined as the part of the network, where the size of the tensor is minimal. The python code for creating this structure is programmed in a dynamic way. For this reason the depth of the U-Net can be adjusted with a single parameter from a minimum of two to a maximum of six (Tab. 6.1). The maximum depth of the network is limited by the fact that at some point a division of two or three to the dimensions is not possible anymore. The deeper the network, the more trainable parameters the network has and thus the initial filter count has to be lowered. The initial filter count f_{init} is the number of filters for the first convolution and serves as the multiplier for the filter count of each deeper level. For the encoder part the rule to calculate the filters f_e is

$$f_{e,depth} = f_{init} \cdot depth. \tag{6.1}$$

For the convolution in the deepest part of the network the filter size is once more increased to

$$f_{e,max(depth)} = f_{init} \cdot (max(depth) + 1). \tag{6.2}$$

The decoder part uses the rule for the filter count

$$f_{d,depth} = f_{e,depth+1} + f_{e,depth} - f_{init}. \tag{6.3}$$

Most of the time convolutions do not come alone. In section 3.2 it was said that

Table 6.1: *Tensor shapes at the different depth level of the U-Net. Dependent on the divisibility, the Max-Pooling shape was altered to create deeper networks than it would be possible with a static Max-Pooling size.*

Depth	Tensor size	Used Max-Pooling
1	(360, 320)	
2	(180, 160)	(2, 2)
3	(90, 80)	(2, 2)
4	(45, 40)	(2, 2)
5	(15, 20)	(3, 2)
6	(5, 10)	(3, 2)

after the convolution is done activation functions are used to regularize the output. So when speaking of convolutions actually a whole bunch of layers and methods are involved. Each convolution block starts with a dropout layer followed by the convolution itself. But before the convolution is applied polar padding is used to increase the tensor size beforehand. Therefore, after the convolution the tensor has the same size as before (Sec. 6.4). After the convolution a batch normalization layer is used. The batch normalization regularizes the input to the activation. The activation function than adds some non-linearity to the network. All convolution kernels in this network have the size 3 x 3 and the convolution stride length is one. The Max-Pooling and Up-Sampling both were using variable windows with a stride length of the same value as their window size (Tab. 6.1). The activation function used was the *LeakyReLU* (Sec. 6.3). For the output a linear activation was applied. Two of these techniques are optional implemented, meaning they can be included to the network with a single parameter. Those techniques are dropout and batch normalization. The reason for this is they can lead to improved results for some networks, but in others they might produce less accurate results. Therefore, they are used in the CNNR in this optional way so that in the parameter tuning phase (Sec. 6.7) the enabling can be easily made by the tuning program. A table for rebuilding the exact model that was used for producing the results can be found in the appendix.

For the adjustments of the weights the fast converging optimizer Adam was used (Kingma and Ba, 2014). Faster convergence means that the training loss reaches its minimum earlier than with other optimizers and therefore less epochs are needed to reach the highest possible accuracy. This additionally reduces the computational cost.

The U-Net shape was was chosen for a very specific reason. The aim of the CNNR is to produce output images that have the same size, shape and degree of detail as the input data. The encoder part of the network is for gaining properties

of the input images. The number of convolutions is steadily increased until the deepest point of the network is reached. With the number of filters the complexity of the representation inside the CNN increases too. This way it is possible for the CNN to do such highly complex tasks as removing the noise and clutter. If those results would simply be up-scaled, a blurry output would be the result. The shortcuts through the networks are responsible to place the gained information of the encoding part in the right part of the original data (Fig. 6.1, grey arrows). The Up-Scaling in the decoding part is for reattaining the same size and shape of the input data, but the concatenations are fore regaining the same amount of detail that the input images have.

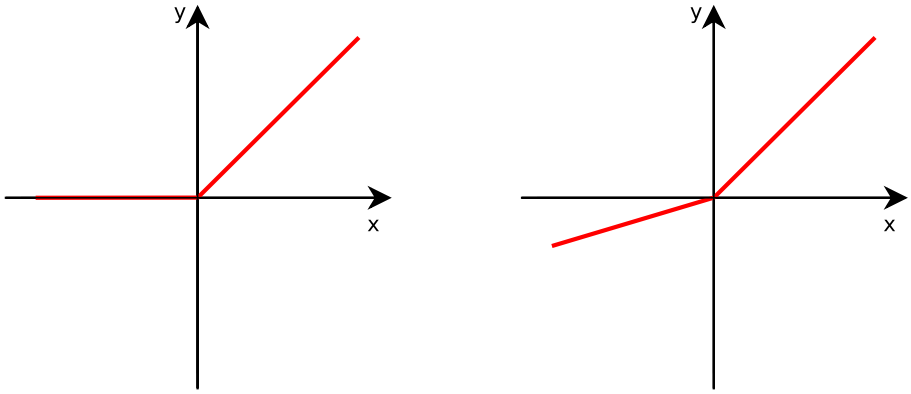
6.2 Normalization of the Input Data

It was already mentioned in the batch normalization layer part of the theory section (3.3) that the input data for the CNN has to be scaled to values between zero and one for the network to ideally adjusted the weights. Weather radar data is already normalized to a certain point, because it was converted from raw reflectivity in $\mathrm{mm}^6\mathrm{mm}^{-3}$ to reflectivity in decibels dBZ which reduces the covariance shift a lot. The radar has a minimum value threshold of -32.5 dBZ which marks rain-free data points. The minimum value of -32.5 dBZ is a convention used by the German Weather Service, among others (Bartels et al., 2004). On the other side of the range, heavy rain events never exceed reflectivities of 100 dBZ. For each input x_i to the CNN the normalized field $\bar{x_i}$ is derived by using 100 dBZ as max and -32.5 dBZ as min, so that

$$\mu = \frac{max + min}{2}, \tag{6.4}$$

$$\bar{x_i} = f(x_i) = \frac{x_i - \mu}{max - min} + 0.5. \tag{6.5}$$

Even if some heavy rain event would exceed 100 dBZ it would only mean that for this rare case the input values would not be perfectly scaled, but the the weights could still be adjusted. On the other hand, the rain field generator used for producing the training data does not output reflectivity fields with a maximum larger than 70 dBZ. Furthermore, the normalization is done to the input values x of the CNN as well to the ground truth y provided during the training and validation phase. The network does not learn to predict y itself, but rather $\bar{y}$. The values of $\bar{y}$ fall in the same range as the scaled inputs. In this case this means $0 \leq \bar{y} \leq 1$. This has to be



(a) ReLU *(b) LeakyReLU*

Figure 6.2: *The ReLU and LeakyReLU activation function in comparison. For all values $x < 0$ the slope of the LeakyReLU activation function is adjustable.*

rescaled to values in dBZ with the inverse function to equation 6.5:

$$\epsilon = (\bar{x}_i - 0.5) \cdot (max - min), \tag{6.6}$$

$$x_i = f^{-1}(\bar{x}_i) = \epsilon + \mu. \tag{6.7}$$

After being rescaled, the predictions $\hat{y}$ by the CNN are comparable to the original ground truth y provided by the data generator and the predictions can be illustrated on a map as radar images.

6.3 LeakyReLU Activation Function

This is not really a custom implementation to the network, but rather a less common configuration. The advantage of the *LeakyReLU* activation function f_{LR} over the more common *ReLU* activation function (Fig. 6.2) is that it adds another hyper parameter which can be adjusted to further improve the model accuracy. This parameter α defines the slope of the function for all values at $x \leq 0$ with the range of $0 \leq \alpha < \infty$:

$$f_{LR}(x) = \begin{cases} x & \text{if } x > 1 \\ x * \alpha & \text{otherwise.} \end{cases} \tag{6.8}$$

For the case that $\alpha = 0$ *LeakyReLU* and *ReLU* are the same. Theoretically a negative α would be possible, but then the node will be activated in any case. The only way to regularize the activation of that node then would be to have only values around zero incoming to that node which would result in a very unstable training.

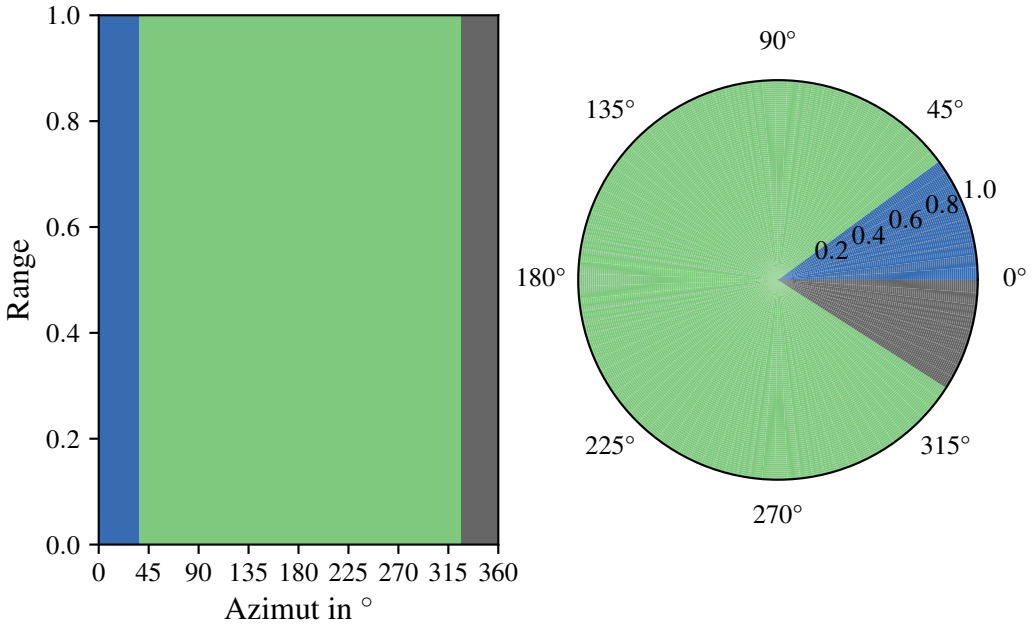


Figure 6.3: *The same data in polar coordinates once plotted on a Cartesian grid (left) and once on a polar grid (right).*

6.4 Polar Padding

As seen in section 3.3 the convolution layer usually reduces the shape of the input data. To counteract this it is possible to pad the tensor before applying the convolution. In most cases this is done by adding zeros to all edges of the incoming tensor. Since the convolution is a dot product of the kernel with the incoming data, adding zeros does not change the outcome. On the contrary it does only work for data in Cartesian coordinates. In polar coordinates the edges of the array do not necessarily represent edges of the data (Fig. 6.3). If the standard zero padding would be applied to the polar data, gaps would be created, because data that in polar coordinates is adjacent would be split by new zero values (Fig. 6.4).

Therefore a padding needs to be applied which has knowledge about the properties of circular data in polar coordinates. In polar coordinates azimuth data can only be mapped to grid values α within the range $0° \leq \alpha \leq 360°$. Values below zero degree i have to be adjusted to grid values $\alpha = 360° - i$ and data mapped to grid values above $360°$ j have to be adjusted to $\alpha = j - 360°$. This is commonly known as cyclic boundary conditions. This condition can only be applied to the azimuth coordinate though, not the range coordinate (Fig. 6.5). The range coordinate still needs to be filled up, but here the zero padding is applied for computation speed.

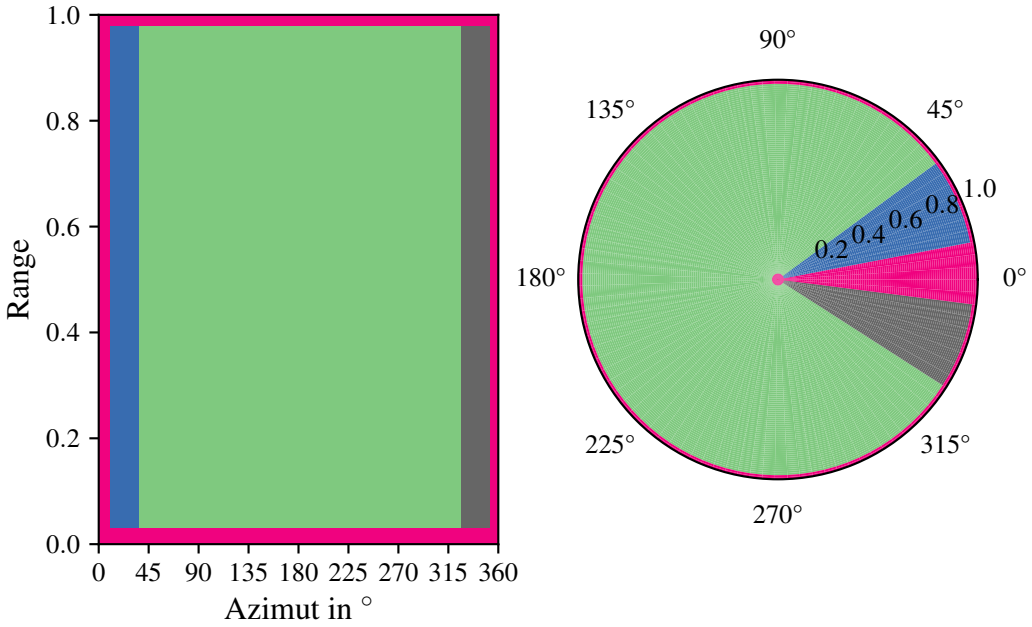


Figure 6.4: *Applying zero padding to polar data (purple area). On a Cartesian grid it looks like only the edges are padded, but when plotted on a polar grid it is noticeable, that it is not only filling out the boundaries.*

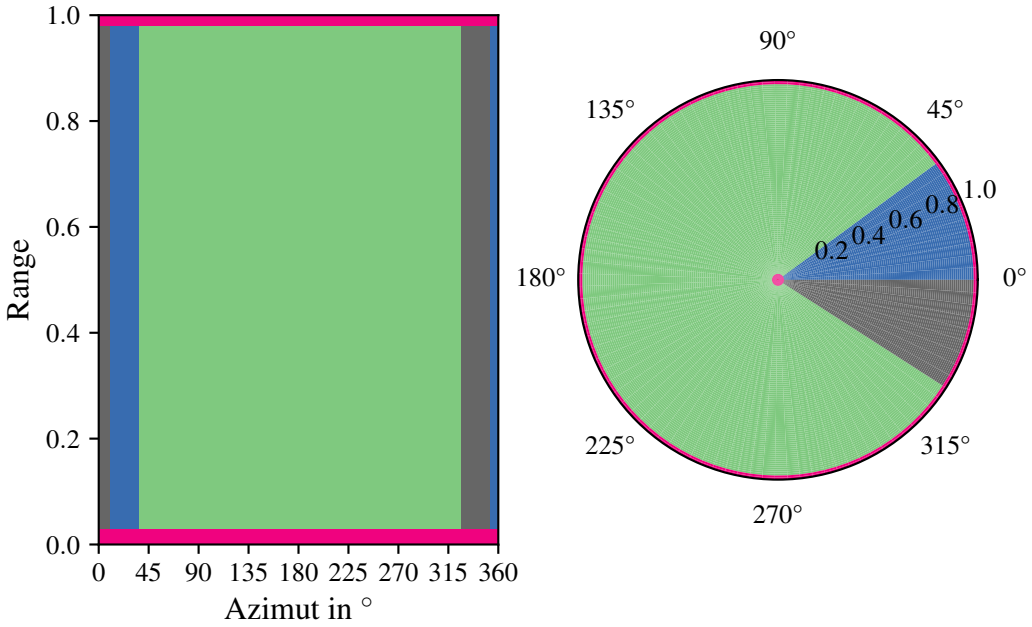


Figure 6.5: *Applying the polar padding shows that the edges in Cartesian coordinates (left) serve as bonding areas in the polar grid (right) where they are not visible anymore.*

6.5 Custom Loss Function

While there are a lot of loss functions which are ready to use out of the box in most machine learning frameworks, they sometimes are not suited for a certain task. This is the case with the CNNR, considering that the network results should not only look correct, but as well contain the original properties a measured rain field has. Therefore, a custom loss function was developed and implemented in the CNNR. In other respects commonly used loss functions for regression networks include, but are not limited to, MSE and MAE.

The problem with the most commonly used MSE is that it can be easily tricked and many different predictions $\hat{y}$ can have the same MSE (Wang and Bovik, 2009). It is defined as

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{N} \sum_{j}^{N} (y_j - \hat{y}_j)^2. \tag{6.9}$$

Cheating the scores is way harder with the Structural Similarity Index Measurement (SSIM), which was particularly developed to measure the similarity of two images (Zhou Wang et al., 2004):

$$\mathrm{SSIM}_{metric}(y_j, \hat{y}_j) = \frac{(2\mu_{y_j}\mu_{\hat{y}_j} + C_1)(2\sigma_{y_j\hat{y}_j} + C_2)}{(\mu_{y_j}^2 + \mu_{\hat{y}_j}^2 + C1)(\sigma_{y_j}^2 + \sigma_{\hat{y}_j}^2 + C2)}. \tag{6.10}$$

Henceforth the index *metric* is suppressed and if not stated otherwise SSIM always refers to SSIM_{metric}. This index compares only a small moving window j of a defined size, with μ_{y_j} being the mean over that window, σ_{y_j} the variance and $\sigma_{y_j\hat{y}_j}$ the covariance. C_1 and C_2 are variables that adjust to the dynamic range L of the images so that

$$\begin{aligned} C_1 &= (k_1 L)^2, \\ C_2 &= (k_2 L)^2, \end{aligned} \tag{6.11}$$

with the default values for $k_1 = 0.01$ and $k_2 = 0.03$. Since for the CNNR all predictions are scaled between zero and one the dynamic range results in $L = 1$. The range of the SSIM is $-1 < \mathrm{SSIM} \leq 1$ with 1 being the best possible value. This is unfavorable for a loss function (Sec. 3.2) so it has to be scaled to become usable for the desired scenario:

$$\mathrm{SSIM}_{loss}(y_j, \hat{y}_j) = \frac{1 - \mathrm{SSIM}_{metric}(y_j, \hat{y}_j)}{2}. \tag{6.12}$$

The division by two clips the values between zero and one which is a common convention. Since the SSIM is taken over a certain moving window the result is

a map with a different SSIM for each section of the image. The neural network unfortunately needs only a single value for each pair of prediction and ground truth as error index. Therefore, the mean is taken over that SSIM map resulting in

$$\mathrm{DSSIM}(y, \hat{y}) = \frac{1}{N} \sum_{j}^{N} \mathrm{SSIM}_{loss}(y_j, \hat{y}_j). \tag{6.13}$$

The SSIM is a widely used metric for evaluating the similarity of two images, but for the radar image case it does not punish the bias hard enough. This means it does not account enough for the predictions $\hat{y}$ of the model being generally too high or too low. With this the need for a Custom Loss (CL) becomes visible. The idea is to use the advantage of the SSIM to evaluate structures and the MSE with the ability to judge the mean intensity difference of y and $\hat{y}$. The most simple way to combine both and still have all requirements to a loss function fulfilled is to just take the euclidean norm and get

$$\mathrm{CL} = \|(\mathrm{MSE}, \mathrm{DSSIM})\| = \sqrt{(\mathrm{MSE}^2 + \mathrm{DSSIM}^2)}. \tag{6.14}$$

The model now has to get the structure as well as the bias right in order to lower the loss. In the unlikely case of a perfect prediction by the model the CL would result in 0. On the other hand, since the MSE has no upper boundary, the CL can grow infinitely for poor predictions. This results in a range for the CL of $0 \leq \mathrm{CL} \leq \infty$.

The CL was implemented in the model and all further results have been achieved using it. Any mentions of the loss function or loss as a value in this thesis refer to the CL if not noted otherwise.

6.6 Clipping

By reason of using the *LeakyReLU* activation function, the intensity of the activation is not capped in any way and can grow infinitely. Therefore the training of the model is sensitive to high values. If at one point in the network the values grow for any reason, a feedback can establish yielding higher and higher values to the next layers. At this point the optimizer usually has no chance in bringing the values down again and the model just got useless. To address this problem many solutions have been derived and most of them can be used next to each other in the same neural network. Some of these methods have already been presented, like the batch normalization layer (Sec. 3.3) or the scaling of values between zero and one before they are run through the network. Another very simple and effective way of keeping the values within the neural network small is to constrain all values larger

than a chosen threshold t to a lower value m after the activation function has been applied (Pascanu et al., 2013), so that

$$\forall \; y > t \;\; : \;\; y = m. \tag{6.15}$$

The constants m and t are hyper parameters of the network and therefore have to be chosen by the user.

6.7 Hyper Parameter Tuning

While the optimizer can only tune the parameters which are trainable, a lot of parameters remain constant and have to be set at the beginning of the training. The reason they are not trainable can have different origins. Some are linked to the structure of the network itself and for others there is simply no metric to train the parameter on. Furthermore, many of these hyper parameters influence each other so that it is not quite simple to find the best fitting set of hyper parameters. In addition each ideal set of hyper parameters is unique to the problem and the neural network.

Goodfellow et al. (2016) present four methods for finding suited hyper parameters: manual tuning, grid search, random search and model-based hyper parameter optimization, ordered from least to most effort. Manual tuning is the most intuitive and means to adjust the parameters by some simple rules, like to lower the learning rate when the training loss is fluctuating around an assumed minimum. It is useful for getting the model up and running but it can take a very long time to find good hyper parameters this way. The grid search method is already way more effective because each parameter is adjusted to a given set of values. The model then trains for a defined number of epochs, gets evaluated and then runs on the next set of values until all possible combinations have been tested. This however only works if the model does not have too many hyper parameters. The number of training runs increases for each parameter due to $combinations = n^p$, with n being the number of predefined values for each parameter and p the number of parameters. Already for a network with only three hyper parameters and three possibilities for each parameter this would mean to train the model 27 times. Assuming for a deep network that a single epoch takes five minutes to train and at least ten epochs are required. Then the hyper parameter tuning for that given configuration would take around 23 hours computation time, not including the time for setting the model up and tearing it down every time a training finishes. The model used for this thesis has eleven adjustable hyper parameters, of which some are conditional, meaning they

Table 6.2: *Adjustable hyper parameters and their restrictions for the random search tuning method.*

ID	hyper parameter	value interval
1	*use clipping*	$\{0, 1\}$
2	*clip value* $[m]$	$[0, 1)$
3	*use dropout*	$\{0, 1\}$
4	*dropout parameter*	$[0, 1)$
5	*use batch normalization (bn)*	$\{0, 1\}$
6	*bn - momentum*	$[0.6, 1)$
7	*bn - epsilon*	$[0.0001, 0.01)$
8	*LeakyReLU parameter* $[\alpha]$	$[0, 1)$
9	*network depth*	$\{x \in \mathbb{N} : 2 \leq x \leq 6\}$
10	*initial filter size*	$\{x \in \mathbb{N} : 16 \leq x < 50\}$
11	*batch size*	$\{x \in \mathbb{N} : 8 \leq x < 50 \wedge x \vert 4\}$

only have to be chosen if another parameter is triggered. Furthermore, for most of the parameters a given set of only three values each would never cover the complete range of possibilities.

The model-based way of enhancing training results by changing the hyper parameters works just like an optimization problem itself. The parameters are adjusted by trying to find a gradient in the hyper parameter space which leads to a lower validation loss after each training. In other words, a new model is built trying to find the best hyper parameters which could as well be an own neural network. This on the other hand leaves a closure problem, because the new neural network would have hyper parameters itself. Furthermore, it would need some time in the beginning to start converging to lower validation losses. In the worst case there might not even be a gradient leading to better results.

With the random search method the neural network is trained with randomly chosen hyper parameters a defined number of epochs and evaluated afterwards. The randomness is restricted by setting up possible and reasonable ranges for each parameter (Bergstra and Bengio, 2012). For some parameters this means using their complete potential range, for others it means to use ranges which tend to result in a high accuracy in the literature. A parameter that has to be restricted is the batch size, for example. Theoretically every value between one and the number of all training samples could be chosen. Usually this amount of data does not fit in the memory of the computer, though. Furthermore, the literate suggests smaller batch sizes for higher generality (Masters and Luschi, 2018).

The implementation of the random search tuning method is easy, does not take much time and it can be run for as long as the user is not pleased with the validation results. Moreover, it does not depend on any interactions between the hyper pa-

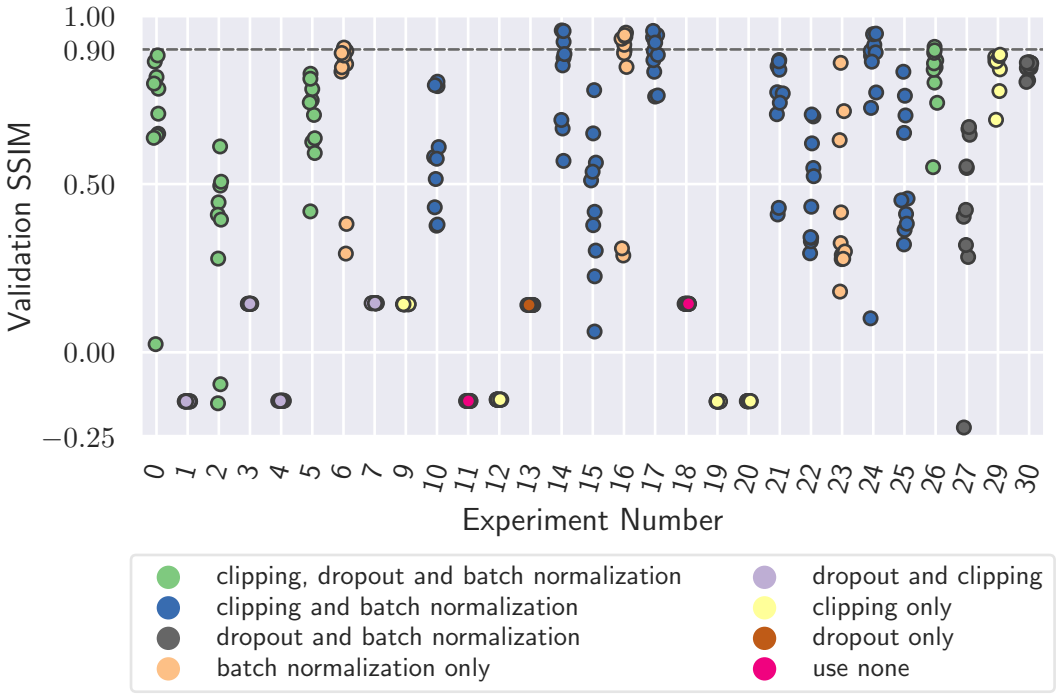


Figure 6.6: *Validation SSIM for the hyper parameter tuning experiments. Experiments which crashed due to memory errors or others are not shown. Each point is the validation score after one epoch and every experiment ran for ten epochs. The color coding gives information about the used optimizations.*

rameters. This makes the random search method superior over the other methods. It is the implemented optimization approach for the CNNR.

As mentioned before there are eleven hyper parameters to be tuned, some of which are natural numbers, some are real numbers and some are binary (Tab. 6.2). The parameters with ID 1, 3 and 5 are binary, meaning that 0 evaluates to $False$ and 1 to $True$. So if *use clipping* = $False$ then the clip value is not used in that model run. The *network depth* parameter defines how many levels the U-Net structure has (Sec. 6.1). The reason that the batch size has to be divisible by four is that the training of the neural network was run on four GPUs. By only allowing batch sizes of multiples of the number of processing units the effectiveness of each unit is maximized (Sec. 3.2).

Not all hyper parameters have been included to the hyper parameter tuning, like the threshold t for the clipping, because they can be adjusted quite well through manual tuning. The clipping threshold has to be $t = 1$, because the input to the neural network was normalized to fall in between zero and one and therefore all values that exceed one should be reset to the clip value m.

Like the training itself, the hyper parameter tuning was run on four GPUs. Each training run consists of 10 epochs and after each epoch the model was validated

using a validation dataset next to the training dataset. Each time when the training was restarted all hyper parameters were drawn from a random uniform distribution within their given range (Tab. 6.2). Moreover a dataset of 5000 generated radar images was used which was split into nine tenths training data and one tenths validation data. Each training starts with a learning rate of 0.01 which than gets reduced by half after every fourth epoch. The monitored quantity for the quality of the model was the validation SSIM (Fig. 6.6). The configurations to those results are shown in table 6.3. Experiments 8 and 28 are missing because the number of trainable parameters was too large, so that the model crashed. Especially the negative SSIMs show that the optimizer was not able with the given parameter set to converge the model to a minimum. Similar to this there are sum runs (1, 3, 4, 7, 9, 11, 12, 13, 18, 19, 20) where the validation SSIM did not change at all, meaning that the model probably got stuck in a local minimum right in the beginning of the training. Furthermore there are eight possible combinations of whether to use batch normalization, clipping and dropout, but only three of these combinations appear in the runs that achieved SSIM > 0.9 and the combinations of batch normalization and clipping or batch normalization alone scored highest. The additional dropout seems to cap the learning, which probably is most noticeable when the dropout parameter is large. Contrarily, the learning with dropout would be more stable than training without dropout if outliers in form of extreme weather scenarios were introduced to the training dataset, but this was not the case for this setup.

When comparing the six experiments which validation SSIM scored higher than 0.9, the only thing in common is that they all use batch normalization (Sec. 3.3). Besides that, the other parameters are spread over the whole range of possible combinations. Especially with the network depth and the initial filter size this is counter-intuitive. Theoretically a larger network should be able to learn more features and intuitively the task of converting raw radar images to clutter- and noise-free radar images is a highly complex one. On the other hand the results speak for themselves: Deeper networks tend not necessarily to result in more accurate predictions. It might be that they would just need larger datasets or longer training in order to reap the full benefits, though.

The takeaway message from the hyper parameter tuning is that for the CNNR batch normalization is essential. Since experiment 14 scored highest with the use of batch normalization and clipping, the hyper parameters from this setting will be used further to investigate and analyze the results. In this chosen experiment not only the score is highest but also the SSIM from the earlier epochs are at around 0.5. This means that a fast convergence within the ten epochs to a maximum of SSIM $= 0.96$ was reached.

Table 6.3: *Random chosen hyper parameters with the best validation SSIM from 10 epochs. For the parameters* `use clipping`, `use batch normalization` *and* `use dropout` *the value 1 evaluates to TRUE and 0 to FALSE.*

Experiment	use clipping	clip value	use dropout	dropout param	use batch normalization	bn momentum	bn epsilon	learning rate	relu alpha	network depth	filter init	validation SSIM
0	1.0	0.674	1.0	0.306	1.0	0.981	0.007	0.01	0.468	4.0	47.0	0.882
1	1.0	0.432	1.0	0.301	0.0	0.651	0.004	0.01	0.751	4.0	34.0	-0.146
2	1.0	0.533	1.0	0.732	1.0	0.811	0.008	0.01	0.937	5.0	35.0	0.612
3	1.0	0.506	1.0	0.74	0.0	0.988	0.007	0.01	0.809	5.0	19.0	0.144
4	1.0	0.23	1.0	0.202	0.0	0.993	0.009	0.01	0.912	5.0	36.0	-0.144
5	1.0	0.094	1.0	0.352	1.0	0.744	0.01	0.01	0.238	4.0	23.0	0.828
6	0.0	0.438	0.0	0.903	1.0	0.623	0.002	0.01	0.479	3.0	43.0	0.905
7	1.0	0.163	1.0	0.759	0.0	0.673	0.009	0.01	0.68	4.0	18.0	0.146
9	1.0	0.703	0.0	0.409	0.0	0.733	0.001	0.01	0.771	2.0	22.0	0.143
10	1.0	0.666	0.0	0.834	1.0	0.835	0.001	0.01	0.835	4.0	20.0	0.804
11	0.0	0.208	0.0	0.229	0.0	0.641	0.001	0.01	0.438	4.0	23.0	-0.144
12	1.0	0.092	0.0	0.411	0.0	0.922	0.004	0.01	0.506	4.0	21.0	-0.141
13	0.0	0.602	1.0	0.749	0.0	0.936	0.006	0.01	0.367	2.0	32.0	0.141
14	1.0	0.853	0.0	0.456	1.0	0.604	0.004	0.01	0.102	5.0	47.0	0.956
15	1.0	0.8	0.0	0.471	1.0	0.999	0.001	0.01	0.333	2.0	19.0	0.779
16	0.0	0.385	0.0	0.357	1.0	0.731	0.001	0.01	0.23	5.0	22.0	0.95
17	1.0	0.422	0.0	0.553	1.0	0.636	0.001	0.01	0.139	5.0	26.0	0.955
18	0.0	0.454	0.0	0.42	0.0	0.921	0.006	0.01	0.181	3.0	30.0	0.144
19	1.0	0.143	0.0	0.018	0.0	0.961	0.01	0.01	0.347	2.0	18.0	-0.146
20	1.0	0.799	0.0	0.91	0.0	0.814	0.006	0.01	0.464	4.0	23.0	-0.145
21	1.0	0.976	0.0	0.201	1.0	0.77	0.001	0.01	0.507	3.0	49.0	0.869
22	1.0	0.934	0.0	0.215	1.0	0.801	0.005	0.01	0.543	2.0	35.0	0.707
23	0.0	0.175	0.0	0.197	1.0	0.639	0.004	0.01	0.377	2.0	36.0	0.86
24	1.0	0.91	0.0	0.467	1.0	0.67	0.006	0.01	0.33	4.0	30.0	0.947
25	1.0	0.338	0.0	0.093	1.0	0.733	0.002	0.01	0.612	3.0	29.0	0.834
26	1.0	0.684	1.0	0.117	1.0	0.75	0.003	0.01	0.333	4.0	22.0	0.908
27	0.0	0.83	1.0	0.499	1.0	0.608	0.003	0.01	0.789	5.0	43.0	0.669
29	1.0	0.349	0.0	0.621	0.0	0.841	0.007	0.01	0.101	2.0	18.0	0.884
30	0.0	0.842	1.0	0.183	1.0	0.923	0.007	0.01	0.142	3.0	19.0	0.862

7 | Results of the CNNR

For inspecting the accuracy of the CNNR a test setup is established where the model gets trained on generator data with noise from rain-free radar images of the first 20 days of July 2019. Evaluated are generator images with noise from rain-free moments of the remaining days of July 2019. For quantifying the accuracy comparisons of the SSIM with a window size of 5 x 5 pixel in polar coordinates are used. The neural network output is compared once against the ground truth of the data generator and once against the analytical noise and clutter removal of the pylawr python package.

During testing an overall mean SSIM of $\mu = 0.98$ was achieved with a standard deviation of $\sigma = 0.009$ over 3000 images. The pylawr package achieves $\mu = 0.96$ on the same dataset with $\sigma = 0.020$ (Fig. 7.1). The testing dataset consists of eight scenes. Each scene starts with 40 rain-free images followed by 360 consecutive artificial radar images. Most of the time both processing algorithms correlate strong with each other, meaning they have their lowest and highest values on the same images. Only for a few scenes the results are anti-correlated, producing little gaps. The CNNR generally achieves higher SSIM values than pylawr. It has to be mentioned that a SSIM of above 0.95 suggests nearly identical images so that the processed data of both algorithms are very close to the artificial truth. The perfect scores of SSIM = 1 for the CNNR at the beginning of each scene show the superiority of the machine learning algorithm for rain-free radar images. Those are processed perfect in most cases, which here translates to: nothing was predicted at all. The pylawr usually leaves some small clutter in those rain-free images, thus resulting in slightly lower scores.

The results show that the model did successfully converge to a small loss in the training phase and the low standard deviation is a sign for a good generalization of the model. Most predicted radar images are close to the created ground truth. For further details on how the results are comparing a closer look on two representative scenes is made. The mean SSIM of the case studies is calculated only for pixels with values greater than 0 dBZ. All other values are clipped to -0.1 dBZ and therefore the SSIM in those areas is not calculated. The SSIM would always be perfect in these areas and therefore raise the mean SSIM to values above 0.95 for both the

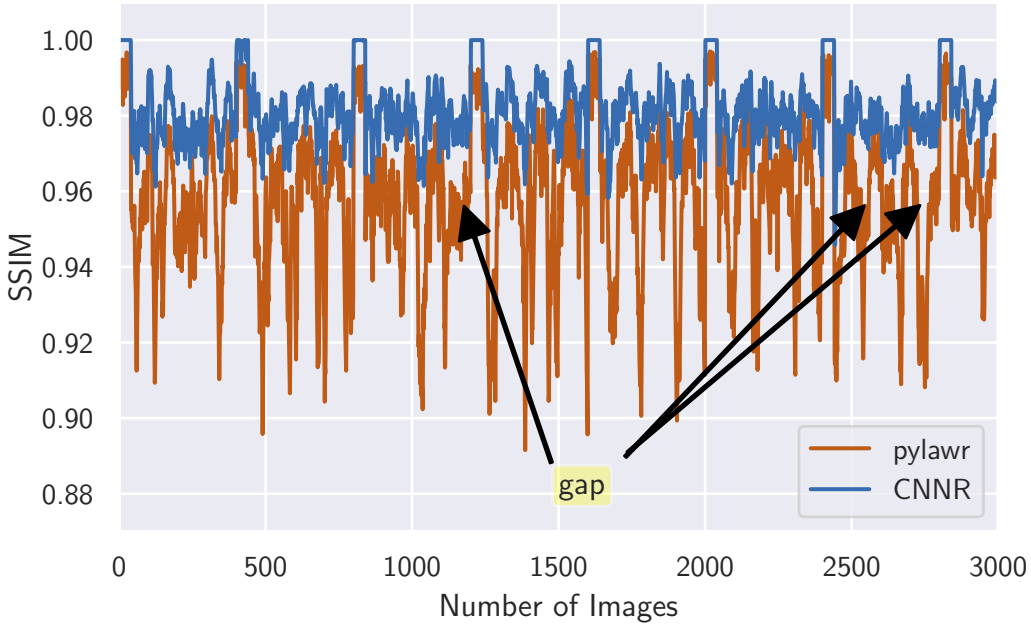


Figure 7.1: *Comparison of the mean SSIM over 3000 images between the pylawr and CNNR.*

CNNR and pylawr. Excluding these values makes the results for the case studies more comparable, because the SSIM is not dependent on the amount of rain-free area in the images anymore. This was only done for the case studies, not the long term comparison (Fig. 7.1).

The resulting images of the first case study look very similar and the SSIM shows this too (Fig. 7.2). Compared is the scene of image number 372 (Fig. 7.1). In the range of 0 dBZ to 5 dBZ the model has the most problems of predicting the correct shape of the reflectivity field. For values larger than 5 dBZ the SSIM is nearly 1. The model is able to predict reflectivities below 0 dBZ, but those values fall below the noise level. Thus, the predicted values below 0 dBZ are not used. Furthermore, the pylawr python package can not handle reflectivities below 0 dBZ and to keep the results comparable they were removed from the predictions of the CNNR as well.

The results of the pylawr on the same image are shown in figure 7.3. As expected the overall shape between the ground truth and the processed image are quite similar. Only in the center some clutter points are not removed and the rain field has some gaps from the clutter removal. Again the problem is with values below 5 dBZ where the python package simply removes most of the signal in order to get rid of the noise. This results in stronger gradients at the edges of the rain field lowering the SSIM there. Above 5 dBZ the reflectivity field is nearly perfectly extracted from the raw data. Only the small rain field in the very northern part is completely erased. A reason for this could be that due to the beam correction the noise of the

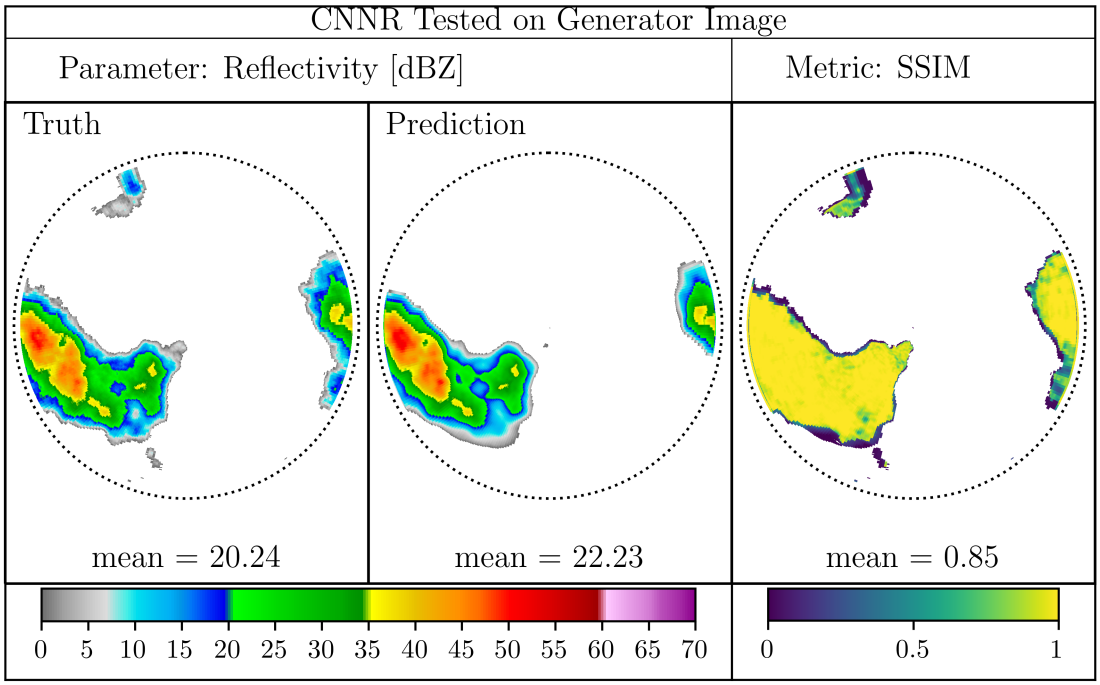


Figure 7.2: *Comparison of a generated radar image (left) to the predicted image (center) by use of the SSIM (right).*

radar gets stronger further away from the center so that this small signal could not be detected and was superimposed by the noise.

While the SSIM is good in quantifying the structures it does not tell a lot about the magnitude of the compared fields. If all values of the predicted reflectivity field in figure 7.2 were added a bias of 100 dBZ the mean SSIM would not fall below 0.82. Therefore, to compare the similarity of the magnitude the prediction is subtracted by the ground truth (Fig. 7.4). The subtraction is done in linear reflectivity $\mathrm{mm^6 mm^{-3}}$, because there are no negatives in the logarithmic scale. Especially for already high reflectivities the NN predicts even higher values. For reflectivities around 20 dBZ to 30 dBZ it underestimates the truth.

With analytic processing some of the original clutter remains in the resulting image (Fig. 7.4, right). This is shown by the circular rings in the subtracted field. Contrary to the remaining clutter, the reflectivities are very close to the truth and this even for high values. The slight green shadow inside the rain fields shows that there is a minimal underestimation of the reflectivities, but this is due to the 103 % noise subtraction mentioned before. Since it falls below 10 $\mathrm{mm^6 mm^{-3}}$ it is not further noteworthy.

A second case study will be made on one of the previous shown gaps (Fig. 7.1) where the pylawr scores fall but the CNNR scores stay constant. Precisely the image number 2756 will be compared from the third and largest gap. The SSIM

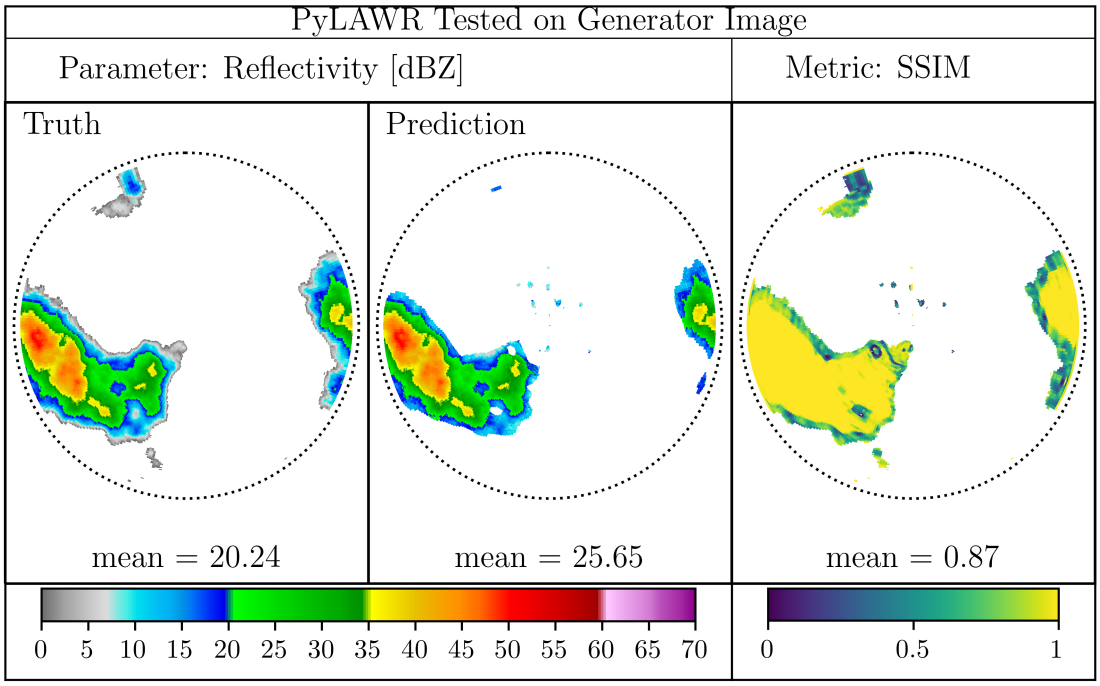


Figure 7.3: *Comparison of a generated radar image (left) to the analytically cleaned image (center) by use of the SSIM (right).*

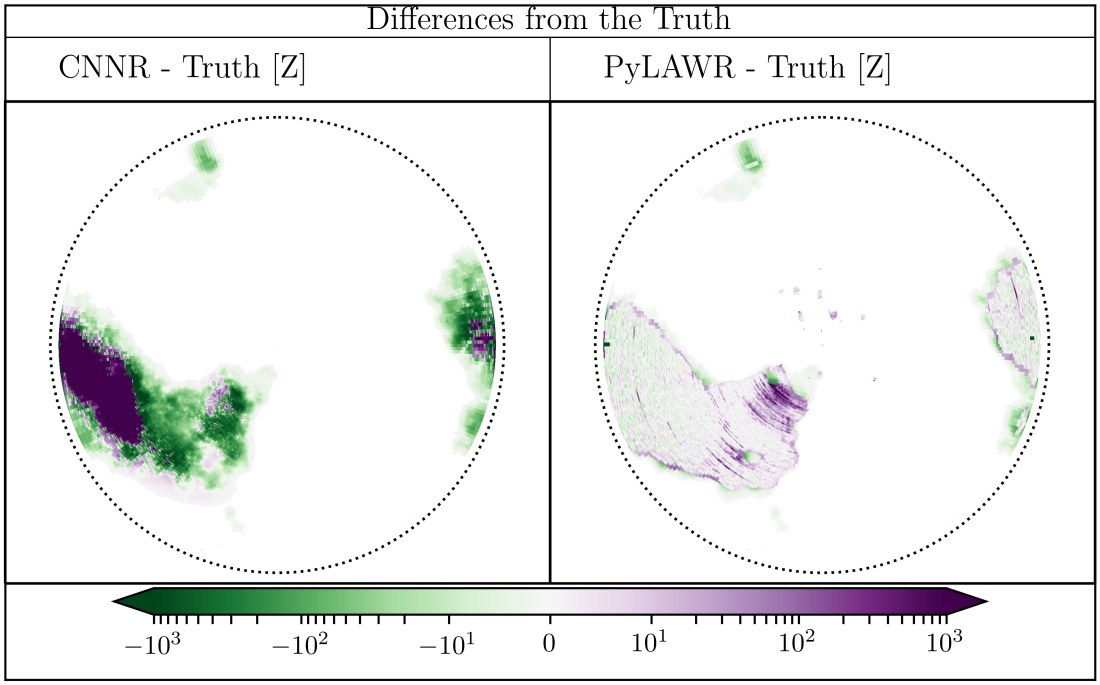


Figure 7.4: *Processed reflectivities from the CNNR (left) and pylawr (right) converted to linear reflectivity Z and then subtracted from the Truth in Z. The color scaling is linear between -10 $mm^6 mm^{-3}$ and 10 $mm^6 mm^{-3}$ and from there continues logarithmic.*

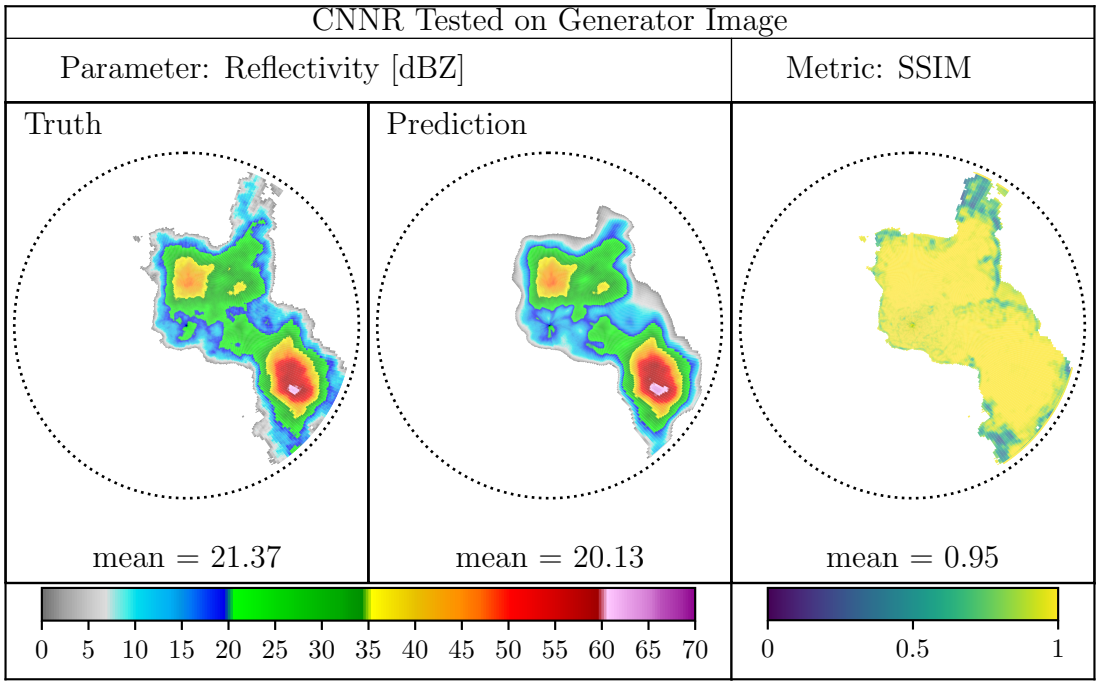


Figure 7.5: *Comparison of a generated radar image (left) to the predicted image (center) by use of the SSIM (right).*

for the CNNR is very good overall (Fig. 7.5). In the upper part some of the lower reflectivities are underestimated, because they are superimposed by noise. Other than this the prediction is very accurate even for the high reflectivities. The center is the part where usually most of the clutter is located. Therefore, small inconsistencies of the predictions with the truth near the center are expected. In this case the predictions do not show a lot of those expected differences. This shows a successful adaption of the network on the training data and a good generalization.

The scores for the pylawr differ a lot for the same radar image (Fig. 7.6). The upper part is completely cut off and the edges are generally way too sharp. The worst part is in the center, though. The filter for static clutter did cut out most of the signal. Since there is not enough rain data around the gaps, they can not be filled by the interpolation. More images from different gaps (Fig. 7.1) where empirically checked. It seems that many small SSIM scores for the pylawr originate from very similar scenarios: The edge of at least one rain field goes straight through the center of the image. Clutter is worst in the center of the radar images. It gets detected and removed by the pylawr, but if there are not enough data points for the nearest neighbour interpolation around, the gaps can not be closed again. This results in lower SSIMs for these images. As shown above (Fig. 7.5) the CNNR does not have this problem around the center and thus its scores are not affected by the same effect.

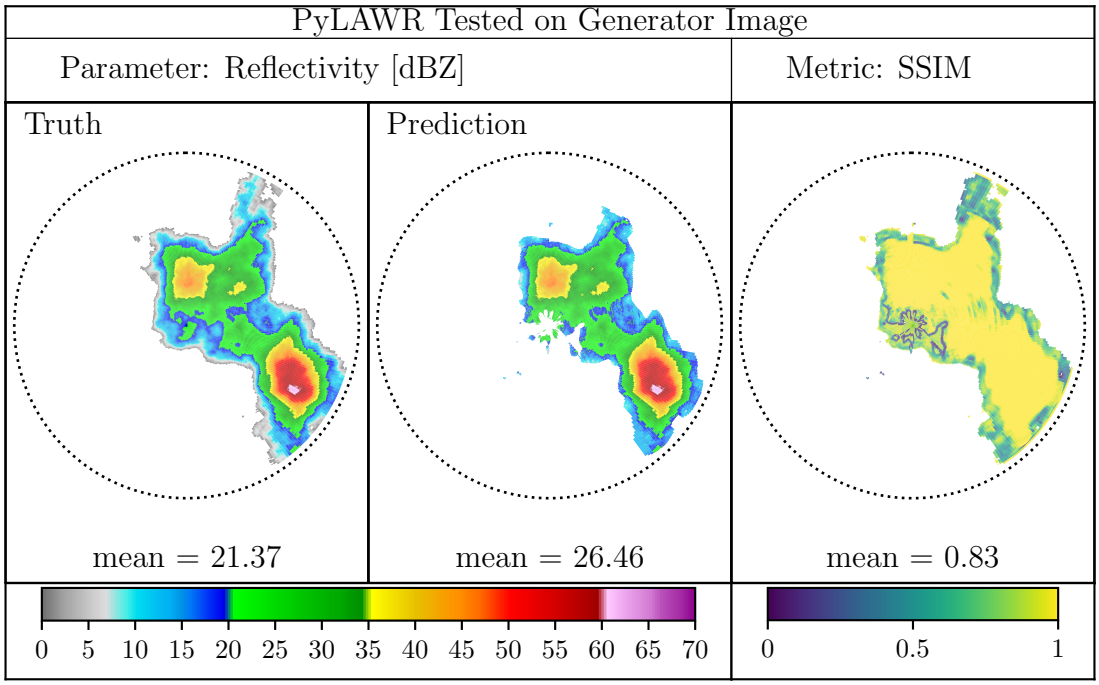


Figure 7.6: *Comparison of a generated radar image (left) to the analytically cleaned image (center) by use of the SSIM (right).*

Last but not least, the CNNR has to be applied to real measurements (Fig. 7.7). Like with the images from the data generator no clutter and noise is left in the processed images. For these images there is no known truth to validate on, but empirical comparisons show that the results are reasonable. The shown scene is a front coming from the south and propagating in the northeastern direction. The images look smooth and show a very high degree on detail. This can be seen good by the smaller areas of higher reflectivity inside the large precipitation field (yellow areas). In all three images there is a tiny gap in the center. This gap does not come from the processing but is inside the measurements and therefore not reconstructable for the CNNR.

Using the pylawr to process the same images the results look only similar on a first glance (Fig. 7.8). The image on the left is the one that is most easy to compare. The main structures are very similar but the CNNR induces some gaps which the processed image of the pylawr does not have. A reason for this could be the interpolation method for clutter of the pylawr which fills up the gaps as well. Another possibility would be that the CNNR mistakes real rain for clutter and therefore produces those gaps. Since there is no real truth, it is not really possible to say which one is better. For the last image on the other hand, a lot of noise and some clutter is left by the pylawr. The rain field extends to the northeastern boundary of the image. For the CNNR the rain field is way smaller. The pylawr

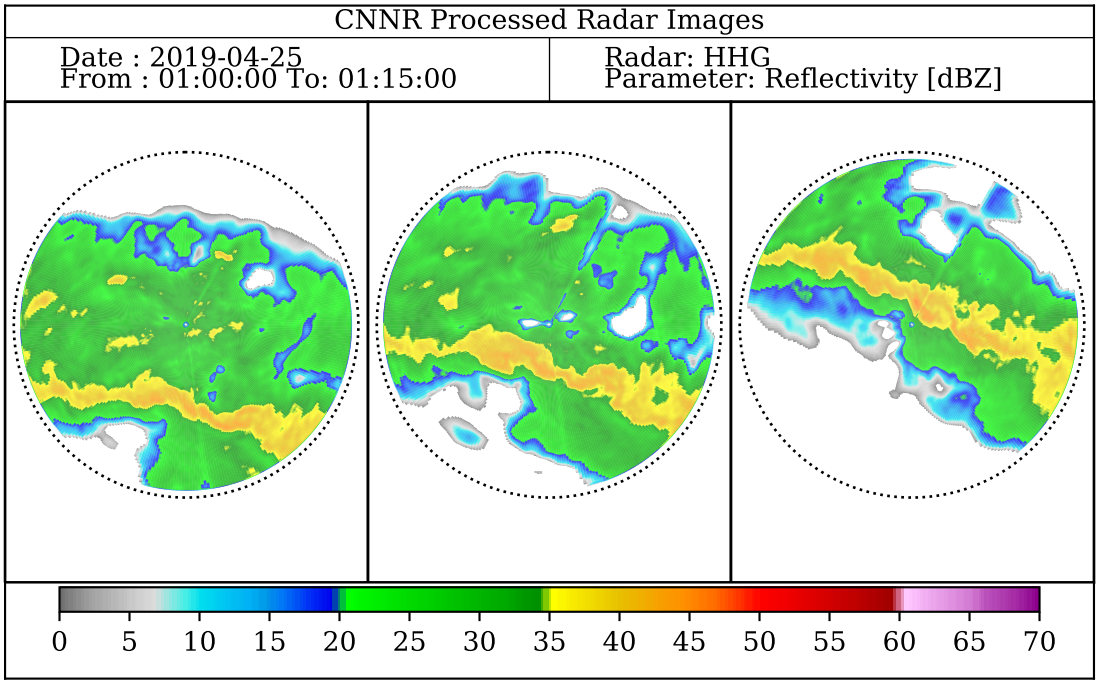


Figure 7.7: *Processed level-1 data with the CNNR. Shown are three images from the frontal passage on the 25th of April 2019.*

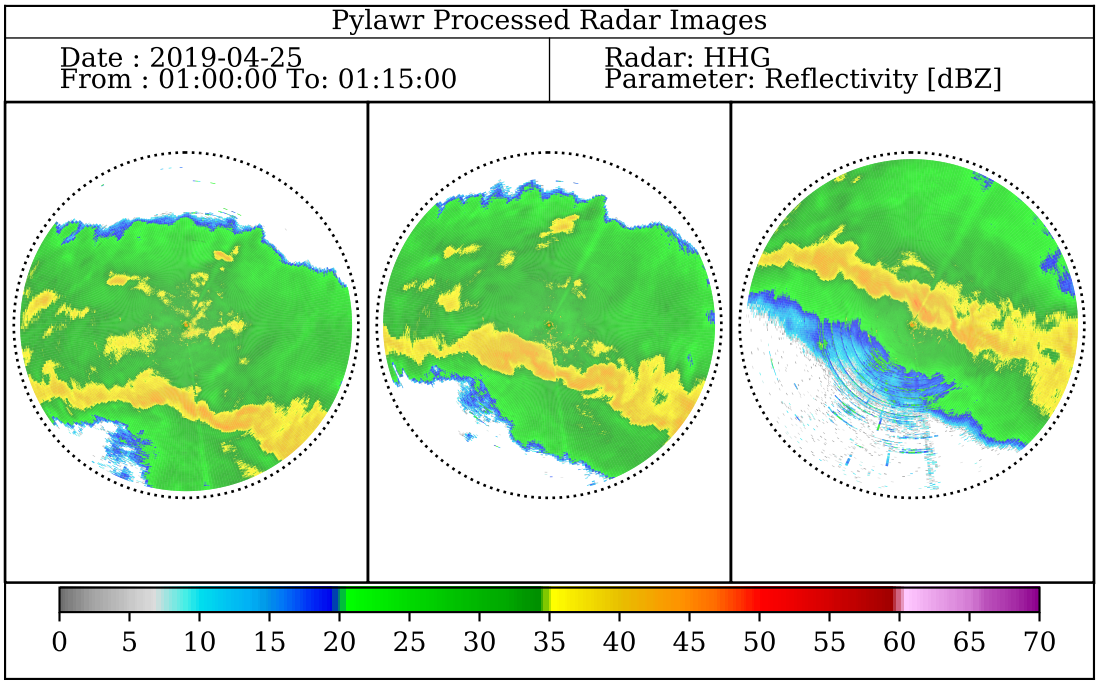


Figure 7.8: *Processed level-1 data with the pylawr. Shown are three images from the frontal passage on the 25th of April 2019.*

is probably closer to reality in this case than the prediction by the CNNR, due to the beam correction. Beam correction accounts for attenuation of the signal. In the third image the strongest precipitation part lies between the radar (center of the image) and the northeastern edge. Since beam correction is not implemented in the CNNR it does not account for the attenuation and therefore no rain is predicted in the area behind the strong precipitation.

None of the processing methods was able to remove the two spikes occurring in the measurements. One going from the center to the northeast and the second from the center to the south. They can be seen in both results as areas with a little less reflectivity. The pylawr interpolates the missing values at the center and therefore induces a little area of higher reflectivity than its surroundings. The CNNR leaves that spot blank. Which of these solutions is better can not be said as both are clearly not true.

Taken all together the results on real data differ a lot more than the results on the data generator. Which of the results is closer to the truth can not be said though, because the the truth for this data is not known. It seams that the pylawr leaves a lot more clutter and noise in the images but the CNNR might delete some of the data in order to remove all the clutter.

8 | Conclusions and Prospects

At the beginning of this thesis it was stated that especially in the meteorological science sector neural networks are not really prevalent. The results of this thesis on the other hand show that there is a lot of potential for NNs in atmospheric sciences. Furthermore, the presented network and the shown results are not even at its possible maximum. For the temporal limitation of this thesis the hyper parameter tuning was stopped prematurely. With the remaining time the presented results were produced. It was further shown that CNNs are indeed capable of processing polar gridded input data.

Besides the already mentioned accomplishments, two further goals were achieved. First, a data generator was derived which proposed reflectivity fields that have the same properties as measured reflectivities by a weather radar. Second, a neural network was developed with the aim to remove noise and clutter from radar measurements without any further information or restructuring of the original data.

The first goal can be verified by two independent incidents. On the one hand side the model did converge during the training and afterwards was applied to real data where it showed decent results similar to the analytic processing. This could only be achieved if the training data had comparable properties to the measured data. Furthermore, the analytical radar processing software pylawr was used on the created rain fields. The software is highly optimized to work on real radar data. Since the results, when applied to the generated reflectivities, look the same as when applied to real measurements, it can again be confirmed that the radar reflectivity generator produces realistic reflectivity fields.

Whether the second goal was fulfilled can not be answered as easy. A neural network was developed which can process raw radar measurements resulting in clutter- and noise-free radar images. To train that network a lot of computational resources were required and a large amount of storage is necessary to save the training data. On the other hand, once trained, the network can process about a 100 images within a second on a decent GPU, or the CNNR can be deployed on a raspberry pi or a smartphone. The smaller the computing power, the longer it takes for an image to be processed, though. Nevertheless, the trained network needs way less computational

resources to process a radar image than the pylawr python package.

There are some restrictions to the CNNR that need to be mentioned. Due to the scaling of the input values the network is not capable of predicting reflectivities larger than 100 dBZ, as this is set as the absolute maximum. Furthermore, the neural network sees the input data as an image. It considers for the value range from 0 dBZ to 10 dBZ the same difference, as for the range 50 dBZ to 60 dBZ. That means it tries to lower the error for every target alike which results in higher absolute errors for high reflectivities, because of the logarithmic nature of the input data. This problem can not simply be solved by converting the input data first into linear reflectivity. Then the input reflectivities could not be approximated by a Gaussian distribution anymore, which is a requirement for training a neural network. Another restriction is the shape of the input data. Due to the models U-Net architecture only multiple times completely divisible numbers can be used as dimension sizes. For the used radar this means that only 320 of the 333 available range gates can be processed. Each range gate averages the reflectivities over a distance of 60 m, so that the 13 range gates that can not be used result in a 780 m shorter field of view.

One big advantage of the CNNR over analytical processing is the absence of physical assumptions. For analytical processing the height of the noise has to be assumed or that static clutter is really not moving, to name a few. The neural network learns this by adjusting the weights of its layers and dependent of the input data it was trained on it can transfer this learning, so that all non-meteorological echoes are the same to the network.

To put it in a nutshell, the CNNR is a good addition to analytical processing. The operational processing might not be perfect at the edges of rain fields but the absolute difference is superior to the CNNR. When it comes to only displaying data to the public on the other hand, the CNNR might be preferable because the processed fields look smoother, no clutter is left, and processed data is available near real time.

The full potential of processing radar data with neural networks would be exploit if combined with analytic pre-processing. The pylawr python package could do the first noise removal, so that the network only had to remove the clutter and fill the holes from the clutter removal. Another combination could be to use the prediction of the CNNR as a mask where rain is and only apply the pylawr in these areas. Furthermore, transfer learning could be used to shorten the training process of the neural network. Transfer learning means that a pre-trained model is trained again for a few epochs on a similar problem. A base model could be trained very general on the radar processing task with clutter and noise background examples from all the years of the radars existence and then for each month only the adjustments had

to be trained. All of the mentioned improvements can be implemented to the here presented CNNR as it is written object orientated and therefore easy to expand without reinventing the wheel.

At the end, it still has to be answered whether it is possible to use neural networks for radar data processing: For single polarized radars, where reflectivity is the only measured parameter, the neural network is superior to the operational processing. It does not rest upon physical assumptions. If properly set up and trained it can represent the truth more accurate than the compared operational processing.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. (2015), 'TensorFlow: Large-scale machine learning on heterogeneous systems'. Software available from tensorflow.org.
URL: *http://tensorflow.org/*

Abrahamsen, P. (1997), 'A review of gaussian random fields and correlation functions'.

Ahmed, E., Jones, M. and Marks, T. K. (2015), An improved deep learning architecture for person re-identification, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 3908–3916.

Bartels, H., Weigl, E., Reich, T., Lang, P., Wagner, A., Kohler, O., Gerlach, N. et al. (2004), Projekt radolan–routineverfahren zur online-aneichung der radarniederschlagsdaten mit hilfe von automatischen bodenniederschlagsstationen (ombrometer), Technical report, Deutscher Wetterdienst, Hydrometeorologie.
URL: *https://www.dwd.de/DE/leistungen/radolan/radolan_info/abschlussbericht_pdf.html*

Bengio, Y. (2012), 'Practical recommendations for gradient-based training of deep architectures', *arXiv:1206.5533* .

Bergstra, J. and Bengio, Y. (2012), 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research* **13**(Feb), 281–305.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J. et al. (2016), 'End to end learning for self-driving cars', *arXiv preprint arXiv:1604.07316* .

Chollet, F. et al. (2015), 'Keras'.
URL: *https://keras.io*

Collobert, R. and Weston, J. (2008), A unified architecture for natural language processing: Deep neural networks with multitask learning, *in* 'Proceedings of the 25th international conference on Machine learning', ACM, pp. 160–167.

Doviak, R. J. et al. (2006), *Doppler radar and weather observations*, Courier Corporation.

Engstrom, L., Tran, B., Tsipras, D., Schmidt, L. and Madry, A. (2017), 'A rotation and a translation suffice: Fooling cnns with simple transformations', *arXiv preprint arXiv:1712.02779* .

Fergus, R., Fei-Fei, L., Perona, P. and Zisserman, A. (2005), 'Learning object categories from google's image search'.

Glorot, X. and Bengio, Y. (2010), Understanding the difficulty of training deep feedforward neural networks, *in* 'Proceedings of the thirteenth international conference on artificial intelligence and statistics', pp. 249–256.

Goodfellow, I., Bengio, Y. and Courville, A. (2016), *Deep Learning*, MIT Press.
URL: *http://www.deeplearningbook.org*

Goyal, S. and Benjamin, P. (2014), 'Object recognition using deep neural networks: A survey', *arXiv preprint arXiv:1412.3684* .

Gunn, S. R. et al. (1998), 'Support vector machines for classification and regression', *ISIS technical report* **14**(1), 5–16.

Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J. and Seung, H. S. (2000), 'Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit', *Nature* **405**(6789), 947.

Ioffe, S. and Szegedy, C. (2015), 'Batch normalization: Accelerating deep network training by reducing internal covariate shift', *arXiv:1502.03167* .

Islam, T., Rico-Ramirez, M. A., Han, D. and Srivastava, P. K. (2012), 'Artificial intelligence techniques for clutter identification with polarimetric radar signatures', *Atmospheric Research* **109**, 95–113.

Karpathy, A. and Fei-Fei, L. (2015), Deep visual-semantic alignments for generating image descriptions, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 3128–3137.

Kingma, D. P. and Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv:1412.6980* .

Krizhevsky, A., Hinton, G. et al. (2009), Learning multiple layers of features from tiny images, Technical report, Citeseer.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* 'Advances in neural information processing systems', pp. 1097–1105.

Kröse, B. and van der Smagt, P. (1993), 'An introduction to neural networks'.

Lange, I. (2016), Verwendung der kamera vivotek fe8174v als wolkenkamera, Technical report, University of Hamburg.
URL: *https://wettermast.uni-hamburg.de/Downloads/Wolkenkamera_FE8174V.pdf*

LeCun, Y., Bengio, Y. et al. (1995), 'Convolutional networks for images, speech, and time series', *The handbook of brain theory and neural networks* **3361**(10), 1995.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E. and Jackel, L. D. (1990), Handwritten digit recognition with a back-propagation network, *in* 'Advances in neural information processing systems', pp. 396–404.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. et al. (1998), 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* **86**(11), 2278–2324.

Lee, G. W. (2003), Errors in rain measurement by radar: Effect of variability of drop size distributions, PhD thesis, Department of Atmospheric and Oceanic Sciences, McGill University, Montreal.

Lee, G. W. and Zawadzki, I. (2005), 'Variability of drop size distributions: Time-scale dependence of the variability and its effects on rain estimation', *Journal of applied meteorology* **44**(2), 241–255.

Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M. and Aila, T. (2018), 'Noise2noise: Learning image restoration without clean data', *arXiv:1803.04189* .

Lengfeld, K., Clemens, M., Muenster, H. and Ament, F. (2014), 'Performance of high-resolution x-band weather radar networks–the pattern example', *Atmospheric Measurement Techniques* **7**, 4151–4166.

Lengfeld, K., Clemens, M., Münster, H. and Ament, F. (2013), Pattern: Advantages of high-resolution weather radar networks, *in* 'Proceedings of American Meteorological Society 36th Conference on Radar Meteorology, Breckenridge, CO, USA', pp. 16–20.

Loh, W.-Y. (2011), 'Classification and regression trees', *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **1**(1), 14–23.

MacQueen, J. et al. (1967), Some methods for classification and analysis of multivariate observations, *in* 'Proceedings of the fifth Berkeley symposium on mathematical statistics and probability', Vol. 1, Oakland, CA, USA, pp. 281–297.

Marshall, J. S. and Palmer, W. M. K. (1948), 'The distribution of raindrops with size', *Journal of meteorology* **5**(4), 165–166.

Masters, D. and Luschi, C. (2018), 'Revisiting small batch training for deep neural networks', *arXiv preprint arXiv:1804.07612* .

McCulloch, W. S. and Pitts, W. (1943), 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**(4), 115–133.

Murata, N., Yoshizawa, S. and Amari, S.-i. (1994), 'Network information criterion-determining the number of hidden units for an artificial neural network model', *IEEE transactions on neural networks* **5**(6), 865–872.

Parkhi, O. M., Vedaldi, A., Zisserman, A. et al. (2015), Deep face recognition, *in* 'bmvc', Vol. 1, p. 6.

Pascanu, R., Mikolov, T. and Bengio, Y. (2013), On the difficulty of training recurrent neural networks, *in* 'International conference on machine learning', pp. 1310–1318.

Powell, C. E. et al. (2014), 'Generating realisations of stationary gaussian random fields by circulant embedding', *matrix* **2**(2), 1.

Quinlan, J. R. (1986), 'Induction of decision trees', *Machine learning* **1**(1), 81–106.

Recht, B., Roelofs, R., Schmidt, L. and Shankar, V. (2018), 'Do cifar-10 classifiers generalize to cifar-10?', *arXiv preprint arXiv:1806.00451* .

Reichstein, M., Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., Carvalhais, N. and Prabhat (2019), 'Deep learning and process understanding for data-driven earth system science', *Nature* **566**(7743), 195–204.

Ronneberger, O., Fischer, P. and Brox, T. (2015), 'U-net: Convolutional networks for biomedical image segmentation'.

Saltikoff, E., Cho, J. Y., Tristant, P., Huuskonen, A., Allmon, L., Cook, R., Becker, E. and Joe, P. (2016), 'The threat to weather radars by wireless technology', *Bulletin of the American Meteorological Society* **97**(7), 1159–1167.

SHI, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-k. and WOO, W.-c. (2015), Convolutional lstm network: A machine learning approach for precipitation nowcasting, *in* 'Advances in Neural Information Processing Systems 28', Curran Associates, Inc., pp. 802–810.

Sibi, P., Jones, S. A. and Siddarth, P. (2013), 'Analysis of different activation functions using back propagation neural networks', *Journal of Theoretical and Applied Information Technology* **47**(3), 1264–1268.

Sigal, L. and Black, M. J. (2006), 'Humaneva: Synchronized video and motion capture dataset for evaluation of articulated human motion', *Brown Univertsity TR* **120**.

Skolnik, M. I. (1970), 'Radar handbook'.

Spaulding, A. D. and Washburn, J. S. (1985), 'Atmospheric radio noise: Worldwide levels and other characteristics', *NASA STI/Recon Technical Report N* **86**.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014), 'Dropout: a simple way to prevent neural networks from overfitting', *The Journal of Machine Learning Research* **15**(1), 1929–1958.

Stone, Z., Zickler, T. and Darrell, T. (2008), Autotagging facebook: Social network context improves photo annotation, *in* '2008 IEEE computer society conference on computer vision and pattern recognition workshops', IEEE, pp. 1–8.

Strasser, B. J. (2012), 'Data-driven sciences: From wonder cabinets to electronic databases', *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences* **43**(1), 85–87.

Wang, Z. and Bovik, A. C. (2009), 'Mean squared error: Love it or leave it? a new look at signal fidelity measures', *IEEE Signal Processing Magazine* **26**(1), 98–117.

Widrow, B. and Hoff, M. E. (1960), Adaptive switching circuits, Technical report, Stanford Univ Ca Stanford Electronics Labs.

Zhang, Z. and Sabuncu, M. (2018), Generalized cross entropy loss for training deep neural networks with noisy labels, *in* 'Advances in Neural Information Processing Systems 31', Curran Associates, Inc., pp. 8778–8788.

Zhou Wang, Bovik, A. C., Sheikh, H. R. and Simoncelli, E. P. (2004), 'Image quality assessment: from error visibility to structural similarity', *IEEE Transactions on Image Processing* **13**(4), 600–612.

Zhou, Y. and Chellappa, R. (1988), Computation of optical flow using a neural network, *in* 'IEEE International Conference on Neural Networks', Vol. 1998, pp. 71–78.

Appendix

This table provides the complete model setup with all layers used for producing the results presented in this thesis. Zero padding and polar padding are split up into two separate layers. Zero padding is only applied to the range dimension while polar padding is only applied to the angle dimension.

Layer Name	**Layer Type**	**Output Shape**	**Input**
input-1	Input Layer	(360, 320, 1)	
zero-padding2d-1	Zero Padding	(360, 322, 1)	input-1
lambda-1	Polar Padding	(362, 322, 1)	zero-padding2d-1
conv2d-1	Convolution	(360, 320, 20)	lambda-1
batch-norm-1	Batch Normalization	(360, 320, 20)	conv2d-1
leaky-re-lu-1	LeakyReLU	(360, 320, 20)	batch-norm-1
zero-padding2d-2	Zero Padding	(360, 322, 20)	leaky-re-lu-1
lambda-2	Polar Padding	(362, 322, 20)	zero-padding2d-2
conv2d-2	Convolution	(360, 320, 20)	lambda-2
batch-norm-2	Batch Normalization	(360, 320, 20)	conv2d-2
leaky-re-lu-2	LeakyReLU	(360, 320, 20)	batch-norm-2
max-pooling2d-1	Max-Pooling	(180, 160, 20)	leaky-re-lu-2
zero-padding2d-3	Zero Padding	(180, 162, 20)	max-pooling2d-1
lambda-3	Polar Padding	(182, 162, 20)	zero-padding2d-3
conv2d-3	Convolution	(180, 160, 40)	lambda-3
batch-norm-3	Batch Normalization	(180, 160, 40)	conv2d-3
leaky-re-lu-3	LeakyReLU	(180, 160, 40)	batch-norm-3
max-pooling2d-2	Max-Pooling	(90, 80, 40)	leaky-re-lu-3
zero-padding2d-4	Zero Padding	(90, 82, 40)	max-pooling2d-2
lambda-4	Polar Padding	(92, 82, 40)	zero-padding2d-4
conv2d-4	Convolution	(90, 80, 60)	lambda-4
batch-norm-4	Batch Normalization	(90, 80, 60)	conv2d-4
leaky-re-lu-4	LeakyReLU	(90, 80, 60)	batch-norm-4
max-pooling2d-3	Max-Pooling	(45, 40, 60)	leaky-re-lu-4
zero-padding2d-5	Zero Padding	(45, 42, 60)	max-pooling2d-3
lambda-5	Polar Padding	(47, 42, 60)	zero-padding2d-5

Layer Name	Layer Type	Output Shape	Input
conv2d-5	Convolution	(45, 40, 80)	lambda-5
batch-norm-5	Batch Normalization	(45, 40, 80)	conv2d-5
leaky-re-lu-5	LeakyReLU	(45, 40, 80)	batch-norm-5
max-pooling2d-4	Max-Pooling	(15, 20, 80)	leaky-re-lu-5
zero-padding2d-6	Zero Padding	(15, 22, 80)	max-pooling2d-4
lambda-6	Polar Padding	(17, 22, 80)	zero-padding2d-6
conv2d-6	Convolution	(15, 20, 100)	lambda-6
batch-norm-6	Batch Normalization	(15, 20, 100)	conv2d-6
leaky-re-lu-6	LeakyReLU	(15, 20, 100)	batch-norm-6
up-sampling2d-1	Up-Sampling	(45, 40, 100)	leaky-re-lu-6
concatenate-13	Concatenation	(45, 40, 160)	up-sampling2d-1 max-pooling2d-3
zero-padding2d-7	Zero Padding	(45, 42, 160)	concatenate-13
lambda-7	Polar Padding	(47, 42, 160)	zero-padding2d-7
conv2d-7	Convolution	(45, 40, 120)	lambda-7
batch-norm-7	Batch Normalization	(45, 40, 120)	conv2d-7
leaky-re-lu-7	LeakyReLU	(45, 40, 120)	batch-norm-7
zero-padding2d-8	Zero Padding	(45, 42, 120)	leaky-re-lu-7
lambda-8	Polar Padding	(47, 42, 120)	zero-padding2d-8
conv2d-8	Convolution	(45, 40, 120)	lambda-8
batch-norm-8	Batch Normalization	(45, 40, 120)	conv2d-8
leaky-re-lu-8	LeakyReLU	(45, 40, 120)	batch-norm-8
up-sampling2d-2	Up-Sampling	(90, 80, 120)	leaky-re-lu-8
concatenate-18	Concatenation	(90, 80, 160)	up-sampling2d-2 max-pooling2d-2
zero-padding2d-9	Zero Padding	(90, 82, 160)	concatenate-18
lambda-9	Polar Padding	(92, 82, 160)	zero-padding2d-9
conv2d-9	Convolution	(90, 80, 100)	lambda-9
batch-norm-9	Batch Normalization	(90, 80, 100)	conv2d-9
leaky-re-lu-9	LeakyReLU	(90, 80, 100)	batch-norm-9
zero-padding2d-10	Zero Padding	(90, 82, 100)	leaky-re-lu-9
lambda-10	Polar Padding	(92, 82, 100)	zero-padding2d-10
conv2d-10	Convolution	(90, 80, 100)	lambda-10
batch-norm-10	Batch Normalization	(90, 80, 100)	conv2d-10
leaky-re-lu-10	LeakyReLU	(90, 80, 100)	batch-norm-10
up-sampling2d-3	Up-Sampling	(180, 160, 100)	leaky-re-lu-10
concatenate-23	Concatenation	(180, 160, 120)	up-sampling2d-3 max-pooling2d-1
zero-padding2d-11	Zero Padding	(180, 162, 120)	concatenate-23
lambda-11	Polar Padding	(182, 162, 120)	zero-padding2d-11

Layer Name	Layer Type	Output Shape	Input
conv2d-11	Convolution	(180, 160, 80)	lambda-11
batch-norm-11	Batch Normalization	(180, 160, 80)	conv2d-11
leaky-re-lu-11	LeakyReLU	(180, 160, 80)	batch-norm-11
zero-padding2d-12	Zero Padding	(180, 162, 80)	leaky-re-lu-11
lambda-12	Polar Padding	(182, 162, 80)	zero-padding2d-12
conv2d-12	Convolution	(180, 160, 80)	lambda-12
batch-norm-12	Batch Normalization	(180, 160, 80)	conv2d-12
leaky-re-lu-12	LeakyReLU	(180, 160, 80)	batch-norm-12
up-sampling2d-4	Up-Sampling	(360, 320, 80)	leaky-re-lu-12
concatenate-28	Concatenation	(360, 320, 81)	up-sampling2d-4 input-1
zero-padding2d-13	Zero Padding	(360, 322, 81)	concatenate-28
lambda-13	Polar Padding	(362, 322, 81)	zero-padding2d-13
conv2d-13	Convolution	(360, 320, 20)	lambda-13
batch-norm-13	Batch Normalization	(360, 320, 20)	conv2d-13
leaky-re-lu-13	LeakyReLU	(360, 320, 20)	batch-norm-13
zero-padding2d-14	Zero Padding	(360, 322, 20)	leaky-re-lu-13
lambda-14	Polar Padding	(362, 322, 20)	zero-padding2d-14
conv2d-14	Convolution	(360, 320, 20)	lambda-14
batch-norm-14	Batch Normalization	(360, 320, 20)	conv2d-14
leaky-re-lu-14	LeakyReLU	(360, 320, 20)	batch-norm-14
conv2d-15	Convolution (Output)	(360, 320, 1)	leaky-re-lu-14

Acknowledgement

At the Meteorological Institute of the University of Hamburg, a python package is being developed, which is meant for easy processing and displaying radar data (pylawr). It was a great convenience while working with the data for this thesis, remapping grids and plotting reflectivity fields with ease. So thanks to Yann Büchau, Maximilian Schaper, Claire Merker and Marco Clemens. Special thanks goes to Tobias Finn and Finn Burgemeister, which always took the time to explain all the little details of the package to me and provided lots of help with it.

Another special thanks goes to my first advisor Marco Clemens. For him the open-door policy is not just a phrase. I could always come over for a little chat. Usually these occasions start by me asking a scientific question and ending in chats and questions like where in Hamburg the best open-air pool is.

Furthermore, I would like to mention all the people who read the alpha-version of this thesis: Marcus Klingebiel, Marco Clemens, Felix Ament, Tobias Finn and Finn Burgemeister. Thanks to your amazing feedback I was able to improve this thesis a lot, both grammatically and substantial.

Tobias Finn has to be mentioned a third time here. He not only helped me with the pylawr, but also in a lot of questions to NNs, at the beginning of my research. Thanks to the lively discussions we had and his literature suggestions the thesis became what it is.

Last but not least I would like to thank my girlfriend Hannah Schanz. She does not understand a lot of the content of this thesis, but she always lend a sympathetic ear when I explained her the problems I encountered. She showed understanding for me staying at the university until late in the evening and when I was down because of bugs that took days to find or because of anything else, she always made those problems appear not so big anymore.

Versicherung an Eides statt

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang M.Sc. Meteorologie selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum	Tobias Machnitzki