Precipitation Nowcasting:
a Deep Learning Approach

Master's Thesis

in Environmental Meteorology

Submitted to the
Faculty of Geo- and Atmospheric Sciences
of the
University of Innsbruck

in Partial Fulfillment of the Requirements for the Degree of
Master of Science

by
Stefano Campostrini

Advisor
Univ.-Prof. Dr. Georg Mayr

Innsbruck, May 2022

To all who have been close to me in this journey,
especially from far away

Abstract

Forecasting precipitation at very short lead times is a task that is difficult to address through numerical weather prediction models due to the great amount of computing power required to solve the physical equations that govern the behaviour of the atmosphere in a timely fashion. The advances in the field of deep learning and the large amount of historical data of radar precipitation maps that are openly available make the precipitation nowcasting problem a good candidate for deep learning solutions. To assess the potential of the U-Net neural network architecture at the 1hr time scale a comparison is made of its performance to persistence, i.e. the assumption that the prediction is equal to the last available frame. The problem is structured as a multi-class classification task over 4 precipitation intensity classes. The model takes 6 input images of 1h cumulative precipitation of the RW product from the RADar Online ANeichung (RADOLAN) dataset of the German weather service (DWD) and an area of 256km$\times$256km over northern Germany is selected. The task is to predict the precipitation classes of the pixels of the RADOLAN image at 6h from the first input image. It is found that the model performs better than persistence at higher precipitation rates. For rainfall rates in [0mm/h, 0.1mm/h) the model has a Critical Success Index (CSI) of 0.853 compared to 0.893 of persistence. For rainfall rates in [0.1mm/h, 1mm/h), [0.1mm/h, 1mm/h) and [0.1mm/h, ∞mm/h) the model has better CSI than persistence (0.292 vs 0.276, 0.253 vs 0.182 and 0.191 vs 0.128 respectively). For the same classes the Heidke Skill Score (HSS) absolute performance diminishes with increasing precipitation rate but gets better relative to persistence (0.579 vs 0.583, 0.438 vs 0.441, 0.404 vs 0.314 and 0.321 vs 0.229). Finally, both a higher average HSS for the model compared to persistence (0.436 compared to 0.392) and a higher average CSI (0.397 compared to 0.370) are found. A visual inspection of the predictions suggests that the neural network is able to infer the general structure of the precipitation field but has a tendency to predict higher rain rates. It can be concluded that the results are encouraging and that the application of the U-Net architecture to 1h cumulative precipitation maps should be further investigated.

Contents

Chapter 1

Introduction

1.1 Motivation

Current weather forecasts are based on global numerical weather prediction (NWP) models that solve the governing physical equations of the atmosphere to predict its future states. If the forecast happens for very short time-frames in the order of a couple of hours to, according to some authors, 6 hours, the forecast is referred to as nowcast. However, for these time-frames, numerical methods of predicting the weather are bound by the incredible amount of computing power required leading to a necessary trade-off between timeliness of the prediction and spatial resolution. For instance the High-Resolution Rapid Refresh model (HRRR) is a National Oceanic and Atmospheric Administration (NOAA) atmospheric model with 3km spatial resolution updated hourly (NOAA 2021) that covers the extension of the United States.

For predicting variables of high interest to the public such as spatial precipitation distribution, optical flow algorithms are currently being used for time horizons from 1h to 4h. These algorithms use geometrical features in the input images and their spatial movement – for instance by using radar reflectivity maps – to extrapolate their position at future timestamps.

The governing equations of the physical processes behind the behaviour of the atmosphere are of a non-linear nature. This suggests that non-linear statistical models such as artificial neural networks – that abstract features from a historical dataset – would be able to represent these processes. Deep learning – i.e. neural networks with many hidden layers – has recently become a plausible method of predicting physical processes mainly driven by historical-data availability and reduced cost of the devices used for the relevant computations (Graphics Processing Units or GPUs).

Given the historical archives of radar precipitation maps it is clear that deep learning could be applied to the precipitation nowcasting problem. A significant point in favor of investigating this approach is that, after the moderately low cost of the training of a deep neural network model, the production of predictions (also called inferences) are characterized by both an incredibly low cost and very short time frames (in the order of 1sec). So much so that this can even happen consumer-grade GPUs. This properties satisfy the timeliness requirement of precipitation nowcasting.

1.2 State of the Art

Artificial neural networks find their origin in McCulloch and Pitts (1943) where the authors describe them as having thresholds and weights finding that, in principle, a neural network could compute every function that a physical computer is able to compute. Rosenblatt (1958) proposed the first trainable neural network. The perceptron – as it is called – has all the characteristics of modern day neural networks except that it is very simple in its design. From this original design the multi-layer perceptron has become the standard artificial neural network. It is also known as fully-connected feed-forward neural network and is fundamentally made of units grouped into layers. The information flows from the input layer to the output layer passing through the (hidden) layers in-between. The units in a layer take as input the signals from all the units of the layer before them, they weight them according to specific weights that are subject to optimization, apply a non-linear function acting as threshold and finally pass on a value to all the units of the following layer until the signal reaches the output unit. Neural networks have found many applications throughout the sciences due to their versatility in finding statistical relations. Their main drawbacks are the design – no canonical designs exist for a given task – the search for the right combination of parameters governing the behaviour of the model (also called hyperparameters) and the computational demands compared to other statistical methods.

Deep learning has been used in the physical sciences for predictions spanning the entire field for quite some time now, mainly led by the general advances of the machine learning field. A clear example of the use of deep learning in the Earth sciences is Steffenel et al. (2021) where a deep neural network forecasted O3 advection. Reichstein et al. (2019) give a thorough overview of deep learning in Earth Sciences.

As it pertains to the particular task of predicting the weather, many approaches have been proposed for a variety of different tasks. Using deep learning methods to forecast future states of the atmosphere is also called deep learning weather prediction (DLWP). Ren et al. (2021) propose three different groupings for the architectures that solve weather prediction problems. (1) those based on well known architectures from the general literature, such as autoencoders (Lin et al. 2018; Hossain et al. 2015), convolutional neural networks (Qiu et al. 2017) or long-short-term memory networks (Karevan and Suykens 2018), (2) hybrid architectures composed of a combination of those in (1), (3) a union of deep neural networks with NWP models to join the benefits of both theory-driven and data-driven approaches.

In the specific case of precipitation nowcasting, many different solutions have been employed (see Ren et al. (2021) for a survey of the current methods).

Shi et al. (2015) propose a convolutional long-short-term memory (ConvLSTM) network for precipitation nowcasting that consists of a combination of convolutional and LSTM network structures. However, a main drawback is that as structured, it is location invariant. For this reason Shi et al. (2017) propose the trajectory grated recurrent unit (TrajGRU) model that adds the ability for the model to actively learn the structure of natural motion and transformation which is location-dependent. This kind of model has been shown to perform exceptionally well for time-series predictions as the recurrent structure implicitly assumes a chaining of past and future events of the series. Wang et al. (2017, 2018) also adopt variations of the LSTM core structure proposing the predictive recurrent neural network (PredRNN) and the improvement PredRNN++ respectively. Wang and Hong (2018) successfully applies them to the task of precipitation nowcasting.

Google Research Agrawal et al. (2019); Sønderby et al. (2020) were both able to successfully apply DLWP to precipitation now-casting even surpassing NOAA's HRRR model. The U-Net architecture, used in Agrawal et al. (2019) and Ayzel et al. (2020), is directly inspired from the well known model introduced in Ronneberger et al. (2015) and operate with radar precipitation images at 10min and 5min intervals respectively. With the implementation of skip connections (He et al. 2016a,b), this model gave good results in both applications to precipitation nowcasting. The main idea behind this type of architecture is the division of the network into two distinct parts, the encoder and the decoder. The first part is responsible for abstracting features that, with increasing depth, become more high-level. The second part of the network tries to rebuild a plausible image by using more low-level features when the stream nears to the output layer.

1.3 Goals and Outline

The aim of this project is to investigate the use of deep learning with 1h cumulative data of precipitation in northern Germany with a U-Net architecture model inspired by Agrawal et al. (2019). Some intuitions from Ayzel et al. (2020) are used. The U-Net is applied to a dataset of hourly cumulated precipitation amounts freely provided by the German weather service - Deutscher Wetterdienst (DWD). The model is tasked with predicting the future radar precipitation maps in the simplified form of 4 mutually exclusive precipitation classes by taking as input multiple continuously-valued precipitation maps for the preceding timestamps. This task is also known as multi-class classification or multi-class segmentation. To better understand the predictive power of the model, it is compared with the persistence benchmark. This benchmark consists of predicting a future state of the atmosphere by assuming it remains unchanged from the last available timestamp before the prediction. Albeit its simplicity, persistence predictions are not trivially easy to outperform. To facilitate the training process of the model, avoid overfitting and better the final results – a process called regularization – this project uses methods taken directly from the deep learning literature. The thesis will address the following questions:

(1) Does the employed neural network outperform persistence?

(2) How does model performance depend on the rainfall intensity?

(3) The model of Agrawal et al. (2019) after which this project is inspired outperformed persistence for all precipitation classes. Can this result be confirmed?

The second and next chapter of this thesis covers the theoretical principles behind the production of radar precipitation maps, addresses the dataset used in the project and its pre-processing for the specific applications to the task addressed with the proposed model. It illustrates the workings of neural networks giving more attention to the components of the model architecture that are used in this project. Details are given on the deep learning regularization techniques that are later employed. Chapter three provides the results of the project. Finally, chapter four states the answers to the research questions, discusses and compares them with similar studies and suggests plausible developments and open questions.

Chapter 2

Data and Methods

2.1 Data

2.1.1 Radar Principles

Radar meteorological observations are of extreme importance for a multitude of variables. The main application of interest in this project is the use of radar for determining the precipitation amount through the so-called reflectivity. The fundamental principle behind weather radars is the transmission of electromagnetic waves at a known wavelength (typically in the range of 3-10GHz) in the microwave range that are either absorbed or scattered and their subsequent detection. The emission power is in the order of 10^2W - 10^3W and the power received amounts to a mere 10^{-16}kW hence a staggering 18 orders of magnitude. Another challenge for the receiver is the 9 orders of magnitude spanned by the incoming power depending on the difference in distance and type of possible targets that go from hail to dust. The signal is emitted in a beam of approximately 1-2°width. Many factors affect absorption and scattering of the incoming radiation such as shape, dielectrical properties of the targets and the ratio of the wavelength to the size of the particle doing the scattering. Scattering itself occurs at two different regimes, one for a high wavelength-to-size ratio and one for low wavelength-to-size ratio; In the *Rayleigh scattering regime* – which is the first of the two – scattered energy goes with D^6 where d is the diameter of the scatterer. *Mie scattering* happens in the other case and the governing law is not a simple function of target size.

A very important quantity in radar observations is the so-called *logarithmic reflectivity factor* expressed as

$$Z = 10 \log_{10} \left(\frac{z}{1\text{mm}^6/\text{m}^3} \right) \tag{2.1}$$

with z the reflectivity factor none other than the summation of target diameters to the sixth power, i.e.

$$z = \int_0^\infty N(D) D^6 dD \tag{2.2}$$

where $N(D)$ is the drop-size distribution considered to be a continuous function of drop size D.

Z is given in dBZ (decibels relative to 1mm^6/m^3). Reflectivity factor z and received backscattered power P_r are bound by

$$z = CP_r r^2 \tag{2.3}$$

with a radar constant C dependent on the charachteristics of the radar, and r the distance to the target.

Another important factor affecting backscattered power is attenuation, i.e. the absorption and scattering of the radar signal along the path to the target. This affects radars with $\lambda <$ 10cm the most.

The signal wavelengths most commonly used for weather radar applications are in the S-Band (7.5cm $< \lambda <$ 15cm) and C-Band (3.75cm $< \lambda <$ 7.5cm). The US National Weather Service (NWS) employs a set of 160 S-band radars in the Nexrad (Next-Generation Radar) radar network. European weather services, on the other hand, mostly employ C-band radars.

The assignment of part of the incoming scattered signal to a specific range (or distance) from the radar is attained through a periodical sampling (every 0.1-1μs). The location of the sampled volume is readily determined by knowing azimuth angle, elevation angle and the distance from the antenna given by

$$r = \frac{ct}{2} \tag{2.4}$$

with c being the speed of light and t the amount of time for the signal to arrive at the target scatterer volume and come back. Additional corrections should be made to account for the Earth's curvature and refraction - the refractive index is a strong function of the air density. In particular cases, such as when a strong temperature inversion is present, the beam can make contact with the ground through strong deflection.

Z-R relationships

The simplest relationship between logarithmic reflectivity factor Z and rainfall rate R is based on the assumption that

$$Z = AR^B \tag{2.5}$$

with A and B being constants. Given a D^6 dependence of Z and a D^3 dependence of R, without knowing the drop size distribution this method is prone to errors. Nonetheless, with reasonable assumptions (i.e. a Marshall-Pamer drop size distribution) one obtains

$$Z = 200R^{1.6} \tag{2.6}$$

A correction to this relationship is made during the RADOLAN routine (RADar-OnLine-ANeichung Deutscher Wetterdienst (2022b)) – see Section 2.1.2.

In addition to the uncertainties of A and B in Eq. (2.5) due to not knowing the drop size distributions, other sources of errors for the correct determination of the rainfall rate include:

- The inability to measure the reflectivity near the ground the further away from the radar site. Not only is there a normal increase in distance due to a slight angle of the radar beam with respect to the horizon (≈0.8°), but the curvature of the earth also has a significant impact on the height reached by the beam.
- Attenuation of the beam due to orography
- Rainfall rate gradients
- Errors in the calibration of the radar
- The presence of hail, thus invalidating the Rayleigh scattering approximation

Rain gauges, however, measure the actual rainfall rate on the ground but only in the spot where they are located. The measured rainfall rate is therefore representative for a limited area. The RADOLAN procedure, described in the following section, combines the the real rainfall rates measured through the rain gauges with the radar obtained precipitation maps. The final product of this procedure is a precipitation map that retains the spatial characteristics of the precipitation distribution from the weather radars but has values closer to the ground truth compared to a precipitation map obtained only through the conversion of the radar reflectivity to the rainfall rates with a simple Z-R relationship.

2.1.2 Radolan Dataset

The German Weather Service (DWD - Deutsches Wetter Dienst) offers a variety of products derived from their deployed sensor suite of weather radars and rain gauges. One of these products is the timely online calibration of radar spatial precipitation distributions (RADOLAN - RADar-OnLine-ANeichung Deutscher Wetterdienst (2022b)) that are used to produce quantitative radar-nowcasting distributions (RADVOR - RADar - NiederschlagsVORhersage Deutscher Wetterdienst (2022c)), as an input for the COSMO-DE Numerical Weather Prediction model and for the convection development model (KONRAD - KONvektion Entwicklung in RADarprodukten Deutscher Wetterdienst (2022a)).

Sensor Field

For the production of RADOLAN precipitation distribution maps the DWD employs 17 C-Band Radars that cover the entirety of Germany. These measure the precipitation rate indirectly by capturing the reflection of the radar signals by the hydrometeors above ground. Fig. 2.1 shows the map and the 150km ranges of the various Weather Radars of the DWD.

The reflectivity obtained through radar measurements can in theory be converted to a precipitation amount on the ground but in practice the Z-R relationship of Eq. (2.5) produces deviations from the true value at the ground. For this reason it is useful to utilize the measured precipitation rate at the various rain gauges as true values with which to produce a more value-accurate precipitation distribution. The DWD employs its rain gauges exactly for this purpose. Fig. 2.2 shows the location of these gauges in Germany.

To include the entire catchments covering Germany, rainfall distributions from other radar networks and rain gauges are integrated in the RADOLAN products covering the middle-european area.

Calibration method

The RADOLAN calibration routine is very elaborate. I will try to give a very brief overview of its most important components and principles. Please refer to Weigl et al. (2004) for a complete account and to Weigl (2021a) for the subsequent updates.

The routine is applied on a *grid* representing 1km×1km *squares* or *tiles* of data obtained by joining the radar reflectivity maps from the single weather radars. The grid covers the area of Germany and part of its surroundings depending on which Radolan product is considered. The coordinates of the tiles are obtained trhough polar stereographic projection. The projection plane touches the Earth's sphere at 60.0°N. The Cartesian coordinate system has dimensions of 900km×900km and is parallel to the 10.0°E-meridian. The coordinates of the central point of the composite are 9.0°E and 51.0°N. The Earth's sphere is assumed to be regular and with a radius of 6370.04 km . Fig. 2.3 shows this spatial extension and Fig. 2.4 shows a depiction of the grid structure where the coordinates of the grid points represent the lower left corner of each tile.

The first step in the RADOLAN calibration procedure is a pre-processing step. Here the radar data is compensated for shadowing effects caused by the orography, an empirically derived and more precise reflectivity-rainrate (Z-R) relation is applied following Weigl (2015). Then the various sub-hour radar images are put together, a statistical suppression of clutter is applied by removing hot pixels and substituting

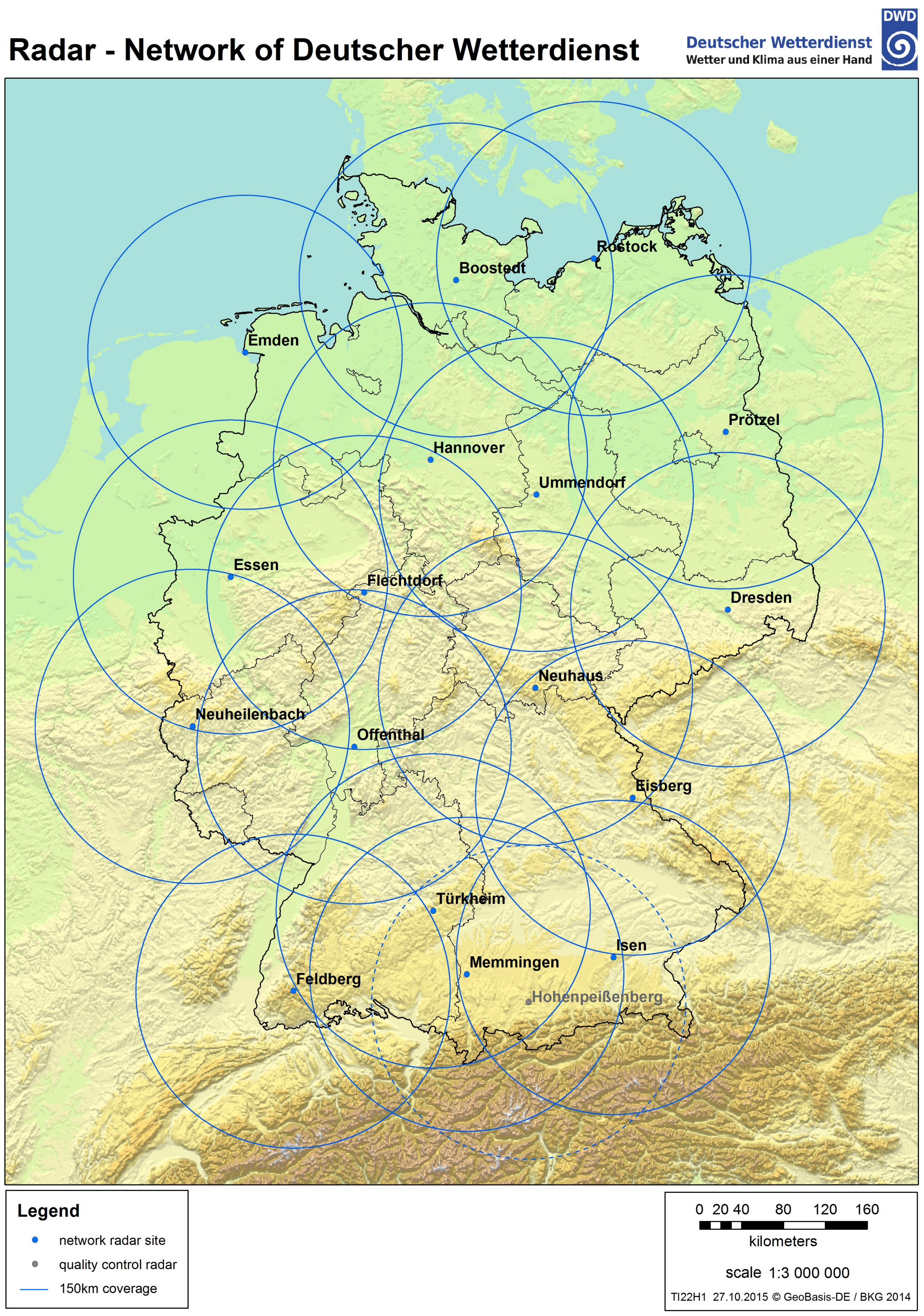

Figure 2.1: Radar network of the DWD composed of 18 C-Band Radars. 17 of them are operational radars and 1 is a quality assurance radar for testing of new hardware and software (Hohenpeißenberg). The circles depict the 150km range of the radars at their respective centers that are used for the quantitative assessment of precipitation for the RADOLAN products. Figure taken from Deutscher Wetterdienst (2022d).

them with an interpolated value. Finally in the last part of the pre-processing step

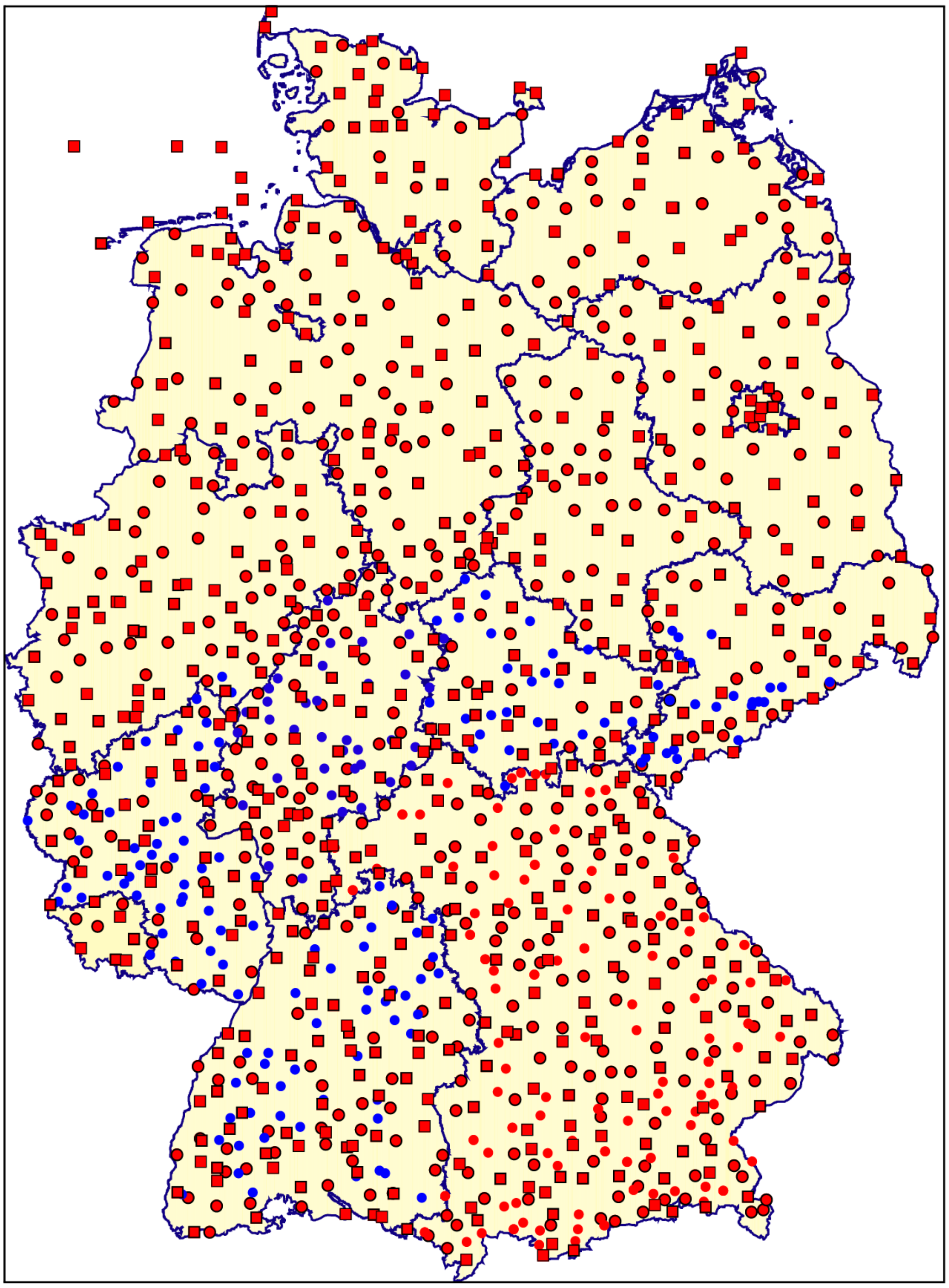

Figure 2.2: Regional distribution of the rain gauges of the DWD. **Squares**: gauges with hourly retrieval; **circles**: gauges with hourly retrieval in case of precipitation incident (more on when these gauges are activated can be found in Weigl et al. (2004)) — **Red**: data transmitted through the AMDA III system. (**without border**: Bavaria); **blue**: other rain gauges operated by the *Länder*. Image from Weigl et al. (2004).

the gradients are smoothed down and a pre-calibration with the rain gauges values is put into place (Weigl et al. 2004).

Subsequently, for the step of the actual calibration, the RADOLAN routine

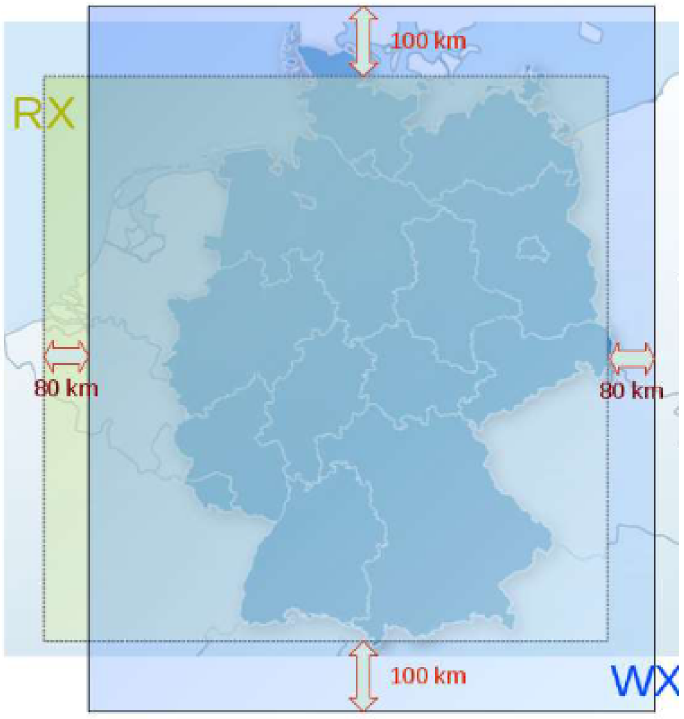

Figure 2.3: Depiction of Germany and the area covered by the RX composite (yellow). The RX composite has the same extension as the RW composite but it represents the uncorrected radar data in RVP6-units. The EX composite (blue) is an expansion of the RX product; it involves a greater area taking into account more radars that are not directly operated by the DWD. Figure reprinted from Weigl (2021b).

blends three different values for every tile of the composite. The three values are the result of three different calibration methods.

1. The first method consists in finding a multiplicative factor for the radar map. It is based on the assumption that the rain gauges values (RG) and the radar precipitation values (Rad) are related through

$$F_i = \frac{R_i^{\text{RG}}}{R_i^{\text{Rad}}}. \tag{2.7}$$

   The factors are computed for every relevant i-th rain gauge and corresponding radar composite tile associated to it. The 8 adjacent radar composite tiles are also taken into account. A weighted average takes place to find the interpolation factors for all of the radar tiles. The weights for the average are given by

$$w(d) = \frac{\exp(ad)}{1 - \exp(ad)} \quad \text{with} \quad a = \frac{\ln(0.5) \cdot 20}{r_{\text{search}}} \tag{2.8}$$

   where d is the distance of the tile from a station within a search-radius r_{search} (typically 40km). This is called the factor-method.

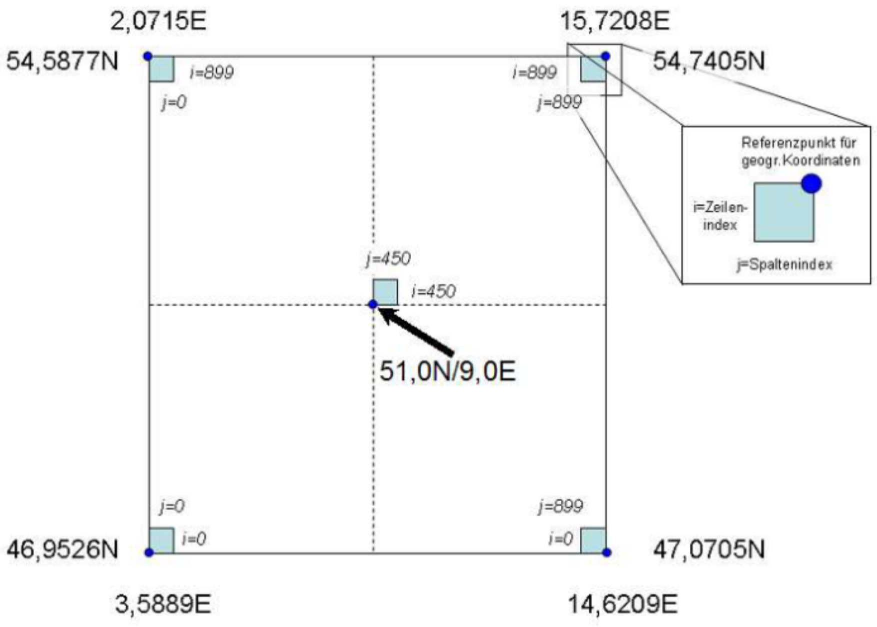

Figure 2.4: Depiction of the RADOLAN grid and the respective tiles in polar stereographic projection. The arrow points to the center grid point and its respective tile. The indices i and j represent the row and column number of the tile respectively. The blue grid points on the upper left and right corners and on the bottom right corner are shown only to better characterize the extension of the tiles grid. Figure reprinted from Weigl (2021b).

2. The second method finds a constant to be summed to the radar map. The assumption governing this method is that the relationship between the rain gauges values and the radar precipitation values is of the kind

$$D_i = R_i^{\mathrm{RG}} - R_i^{\mathrm{Rad}}. \tag{2.9}$$

The procedure to obtain the values of the differences for every tile is the same as for the factor-method. We shall call this the differences-method.

3. A third method consisting of the combination of three rainfall distribution maps (1) an interpolated precipitation distribution field on the radar values grid obtained through ordinary kriging using the rain gauges rainfall rates, (2) an interpolated precipitation distribution field on the radar values grid obtained through ordinary kriging using the radar-derived rainfall rates at the points of the rain gauges and (3) the observed radar precipitation distribution. We shall call this the merging-method.

The combination of these three methods follows the equation

$$\mathbf{R}_{ij} = \frac{\mathbf{F}_{ij} + \mathbf{D}_{ij} + \mathbf{M}_{ij}}{w(\mathbf{F}_{ij}) + w(\mathbf{D}_{ij}) + w(\mathbf{M}_{ij})} \tag{2.10}$$

where **R** is the precipitation matrix and ij are the row and column indices. **F**, **D** and **M** are the results of the factor-method, the differences-method and the Merging-method respectively. The w represent the interpolated weights for the tile ij. The weights are derived by comparing the values obtained through the three calibrations method with the true value at a specific subset of the rainfall-affected rain gauges. This subset is called the control rain gauges. The weight for the best calibration method at every control station individually is set to 1 and all the others to 0. An interpolation from these weights to the entire grid of the radar field gives the resulting weights (Weigl et al. 2004).

Final product

The final and best openly available product outputted by the RADOLAN calibration procedure is the so-called RW composite. This composite doesn't include the merging-method of calibration and therefore only employs the factor-method and the differences-method. It is computed every hour and represents the hourly sums of precipitation as calibrated through the RADOLAN procedure (Weigl 2021b). Fig. 2.3 depicts the extension of the composite over the map of Germany and Fig. 2.5 shows an example of a single frame of the RW composite. The rainfall rate is given in units of 0.1mm/h for a total of 900km×900km over Germany as previously described.

The RW composite is accessible at `https://opendata.dwd.de/climate_environment/CDC/grids_germany/hourly/radolan/`.

2.1.3 Dataset Selection - RADNET Dataset

I now describe the area of interest and the periods of interest for this project. I also give a short overview of the algorithm used to produce the relevant data for the neural network employed in this thesis. I call this selection the RADNET dataset.

Area of Interest

Given that additional measurements errors are introduced by topography, I select a reasonably flat 256km×256km region of the RW product. This part also falls completely inside the area where the RADOLAN routine produces actual values. I therefore select a 256km×256km square with the upper left corner being the nearest grid point to the coordinates 53.6°N and 8.6°E. Fig. 2.6 shows the spatial extension of the selection on a map of Germany.

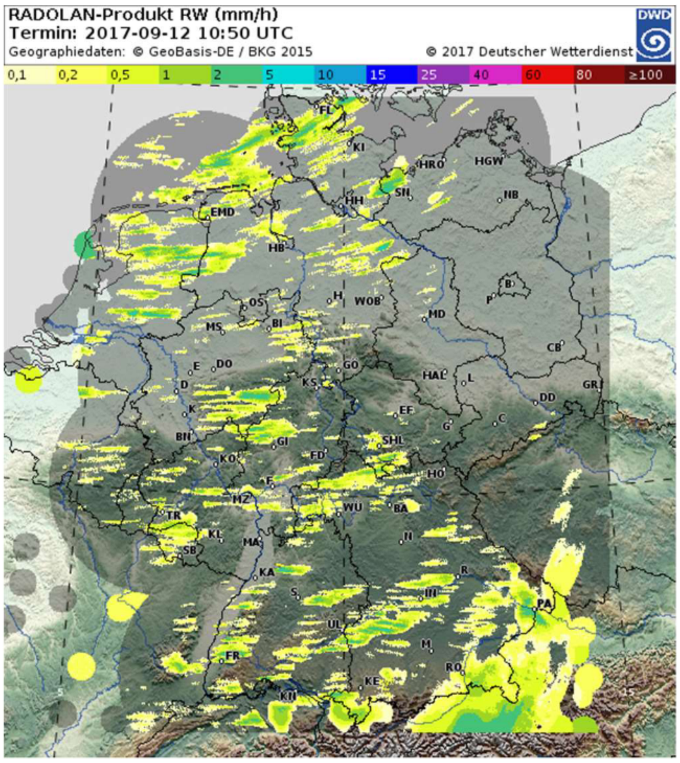
RADOLAN-Produkt RW (mm/h)
Termin: 2017-09-12 10:50 UTC
Geographiedaten: © GeoBasis-DE / BKG 2015
© 2017 Deutscher Wetterdienst
DWD
0,1 0,2 0,5 1 2 5 10 15 25 40 60 80 ≥100

Figure 2.5: Example of the RW product used in this thesis for the hourly sum at 10:50 UTC of the 12th September 2017. The values are in mm/h and the grey areas are those for which the RW composite is produced. Image reprinted from Weigl (2021b).

Period of Interest

In order to not address convective type-precipitations which typically have very short time scales, I decide to concentrate my efforts on the winter months. These go from October through April. I also decide to divide the dataset in three parts, I use the period from 1st October 2005 until 31st December 2017 as my training period. The period from 1st January 2018 until 31st December 2019 is used as the validation set and the period from 1st January 2020 until 31st December 2021 is used as the test set (cf. Section 2.2.3).

Preprocessing for Neural Network Application

The purpose of the proposed model is the prediction of a rainfall rate distribution map for the 256km×256km area of interest.

Instead of predicting the specific value of the rainfall rate for a tile of the area of interest, the study after which I based this project predicts whether the rainfall rate falls above a certain threshold (Agrawal et al. 2019). This is done for every tile and for each of the 3 thresholds 0.1mm/h, 1mm/h and 2.5mm/h.

To make the prediction more straightforward I address the problem in an equiv-

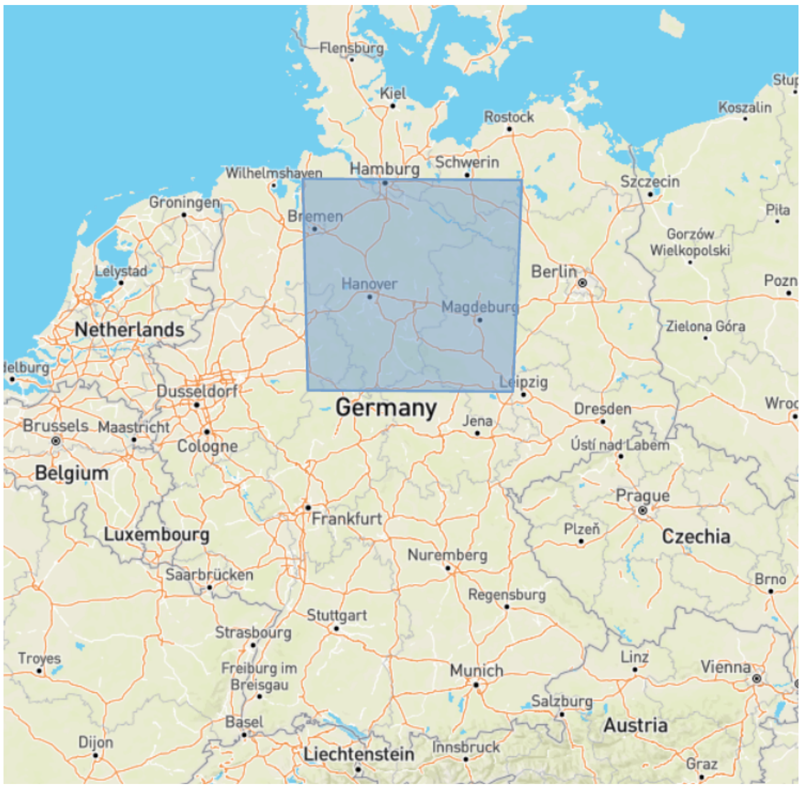

Figure 2.6: Area of interest used in this study. The coordinates of the upper left corner are those of the nearest RADOLAN grid-point to 53.6°N and 8.6°E. The other corner grid-points are obtained by going 256pixels downwards and 256pixels across. These correspond exactly to 256km southwards and 256km eastwards in the RADOLAN projection plane. The distortion present in this figure is due to the reprojection from the RADOLAN plane to the Mercator projection plane. The figure was created using openstreetmap (https://www.openstreetmap.org/) and geojson.io (https://geojson.io/).

alent way and decide to predict whether the rainfall rate belongs to one of 4 mutually exclusive classes defined using the same thresholds. These are shown in Table 2.1.

class	mm/h
0	[0, 0.1)
0.1	[0.1, 1.0)
1.0	[1.0, 2.5)
2.5	[2.5, ∞)

Table 2.1: Precipitation classes used in this study.

The problem formulated in this way is called multi-class segmentation or multi-class classification.

The model, as is expected, receives a series of inputs. The elements given as inputs to the model should contain relevant information for the model to predict the rainfall rate distribution. I decide to call these input elements **frames**. The frames are composed of 5 channels and have a 256×256 spatial extension.

The **first channel** is a 256×256 selection taken from the RADOLAN RW product (see Section 2.1.2). The values of the elements of this channel represent precipitation rate values at a precision of 0.1mm/h as described in the previous section. I call this channel the **image**.

The **second** and **third channels** are respectively the longitude and latitude of the tiles of the image.

The **fourth** and **fifth** channels contain information about the time at which the image was produced. The **fourth channel** is a normalized value for the hour and minute at which the image was taken. Its values range from -1 to 1 where 1 is 00:00 UTC and 1 is 24:00 UTC. The **fifth channel** contains a normalized day of the year at which the image was produced. It goes from -1 for the 1st of January to 1 for the 31st of December. Agrawal et al. (2019) uses only one channel for the time information and employs only the not-normalized time of the day. The choice to normalize the time information is due to the fact that this facilitates a good behaviour of the model. Deciding to add an additional channel for the time information was done because I believe it to be relevant information for the model. If it is not relevant then the model will learn to discard it (see Section 2.2).

Every input frame is referred to with the same timestamp of the image it contains. Fig. 2.7 shows a sketch of a frame and its channels.

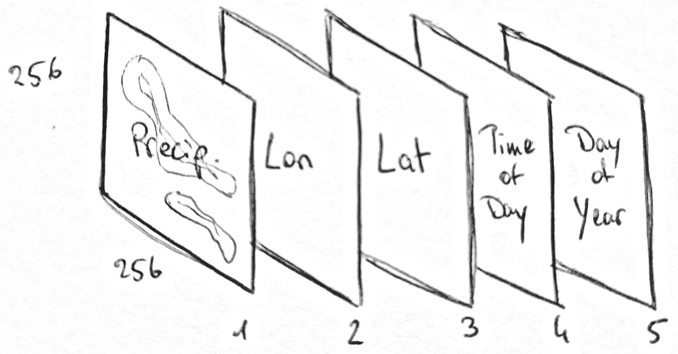

Figure 2.7: Sketch of a frame. The numbers 1 to 5 represent the channels. 256 stand for the spatial dimensions.

Finally, considering that a meteorological front has a specific speed when going through the 256km×256km area of interest, I decide to choose the number of frames to give as input to the model by approximating how much time it takes for a front to traverse the selection. Hence, with an average front speed[1] of ≈10m/s I obtain 6 input frames for the model. Therefore, the model is given 6 input frames taken at 1h intervals from timestamp t until timestamp t+5 and it predicts the class of each tile of the precipitation map for a timestamp t+6. The collection of the 6 input frames together with the ground truth classes precipitation map is called an **example** (following the general use of the term in the machine learning literature) or **scene** (to avoid confusion in this project with the common word *example*) and I will refer to it with the timestamp of the first frame.

I discard from the dataset the scenes with no rain and the scenes where one of the frames has a percentage of tiles of the image with missing precipitation values higher than 10%. If that percentage is lower I fill the missing values through interpolation. Fig. 2.8 shows a single scene and the setup as described above.

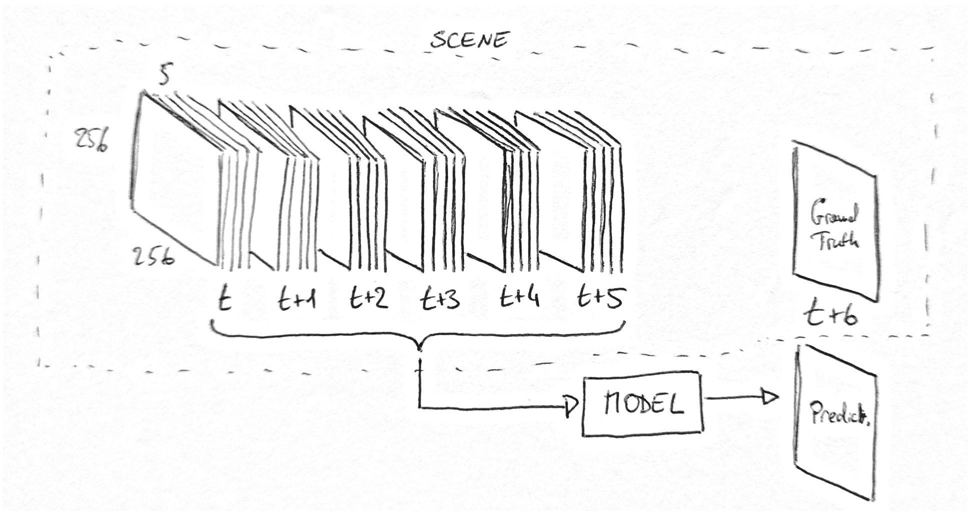

Figure 2.8: Sketch of a scene. The group of 6 frames on the left goes from time t to time t+5. All frames are at 1h distance. These are used by the model to make a prediction that is compared to the ground truth at time t+6. The frames together with the ground truth make a scene.

[1]Front speeds have very diverse values and depend on different conditions of the system.

2.2 Neural Networks

In this section I will go through the definition of *machine learning*, its subfield *deep learning* and the rationale behind it (Section 2.2.1). Next, Section 2.2.2 introduces the concepts of *task*, *performance* and *experience* upon which the machine learning algorithm of this project is built. Section 2.2.3 clarifies the differences between *training*, *validation* and *testing* and how they relate to *hyperparameters*, Section 2.2.4 in turn illustrates the concept of a neural network and its layers. Section 2.2.5 to Section 2.2.9 describe all the necessary building blocks of the *neural network architecture* employed for this thesis which is illustrated in Section 2.2.10. The subsequent sections describe the procedures with which the chosen architecture is optimized for the problem addressed in this thesis.

2.2.1 Rationale

The drive to create machines – or more specifically algorithms – that think has been with mankind from the very early days given that it was seen as challenge to recreate, in a way, artificial life. With the rise of digital computers problems that were difficult for humans but that could be written as simple formal mathematical rules were the first to be addressed. A novice would think that tasks that come natural to us, such as recognizing images or written or spoken words, would also be easy for computers. However, human brains and brains in general are very complex structures tuned and refined by millennia of evolution. It is for this reason that those tasks are some of the most difficult tasks to solve with the help of computers (Goodfellow et al. 2016).

The field responsible for finding a solution to these and more complex yet similar problems goes by the name of **artificial intelligence** (AI)[2].

The very first AI tasks to be solved at the turn of the century were tasks with easy and formal rules; the game of Chess is a notable example. IBM's Deep Blue defeated the world champion Garry Kasparov in one of humanity's most loved strategic games (Campbell et al. 2002). However, every-day life is not made of easy to describe rules like chess; an enormous amount of knowledge that we use is intuitive and subjective, it comes *naturally* to us. In order to solve these same problems a computer needs to be able to capture the same knowledge be it hard-coded or from experience. The first method has been pursued by different projects yet with unsuccessful results hence suggesting that the right path is approaching the quest by building AI systems that form their own intuitive knowledge through extraction from many examples. This ability goes by **machine learning** (ML)

[2]AI is not to be confused with the more specific **artificial general intelligence** (AGI) which is what laymen think intuitively of when hearing AI.

(Goodfellow et al. 2016). Very simple ML algorithms can solve important problems but need features of the world to be given to them explicitly. For instance, **logistic regression** can be used to recommend cesarean delivery Mor-Yosef et al. (1990) yet explicit information about the patient, like the presence of a uterine scar, have to be fed directly. These explicitly coded characteristics of the problem are also called **features**. Not every task can be solved through **feature engineering** and ideally a good AI system should be able to extract those features by itself. **Deep learning** (DL) solves the problem of extracting high-level abstract features from raw data by expressing them through simpler features and by ignoring variations that don't matter for the task at hand[3] (Goodfellow et al. 2016).

Part of the first ML algorithms that have now become prominent were thought as computational models for biological learning to study ways in which brains in general would learn. This is the reason for which DL, between other names, is also known as **neural networks** (NNs) or **artificial neural networks** (ANNs). As the name suggests these are a class of machine learning models that take inspiration from the putative workings of the brain (be it human or of another animal) (Goodfellow et al. 2016).

Hence, neuroscience gives certain guidelines on which to build upon, the main one being that computational units become intelligent via their interactions; these units have a loose resemblance to neurons. Units are organized in different ways and layers, they take inputs from various other units and produce an output that is gated by an activation function, i.e. the unit forwards the signal only if it exceeds a certain threshold (Section 2.2.9) (Goodfellow et al. 2016).

The following sections describe the various parts of a neural network and the algorithm employed in this project.

2.2.2 Problem Formulation

As I stated in the previous section, machine learning algorithms – and therefore deep learning algorithms – are programs that learn from experience. In general, quoting Mitchell (1997): "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance in task T, as measured by P, improves with experience E".

For this project the task T is: given a sequence of images as input I want to predict a subsequent image that appears in the timeseries. This is also called an **image-to-image problem**. However, for this thesis no prediction of an actual precipitation is made but of categories to which the real values of the predicted

[3]Variations that don't matter are for example the time of the day or the color of a car if the task is distinguishing between a car and a motorcycle.

pixels would belong to. Hence, this image-to-image problem can be also interpreted as a **multi-class classification** problem. In fact, for all purposes it is treated as a multi-class classification where the targets are derived by binning the ground truth image. More concisely: for a timestamp t I give as inputs 6 radar-precipitation images embedded into frames from t to $t+5$ (cf. Section 2.1.3) to predict the probabilities that a single pixel belongs to one of the precipitation classes of the image at $t+6$. This probability reads, formally

$$Pr(R_{t+6}|F_t, ..., F_{t+5}) \tag{2.11}$$

F_t represents a frame at time t and R is the binned rainfall rate target part of the scene associated to the frames F. The outputted probability is computed for the 4 mutually exclusive classes for every tile of the 256x256 image.

The performance measures P are the cross-entropy loss, the precision, the recall, the F-measure, the Critical Success Index and the Heidke Skill Score as defined in Section 2.3.

The experience E is the RADNET dataset as defined in Section 2.1.3. It is made of scenes which are obtained from the ordered series of radar precipitation images of the RADOLAN dataset.

2.2.3 Dataset Division and Hyperparameters

In general one is interested to know the goodness of the model on new data. Hence, the performance measures P of the previous sections are normally evaluated on a **test set**. This subset of the dataset is separate from the data used for training the model which is, in turn, called the **training set** (Goodfellow et al. 2016).

Machine learning algorithms, in general, posses so-called **hyperparameters**. These are user-controlled settings that govern the behaviour of the algorithm but are not learned. Nonetheless, algorithms exist to find the optimal hyperparameters set (cf. Section 2.2.14). Frequently, parameters that could in theory be optimized on the training set are used as hyperparameters because their optimization would lead to overfitting (see Section 2.2.12). To address this issue, one further breaks down the training set in **training set** and **validation set**. Although the nomenclature overlaps with the former interpretation of *training set* when hyperparameters need optimization a *validation set* is needed. This subset of the dataset is used to learn the hyperparameters whilst the *test set* is left completely untouched as it has to serve its role in estimating the generalization error of the model (Goodfellow et al. 2016). A typical ratio of the *training* and the *validation set* is 8:2 but if the dataset is large enough the amount in the *validation set* can be reduced.

2.2.4 Neural Networks and Layers

Neural networks are machine learning algorithms that come with different characteristics. In this project a so-called feed-forward neural network is employed in which the information flows only in one direction from the input to the output without coming back to itself with feedback connections.

The structure of a neural network has at its core a fundamental element called unit. Units are vector-to-scalar functions that precisely resemble a neuron's workings because they take inputs from multiple other sources (neurons) and are activated to a different degree based on these weighted inputs (Goodfellow et al. 2016). Multiple neurons acting in concert after the same number of connections from the input are organized in a so-called layer where the input units form the input layer and the output units form the output layer. Additional layers that fall in between the input and the output layers are called hidden layers and their presence renders a neural network deep[4] (Goodfellow et al. 2016). The depth of the network is given by the number of layers, while its width corresponds loosely to the number of neurons composing a layer.

Finally, another parallelism with neuroscience is drawn by using activation functions. These functions are non-linear and differentiable[5] and represent a threshold that the input signal to a unit has to overcome in order to activate said unit (see Section 2.2.9).

Neural networks come in different shapes and employ different layers and components; the specific set of layer choices and other characteristics that make the network are collectively known as the **architecture** of the network.

The next sections contain the building layers and other architectural characteristics of the neural network used in this thesis up until Section 2.2.10 where the architecture is presented to its fullest.

2.2.5 Fully Connected Layers

One of the conceptually simplest architecture for a layer in a neural network is built by connecting every unit in a layer to every unit in the layer directly after. Each connection has an associated *weight* and *bias* term that modulate the activation of a single unit based on the signals coming from the units before it (Bishop 2006).

The mathematical expression for a hidden layer would be through a linear com-

[4]The term deep learning comes, in fact, from Deep Neural Network.

[5]Activation functions must only be differentiable in the regions where a gradient for the optimization procedure must be computed. Functions that are normally utilized, such as the Rectified Linear Unit, are not differentiable in some points. This problem is usually of no concern since the probability that the function is evaluated at its non-differentiable point is negligible. Even if this were to happen, the issue could be defined away by simply taking the right derivative.

bination of its inputs like:

$$\boldsymbol{h} = g\left(\boldsymbol{W}^{\top}\boldsymbol{x} + \boldsymbol{b}\right) \tag{2.12}$$

where $\boldsymbol{W}$ is the weights Matrix, $\boldsymbol{x}$ the input vector, $\boldsymbol{b}$ the bias vector and g a differentiable non-linear activation function (Bishop 2006; Goodfellow et al. 2016). An example of which is the common function $g(z) = \max\{0, z\}$ also known as Rectified Linear Unit (ReLU). More in Section 2.2.9. A single layer is rarely a good solution and invariably one finds itself stacking multiple of these fully connected layers one after the other (Goodfellow et al. 2016). When multiple layers are employed then the architecture becomes mathematically

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \tag{2.13}$$

for the first layer. Changing this with the second layer produces

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right) \tag{2.14}$$

The subsequent layers are straightforward as is the output layer:

$$\boldsymbol{y} = g^{(n)}\left(\boldsymbol{W}^{(n)\top}\boldsymbol{h}^{(n-1)} + \boldsymbol{b}^{(n)}\right) \tag{2.15}$$

where n is the number of *hidden layers* in the network (Goodfellow et al. 2016).

2.2.6 Convolutional Layers

Convolutional layers as an architectural solution are employed when the data has a known grid-like topology as is the case for 1-D grids like time-series or 2-D grids like simple pictures. The intuition underpinning this architecture is that spatially (or temporally) adjacent data-points are highly correlated. A neural network is said to be **convolutional** if it employs at least one convolutional layer instead of the vanilla matrix multiplication. The name convolution comes from the mathematical operation involved in this type of layers, i.e. **convolution** (Goodfellow et al. 2016).

Most generally convolution is a mathematical operation on two functions with real-valued argument. It's intuitive interpretation is the weighting average of a function x by a weighting function w. The mathematical definition hereof is the following intergral:

$$s(t) = \int x(a)w(t-a)\mathrm{d}a. \tag{2.16}$$

In order for the output to be a weighted average w needs to be a valid probability density function. However convolution is defined for more general situations and it is a valid operation as long as the integral in Eq. (2.16) is defined.

Convolution has also its specific mathematical notation:

$$s(t) = (x * w)(t). \tag{2.17}$$

In the case of convolution x is called the **input**, w the **kernel** and the output is called the **feature map**.

The functions in Eqs. (2.16) and (2.17) are continuous and are therefore not a reasonable representation of real-world cases. Real-world data is, unsurprisingly, given in discrete intervals like for instance small timesteps in a time-series or different pixels in an image. One can obtain the discrete version of the convolution operation by employing a discrete sum in place of the integral:

$$s(t) = (x * w)(t) = \sum_a x(a)w(t-a)\Delta a. \tag{2.18}$$

with Δa to be interpreted as a weighting window (Goodfellow et al. 2016; Bishop 2006). Both the input and the kernel are normally multidimensional arrays (sometimes also called **tensors**) and the values of the latter are learned through the learning algorithm. The fact that both have to be stored justifies the reasonable assumption that the functions are 0-valued everywhere on the domain upon which they are defined expect where their values are stored (Goodfellow et al. 2016) making the summation finite.

In real-world DL applications convolution is applied on multi-dimensional tensors and on multiple axis. The simplest case arises for 2-D tensors like images, therefore the 2-dimensional version of Eq. (2.18) with an image I as input and a 2-dimensional kernel K is

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.19}$$

that, given the commutativity of the convolution operation is equivalent to

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, i-n)K(m,n) \tag{2.20}$$

The commutativity of convolution is due to the fact that the kernel is flipped with respect to the input, i.e. when indices increase in one they decrease in the other. Modern libraries, however, implement what is called **cross-correlation**. This operation is closely related to convolution and in fact it's the same without a kernel-flip (Goodfellow et al. 2016). Its mathematical expression reads

$$S(i,j) = (K \star I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n); \tag{2.21}$$

Note the star instead of the asterisk for the cross-correlation. The learning algorithm will learn the values of the appropriate values of the kernel. The kernels resulting from a neural network using cross-correlation will be flipped with respect to the values learned if the implementation is a convolution (Goodfellow et al. 2016). Fig. 2.9 shows a case of a 2-D discrete convolution without Kernel flipping.

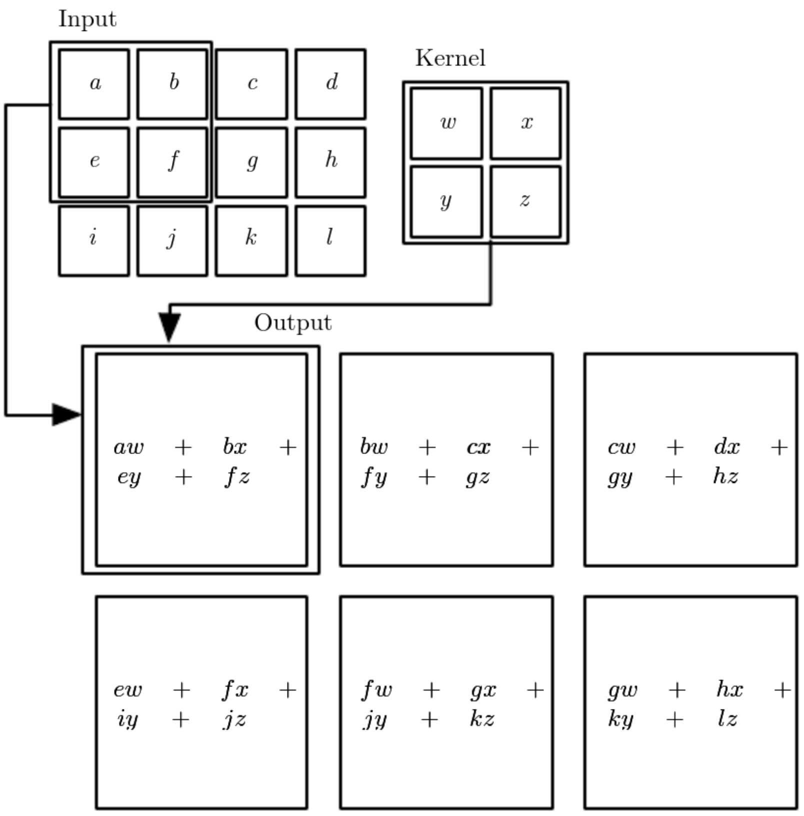

Figure 2.9: 2-D convolution without kernel flipping. The ouptput is restricted to the places where the kernel fits entirely on the input. The arrows indicate how the indicated output element is obtained from the combination of the selected region of the input tensor and the kernel tensor. Image taken from Goodfellow et al. (2016).

Since the appropriate values are the subject of learning by the algorithm, the kernel-flip is irrelevant.

Advantages

As it can be intuited from Fig. 2.9 and as already stated, elements of a tensor that are near each other will influence the corresponding output more than elements that are far from each other. The three main advantages that arise from how convolution is built are sparse interactions, parameter sharing and equivariant representations.

Sparse interactions

Fully connected layers (Section 2.2.5 and Eqs. (2.13) to (2.15)) work through a matrix by matrix multiplication of equal dimensions. Sparse interactions in convolution arise due to the kernels being much smaller than the input matrix, for instance edges in an image of $\sim 10^6$ pixels can be easily detected with kernels of $\sim 10^2$ pixels (Goodfellow et al. 2016). Smaller kernels mean fewer values to store and increased efficiency[6].

Parameter sharing

During a simple matrix multiplication, i.e. the fully connected case, each and every element of the weight matrix is used only once. In contrast, during convolution each parameter of the kernel matrix is used multiple times as the kernel shifts through the input grid. The boundaries of the image are particular in this regard but depend on implementation choices. The runtime is not affected by this property since the number of operations is still $O(k \times n)$ where k is the number of elements in the kernel and n the number of elements of the input. However, the memory footprint is affected substantially given that the number of weights to be stored is now k instead of $m \times n$, which is the dimension of the weight matrix in the fully-connected case (Goodfellow et al. 2016). Fig. 2.10 depicts the workings of parameter sharing.

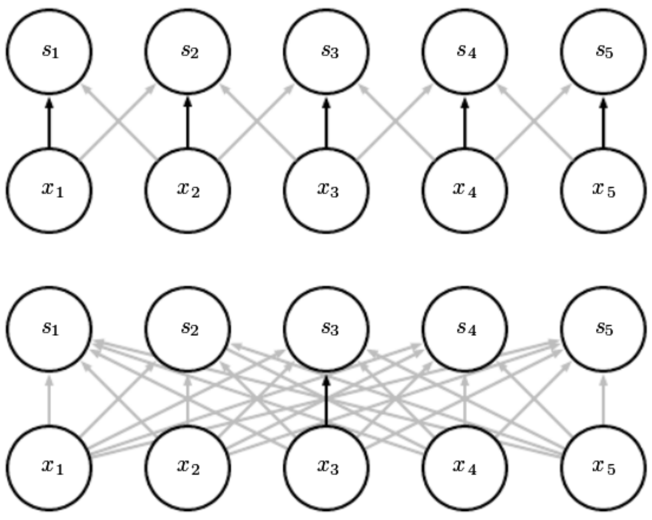

Figure 2.10: The top graph is a representation of a convolution with kernel made of $k = 3$ elements and an input $\boldsymbol{x}$ of $n = 5$ elements. The black arrows are associated with the central element of the kernel. The bottom graph shows the fully-connected case. Here the black arrow represents the central element of the $n \times m = 5 \times 5$ weight matrix $\boldsymbol{W}$. Image taken from Goodfellow et al. (2016).

Equivariance to translation

The property of equivariance to translation is a consequence of the parameter sharing property: given that a kernel is applied everywhere on the input, the features

[6]Considering m inputs and n outputs the simple matrix multiplication has $m \times n$ parameters, $O(m \times n)$ runtime for every example. However, with convolution limiting the number of connections to k for each input, with a much smaller k, the number of parameters and the computing time decrease to $k \times n$ and $O(k \times n)$ respectively (Goodfellow et al. 2016).

it detects can be detected anywhere. The term equivariance comes form the mathematical equivariance of a function with respect to another function. A function f is said to be equivariant to a function g if and only if $f(g(x)) = g(f(x))$. If g is a function that shifts the input by some distance, we have the case of the equivariance to translation of features to be detected by the kernel f (Goodfellow et al. 2016).

Indifference to input size

One last advantage of convolution is its indifference to the input's size. This is a trivial consequence of how kernels are applied to the input. Fully connected layers always need the same input dimensions given that the weight matrix $\boldsymbol{W}$ has a fixed size (Goodfellow et al. 2016).

2.2.7 Pooling and Unpooling

Pooling layers are typically employed after a convolution layer and consist of units that substitute an area around a specific location with a summary value such as a maximum (max pooling) or an average (average pooling) LeCun et al. (2015); these two are the most frequently used pooling functions. Other functions exist[7] and are sometimes employed but are not described for simplicity. Given that the averaging and maximum functions are parameter-free no learning occurs in these layers.

Given that a summary statistic is produced over a range of pixels, the pooling areas can be spaced by k pixels instead of just 1. This, in turn, reduces the number of subsequent units by a factor of approximately k. An example of this mechanism is shown in Fig. 2.11.

Summary operations make the network approximately invariant to small translations of the input giving more importance to the approximate location and existence of a feature than its precise location in the input grid (Goodfellow et al. 2016).

As shown in Fig. 2.11 the pooling operation with a stride makes the output height and width smaller than the input dimensions. Given that for the architecture employed in this project it's essential for the output to be of the same dimensions as the input (see Section 2.2.10) so-called unpooling layers are used. These layers neither add, nor remove information. When the unpooling operations are done through an *upsampling to the nearest neighbour*, every original pixel is substituted with a new $k \times k$ matrix where every element of the new matrix is equal to the value of the pixel it originated from.

[7]Some of these functions are L^2 norm pooling and distance-weighted average pooling

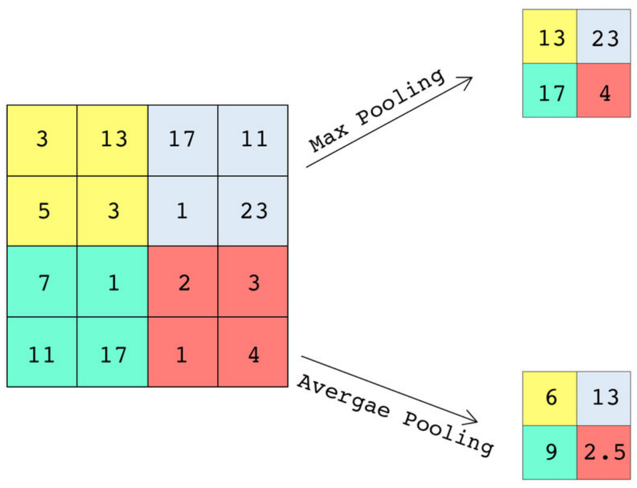

Figure 2.11: Comparison between average pooling and max pooling of a 4×4 tensor. The different colors identify the 2×2regions over which the average is computed. As is common for pooling the kernel has a stride, i.e. it jumps, by its width instead of going through every pixel. Image taken from Aljaafari (2018).

2.2.8 Skip Connections

Generally more connections are thought to add additional degrees of freedom to a network that give it the possibility to fit much more intricate functions. An hypothesis connected to this seemingly correct assumption is that if errors would arise from adding more layers they would manifest in the form of overfitting. Further, if a layer is not really needed, the network should in theory be able to fit an identity mapping with the non-beneficial layers. However, this has been shown to not be the case in He and Sun (2015) and in Srivastava et al. (2015) where additional layers lead to degradation not due to overfitting.

As in He et al. (2016a); Srivastava et al. (2015), the problem was identified to be a difficulty by the layers to allow a flow of unaltered information to pass through the network. The simplest solution is given by fitting a residual mapping for smaller building blocks of the network instead of the general mapping. This adds no computational cost and is a proven method to address degradation in deeper networks effectively (He et al. 2016a). Formally that would amount to fitting $\mathcal{F}(\boldsymbol{x}) = \mathcal{H}(\boldsymbol{x}) - \boldsymbol{x}$ and obtaining the desired mapping $\mathcal{H}(\boldsymbol{x})$ by simple summation $\mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}$. Hence, in the extreme case, to obtain the identity mapping, the weighted layers would only need to reach 0. Particular implementations of this fundamental idea were further discussed by He et al. (2016b). For a graphical representation see Fig. 2.12.

In the case of U-Nets used in this project (see Section 2.2.10 for the implementation), it is good practice to adopt long range skip connections to pass information

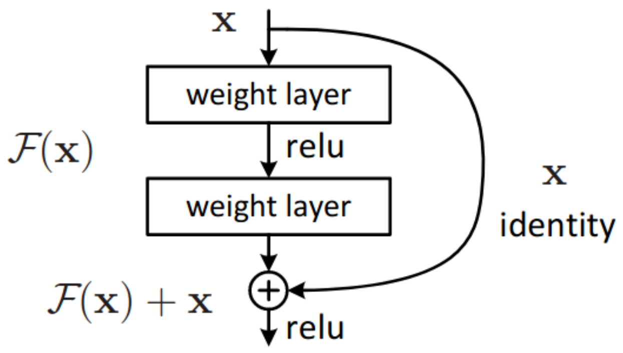
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.12: The representation of a residual block that fits a desired mapping $\mathcal{H}(\boldsymbol{x})$ by making the tunable (weight) layers optimise for the residual $\mathcal{F}(\boldsymbol{x}) := \mathcal{H}(\boldsymbol{x}) - \boldsymbol{x}$. "relu" stands for the correpsonding activation function (cf. Section 2.2.9). Example and image taken from He et al. (2016a).

from a downsampling[8] block to an upsampling[9] block (Drozdzal et al. 2016). The use of long-range and short-range skip connections for image segmentation was linked to both better performance and faster convergence (Drozdzal et al. 2016). Both kinds of skip connections can be achieved either by summation – therefore applying He et al. (2016a) also to long-range skip connections – or by concatenating the relevant information as in Shelhamer et al. (2017).

Further details on the specifics of the implementation in this project will be discussed in Section 2.2.10.

2.2.9 Activation Functions

Activation Functions are an essential building block of neural networks due to the fact that they introduce a necessary **non-linearity** in the architecture. Were they linear then the neural network would become a combination of only linear components which is equivalent to an architecture without hidden layers (Bishop 2006). There is no canonical choice of activation functions, the only requisites are that they be (1) non-linear and (2) differentiable (Bishop 2006; Goodfellow et al. 2016). The appropriate activation function for hidden layers is normally chosen by looking at its properties like ease of computation and lack or presence of saturation. However, for the output layer the choice of the activation function and the corresponding *loss function* are tightly coupled to the problem being solved (Bishop 2006).

Many activation functions have risen in popularity; the three most common for hidden layers to the day of writing according to Papers with Code (2022) are

[8]See Section 2.2.10
[9]See Section 2.2.10

the Rectified Linear Unit, the Sigmoid activation function and the Tanh activation function. I will address only a variant of the first one: the Leaky Rectified Linear Unit. As for the activation function for the output unit – which are sometimes called output functions – the Softmax activation function is presented.

Leaky Rectified Linear Unit (LeakyReLU)

The LeakyReLU activation function has nearly all the properties of the ReLU activation function of which it is a variation (Maas et al. 2013), i.e. no saturation, ease of computation of the gradient and avoidance of the vanishing gradient problem. The only difference is that it is not zero for negative values but linear with a positive – yet small – slope. Formally it's defined as,

$$LeakyReLU(x) = \begin{cases} x & x \geq 0 \\ a\,x & \text{else} \end{cases} \quad \text{with } a \in \mathbb{R}^+ \tag{2.22}$$

The value of a in Eq. (2.22) is traditionally small and in the order of 0.01 however it can be easily set manually and reach higher values (Xu et al. 2015).

The implication of using the LeakyReLU activation function is that there is a gradient, albeit small, for negative values of the input which allows for updating to take place during back-propagation. Computationally this is very efficient compared to alternatives such as *sigmoid* and *tanh*.

Softmax

The Softmax activation function is normally used in the very last layer of a neural network – also called the output layer – to address multi-class classification problems. For problems of this sort one needs to produce a vector $\hat{\boldsymbol{y}}$ with

$$\hat{y}_i = P(y = i \mid \boldsymbol{x}) \tag{2.23}$$

where every element is between 0 and 1 and where the sum of the elements is 1 therefore representing a valid probability distribution. In the practical application of a neural network the last layer before the output layer, being linear, produces an unnormalized log probability[10] vector as in

$$\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b} \tag{2.24}$$

with

$$z_i = \log \tilde{P}(y = i \mid \boldsymbol{x}). \tag{2.25}$$

[10]More on log probabilites and their relevance in Section 2.2.11

This corresponds to the unnormalized log probability that the input $\boldsymbol{x}$ belongs to the class i.

The definition of the Softmax function comes from the need to obtain a normalized probabilty distribution $\hat{\boldsymbol{y}}$ by exponentiating and normalizing $\boldsymbol{z}$

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp z_i}{\sum_{j=1}^{n} \exp z_j} = \hat{y}_i \tag{2.26}$$

with $i = 1, ..., n$ (Goodfellow et al. 2016). This in turn corresponds to the normalized probability that the input $\boldsymbol{x}$ belongs to the class i.

In the case of image segmentation, as is for this project, for every pixel of the output image there is an associated $\boldsymbol{z} \in \mathbb{R}^n$ on which the Softmax function can be applied to categorize every pixel.

2.2.10 U-Net encoder-decoder

The task of semantic segmentation, that is the assignment of classes to pixels that belong to the same group, is an ubiquitous problem in the machine learning literature and goes from applictaions such as scene recognition (Badrinarayanan et al. 2017) to identifications of neuronal structures in biomedical images as in Ronneberger et al. (2015). The architecture in Ronneberger et al. (2015) – a U-Net – was proposed as an extension of of the first fully convolutional network architecture for semantic segmentation by Shelhamer et al. (2017).

Implementation

An *encoder-decoder* main working is to (1) reduce the spatial dimension of the input and abstract information into a latent space, subsequently (2) return the information to the original format so that it can be interpreted (Goodfellow et al. 2016). This can be done through convolutional layers and, if nothing other than convolution is involved, this architectural choice can be labeled as *fully convolutional encoder-decoders*. A specific implementation thereof is the U-Net architecture (Ronneberger et al. 2015) adopted and re-adapted in this project for the image-to-image problem applied to precipitation nowcasting and inspired from Agrawal et al. (2019). Fig. 2.63 shows the architecture I employ as described in the following paragraphs.

Encoder

The encoder starts with a *basic block* that is composed of a *convolution*, followed by a *batch normalization* layer, a non-linear activation function of the *LeakyReLU* type and an additional *convolution*.

Subsequently a number of *down sampling blocks* follow. Each block is made of, in order, a *batch normalization*, a *Leaky ReLU*, a *max pooling* layer – where the actual downsampling occurs – another *batch normalization* layer, a *Leaky ReLU* and a final *convolution*.

After the *down sampling blocks* of the *encoder* and before the *decoder* I employ an additional *basic block* that serves as a bottleneck.

For the short range skip connections' architecture there are no explicit descriptions in Agrawal et al. (2019) other then a reference to Drozdzal et al. (2016) and He et al. (2016a). Since both prefer summation rather than concatenation for skip connections, I choose the former. As for the layers in the path followed by the identity mapping, keeping in line with He et al. (2016b) and Drozdzal et al. (2016) I use a very simple *average pooling* for the reduction of the spatial resolution and a *1x1 convolution* to adjust the number of feature maps so that a summation is possible after a skip connection. I also add a *batch normalization* layer to mitigate the batch-to-batch variance for small batches.

After every block in the encoder – and before the summation with the identity mapping – the intermediate result is stored to be used in a long-range skip connection to the corresponding block in the decoder. Agrawal et al. (2019) does not explicitly state whether the long range skip connection is built by taking the output of an encoding block right before the identity mapping is added, however, an image of the architecture in a related blog post[11] made by the authors suggests this to be the case.

Decoder

The structure of the *decoder* is similar to the one of the *encoder*. However, an important difference exists: for the purpose of increasing the spatial dimensions it has *upsampling* layers instead of a *max pooling* layers.

The composition of the *upsampling blocks* is: a *nearest neighbor upsampling* layer, a *batch normalization* layer, a *LeakyReLU* activation layer, a *convolution* and again *batch normalization*, *LeakyReLU* and *convoluion*.

The *short skip connections* are implemented for the *decoder* around every *upsampling block*. Compared to their counterparts in the encoder section, these skip connections employ the same layers for the same purposes but with *average pooling* substituted by a *nearest neighbor upsampling* layer. Table 2.3 shows the feature maps for the blocks described in the last two sections.

2.2.11 Network Optimization and Training Algorithm

In this section I will address the principal framework – called **maximum likelihood estimator** – through which a set of neural network parameters $\boldsymbol{\theta}$ are found such

[11]https://ai.googleblog.com/2020/01/using-machine-learning-to-nowcast.html

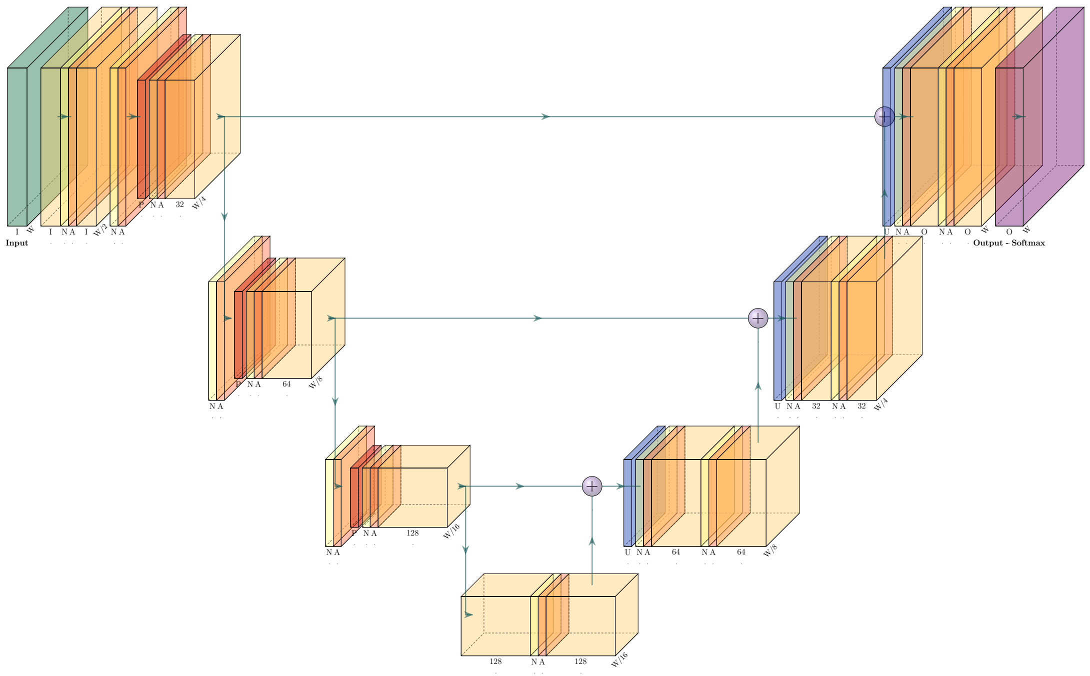

Figure 2.63: Architecture of the U-Net used for this project as in (Agrawal et al. 2019). The name comes from the shape it has when displayed in layers. This specific example has the depth hyperparameter – that controls the number of layers – set to 3. The left part of the network is called the *encoder* whilst the right part is the *decoder*. The thicker and fainter yellow layers are convolutions with a value standing for the number of feature maps. The number of feature maps is also another hyperparameter, which in this case, is 32. For every layer the number of feature maps doubles. The number of feature maps of the convolutions of the last part of the decoder are denoted with the letter O. In my case they are equal to the number of classes of the output. The bright yellow layers are *batch normalization* layers and are denoted with the letter N. The orange layers stand for the activation function, with the letter A, which in my case are LeakyReLUs. The red layers are *max pooling* operations that reduce the spatial dimensions. The blue layers represent *nearest neighbour upsampling* and invert the pooling process. The arrows represent the flow of information; the ones connecting the blocks from the *encoder* to the sister blocks of the *decoder* are the long skip connections. The skip connections are implemented as summations. The Softmax output layer, depicted in purple, has a number of output channels equal to the number of classes. Short-range skip connections are not depicted for simplicity; they would skip every major block. In this picture every block is separated from other blocks by arrows. Table 2.2 contains details on the various parameters used for the differente layers, included short-range skip connections.

that they minimize an error measure called the *loss function*, i.e. the deviation of the estimate from the target data-points. I will also discuss how this framework allows us to obtain an appropriate loss function given a certain problem. Moreover, the actual

U-Net Building Blocks			
Layer name	Mathematical operation	Hyperparameter	Value
Conv2d	Convolution	kernel size	3
		stride	1
		padding	1
		dilation	1
		#feature maps	cf Table 2.3
BatchNorm2d	Batch Normalization	momentum	0.1
LeakyReLU	Leaky ReLU	negative slope	0.01
MaxPool2d	Max Pooling	kernel size	2
		stride	2
Upsample	Upsampling	scale factor	2
		mode	nearest neighbor
Softmax	Softmax Eq. (2.26)	N/A	N/A

Table 2.2: An overview of the different layers that are used as building blocks for the architecture of this project. The name used is the same as the one that appears in the machine learning library *pytorch* in order to favor comparability. An account on how these blocks are put together to form the final architecture is given in both Fig. 2.63 and Table 2.3.

training algorithm and its components as well as the particular implementation used in this project are shown.

Maximum Likelihood Estimation

The maximum likelihood principle is a method that allows to determine the appropriate functions that make good estimators for a specific model, i.e. it indicates a clear loss function to minimize (Goodfellow et al. 2016). In simple terms (1) a model must be assumed (2) from that model the likelihood function is derived – that is the probability that the model results in the given data (3) the likelihood function is maximised (or something equivalent).

Let $\mathbb{Y} = \{\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(m)}\}$ be the family of all the observed targets and $\mathbb{X} = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}\}$ representing all the m inputs drawn from the distribution. I call the tuple $(\boldsymbol{y}^{(i)}, \boldsymbol{x}^{(i)})$ an example. I also define $\boldsymbol{Y} = (\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(m)})$ and $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)})$ to be the input and output tensors respectively.

Let us have a family of model probability distributions with parameters $\boldsymbol{\theta}$

$$p_{\text{model}}(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) \tag{2.27}$$

for the variables $\mathbf{y}$ and $\mathbf{x}$ with images being respectively $\mathbb{Y}$ and $\mathbb{X}$. I use this family

Block architectures			
Block name	Layer name	Input feature maps	Output feature maps
Basic block	Conv2d	C	C
	BatchNorm2d	C	C
	LeakyReLU	C	C
	Conv2d	C	C
Downsampling block	BatchNorm2d	C	C
	LeakyReLU	C	C
	MaxPool2d	C	C
	BatchNorm2d	C	C
	LeakyReLU	C	C
	Conv2d	C	2C or IF for first block
Down skip connection	1x1Conv2d	C	2C
	AvgPool2d	2C	2C
	BatchNorm2d	2C	2C
Upsampling block	Upsample	C	C
	BatchNorm2d	C	C
	LeakyReLU	C	C
	Conv2d	C	C/2
	BatchNorm2d	C/2	C/2
	LeakyReLU	C/2	C/2
	Conv2d	C/2	C/2 or K for last layer
Up skip connection	Upsample	C	C
	1x1Conv2d	C	C/2
	BatchNorm2d	C/2	C/2 or K for last layer
Final layer	Softmax	K	K

Table 2.3: Summary of the number of feature maps associated with each of the layers of the different blocks for the architecture of this project. C is a place holder value for which to compute the other number of feature maps for every layer of the same block. IF stands for the hyperparameter Initial Features which is the number of feature maps of the output of the first downsampling block. K is the number of classes of the task. In this project K=4.

to estimate the probability distribution

$$p_{\text{data}}(\mathbf{y}|\mathbf{x}). \tag{2.28}$$

The likelihood function[12] is given by

$$\mathcal{L}(\boldsymbol{\theta}|\mathbf{y};\mathbf{x}) = p_{\text{model}}(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \tag{2.29}$$

which for the case of i.i.d.[13] examples means

$$\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{Y};\boldsymbol{X}) = p_{\text{model}}(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}) = \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}). \tag{2.30}$$

one then has the definition of **maximum likelihood estimator** for $\boldsymbol{\theta}$ as in Goodfellow et al. (2016). It reads

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}) \tag{2.31}$$

and by assuming that the examples are i.i.d. one can rewrite this expression as

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}). \tag{2.32}$$

However, Eq. (2.32) has a clear issue when implemented numerically since probabilities have values $0 \leq p_{\text{model}}(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \leq 1$ and their product can become really small causing numerical underflow. To facilitate the numerical optimization one can use a monotonically increasing function and obtain an equivalent numerical optimization problem. The function commonly used is the logarithm that converts a product into a sum. One obtains log likelihoods instead of likelihoods:

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}). \tag{2.33}$$

Conveniently a sum of logarithms for small p_{model} will reach negative values instead of the very small but positive values had it been a product (Goodfellow et al. 2016).

I will now show how maximizing the likelihood is equivalent to a minimization of the so-called cross-entropy.

Cross-Entropy

This subsection is written after Cover and Thomas (2006) – where a more precise account on information-theory can be found – and Goodfellow et al. (2016).

One first defines a measure of the uncertainty of a random variable, this measure is normally called **Shannon entropy**. For a discrete random variable x with probability mass function $p(x)$ it reads as follows

$$H(\mathrm{x}) = -\sum_{x} p(x) \log p(x) \tag{2.34}$$

[12]See Rossi (2018) for a precise account.
[13]Independent and identically distributed.

which can be interpeted as the expected value of a random variable $\log \frac{1}{p(\mathrm{x})}$ with x having a probability mass function $p(x)$. One obtains

$$H(\mathrm{x}) = \mathbb{E}_{\mathrm{x}\sim p}\left[\frac{1}{\log p(\mathrm{x})}\right] = -\mathbb{E}_{\mathrm{x}\sim p}[\log p(\mathrm{x})], \tag{2.35}$$

which is also sometimes written as $H(p)$.

As stated, the entropy of a random variable measures the amount of information needed to describe the random variable. A related concept is the relative entropy, which is a measure of the dissimilarity between two probability distributions.

Let $p(x)$ and $q(x)$ be two probability mass functions. Their relative entropy or Kullback-Leibler divergence is defined as

$$D_{\mathrm{KL}}(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{2.36}$$

which, in a similar fashion to entropy it can be interpreted as the expected value of a random variable equal to $\log \frac{p(\mathrm{x})}{q(\mathrm{x})}$ where x follows a distribution p. This means

$$D_{\mathrm{KL}}(p||q) = \mathbb{E}_{\mathrm{x}\sim p}\left[\log \frac{p(\mathrm{x})}{q(\mathrm{x})}\right] = \mathbb{E}_{\mathrm{x}\sim p}[\log p(\mathrm{x}) - \log q(\mathrm{x})]. \tag{2.37}$$

The KL divergence, although sometimes referred to as a *distance* between two distributions, is not really a distance because it's not symmetric, i.e. $D_{\mathrm{KL}}(p||q) \neq D_{\mathrm{KL}}(q||p)$.

The cross-entropy between two distributions p and q is similar to the KL divergence and is defined as

$$H(p, q) = H(p) + D_{\mathrm{KL}}(p||q) \tag{2.38}$$

hence, by virtue of Eqs. (2.35) and (2.37)

$$H(p, q) = -\mathbb{E}_{\mathrm{x}\sim p}[\log q(\mathrm{x})] \tag{2.39}$$

which is equivalent to

$$H(p, q) = -\sum_x p(x) \log q(x). \tag{2.40}$$

It's clear from Eq. (2.38) that minimizing the cross-entropy with respect to q is equal to minimizing $D_{\mathrm{KL}}(p||q)$ with respect to q.

Let $p(\boldsymbol{x})$ be the empirical distribution $\hat{p}_{data}(\boldsymbol{x})$ as implied by the dataset and defined as the relative frequency of $\boldsymbol{x}$ in a set of n samples of the type $\boldsymbol{x}^{(i)}$, that is:

$$\hat{p}_{\mathrm{data}}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \delta_{\boldsymbol{x}}(\boldsymbol{x}^{(i)}) \tag{2.41}$$

and let $q(\boldsymbol{x})$ be the model distribution $p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta})$. By inserting the newly defined p and q in Eq. (2.40) one obtains

$$H(\hat{p}_{data}(\boldsymbol{x}), p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta})) = -\sum_{x} \hat{p}_{data}(\boldsymbol{x}) \log p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta}) \tag{2.42}$$

$$= -\frac{1}{n}\sum_{i=1}^{n} \log p_{\text{model}}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \tag{2.43}$$

which applied to the supervised-learning case and compared to Eq. (2.33) and allows us to conclude that the maximization of the likelihood is equivalent to the minimization of the cross-entropy between the model distribution and the empirical distribution. This reads

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \tag{2.44}$$

$$= \arg\min_{\boldsymbol{\theta}} -\frac{1}{m}\sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \tag{2.45}$$

$$= \arg\min_{\boldsymbol{\theta}} H(\hat{p}_{data}(\boldsymbol{x}), p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta})). \tag{2.46}$$

Since in general one wants to minimize a so-called **cost function**, the maximum-likelihood principle gives us a natural cost function for a certain model distribution. This cost function is sometimes referred with the symbol $J(\boldsymbol{\theta})$ and is simply the negative of the log-likelihood of the model, equivalent to the cross-entropy.

In the next section I will derive the expression for the log-likelihood for the multi-class classification problem.

Maximum likelihood for the multi-class classification problem

The maximum likelihood principle can be easily applied to the multi-class classification problem. For this family of problems one main assumption is needed: that the target vector $\boldsymbol{y}_i \in \mathbb{R}^K$ for the i-th example follows a categorical distribution with K-classes that are mutually exclusive. The distribution has the following form

$$p(\boldsymbol{y}_i|\boldsymbol{z}_i,\boldsymbol{\theta}) = \prod_{c=1}^{K} \pi(\mathbf{y}_i = \mathbf{1}_c|\boldsymbol{z}_i,\boldsymbol{\theta})^{y_{ic}} = \prod_{c=1}^{K} \pi_{ic}^{y_{ic}} \tag{2.47}$$

with the single modelled probabilities $\pi_{ic} = \pi(\mathbf{y}_i = \mathbf{1}_c|\boldsymbol{z}_i,\boldsymbol{\theta})$ that, given the i-th feature vector $\boldsymbol{z}_i$ and the model parameters $\boldsymbol{\theta}$, the i-th element belongs to class $c \in \{1, ..., K\}$ as encoded in a one-hot vector $\mathbf{1}_c$ – a vector with one element equal to 1 and all other elements equal to 0 – of length K (Goodfellow et al. 2016; Bishop

2006). Also, we have the target vector $\boldsymbol{y}_i = (y_{i1}, ..., y_{ic}, ..., y_{iK})$ being too a one-hot vector with 1 in the position c corresponding to the actual class of the example.

The likelihood for a categorical distribution with m examples, following Eq. (2.30), reads

$$\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{Y};\boldsymbol{X}) = \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) = \prod_{i=1}^{m}\prod_{c=1}^{K} \pi_{ic}^{y_{ic}} \tag{2.48}$$

and since on wants to minimize a cost function $J(\boldsymbol{\theta})$, that is defined as the negative of the log-likelihood according to the maximum likelihood principle, it follows

$$J(\boldsymbol{\theta}|\boldsymbol{Y},\boldsymbol{Z}) = -\log\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{Y},\boldsymbol{Z}) = -\sum_{i=1}^{m}\sum_{c=1}^{K} y_{ic}\log\pi_{ic} \tag{2.49}$$

with the feature tensor $\boldsymbol{Z} = (\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)})$ and the target tensor $\boldsymbol{Y} = (\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(m)})$ This loss function is also called *categorical cross-entropy loss*. Some authors refer to it as simple *cross-entropy* or *cross-entropy loss* although this is a misnomer since a loss composed of negative log likelihoods is always a *cross-entropy* between the empirical target distribution of the training dataset and the model probability distribution (Goodfellow et al. 2016).

The *categorical cross-entropy loss* can also be implemented with weights to compensate for class-imbalances in the data-set. The weights are inserted by multiplying every term of the sum in Eq. (2.49) by a weight w_c dependent on the class c.

Depending on the machine learning library the loss of Eq. (2.49) is sometimes implemented with a normalizing factor of $\frac{1}{m}$ to obtain a mean. In the case that weights were used the normalizing factor is the inverse of all weights employed in the computation of the loss. That is,

$$\frac{1}{\sum_{i=1}^{m}\sum_{c=1}^{K} y_c^{(i)} w_c}. \tag{2.50}$$

However, a multiplication of the loss function by a scalar doesn't affect the optimization problem which remains

$$\boldsymbol{\theta}_{\text{ML}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}|\boldsymbol{Y},\boldsymbol{Z}) \tag{2.51}$$

The distribution commonly used for the last layer in the case of multi-class classification is the softmax function (Goodfellow et al. 2016), which after Eq. (2.26) means

$$\pi_c^{(i)} = \pi(\mathbf{y}^{(i)} = \mathbf{1}_c|\boldsymbol{z}^{(i)};\boldsymbol{\theta}) = \text{softmax}(\boldsymbol{z}^{(i)}(\boldsymbol{\theta}))_c = \frac{\exp z_c^{(i)}}{\sum_{j=1}^{K}\exp z_j^{(i)}} = \hat{y}_c^{(i)}. \tag{2.52}$$

The tensor $\boldsymbol{Z}$ is none other than the a tensor of the feature maps of the second-last layer of my actual model. The optimization algorithm, which updates the parameters – weights and biases – $\boldsymbol{\theta}$ that give rise to the loss, does so by updating them necessarily into the non-linear model that outputs $\boldsymbol{Z}$. The mechanism through which this happens is called **error back-propagation** and is addressed further along this section.

Gradient-Based Optimization

As shown in the previous section there is a natural loss function given the particular properties of the task of multi-class classification Eq. (2.49) and the resulting optimization problem in Eq. (2.51) thanks to the maximum-likelihood principle. The objective is then to find a vector of parameters $\boldsymbol{\theta}$ that minimizes the loss function $J(\boldsymbol{\theta}|\mathbf{Y}, Z)$.

Recalling simple calculus we know that when the gradient of a function $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ vanishes then the function has reached a stationary point, also called a critical point and it is either a maximum, a saddle point or a minimum. What distingushes them is the second derivative. If a maximum is also the highest value the function reaches in its entire domain it is called a global maximum, viceversa for a minimum. In theory one needs to find a solution for the equation

$$0 = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}|\mathbf{Y}, \boldsymbol{Z}) \tag{2.53}$$

however this is analytically intractable (Bishop 2006). The main solution to find a point that satisfies Eq. (2.53) is through a gradient-based optimization algorithm (Goodfellow et al. 2016; Bishop 2006). The intuition behind this simple algorithm is that the gradient of a function points towards the direction where the function increases the fastest. By using this information one can step-wise evaluate the gradient of the loss for a certain set of parameters $\boldsymbol{\theta}$ and then update the vector according to the newly computed gradient and opposite to it. The τ-th step of the algorithm looks like

$$\boldsymbol{\theta}^{(\tau)} = \boldsymbol{\theta}^{(\tau-1)} - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}|\mathbf{Y}, \boldsymbol{Z})\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(\tau)}}. \tag{2.54}$$

The value γ is a hyperparameter – also called learning rate – and is normally assumed to be small (Goodfellow et al. 2016; Bishop 2006).

For the computation of the actual gradient over the dataset one would, in theory, need to evaluate the model for every example of the dataset. This is very expensive especially in terms of memory usage and it is commonly preferred to use only a subset of examples for every step. Further, it is normal to expect that the training set has some redundancy in it in the form of examples that contain very similar information and therefore contribute in the same way to the gradient. Finally, there are less then linear returns that come by just adding examples for the estimate of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ so that the benefit of using 100 times more memory and computation yields an accuracy that is less than 100 times higher (Goodfellow et al. 2016).

Optimization algorithms can either use the entire training set, a single example or a number of examples that are less than the whole training set. These three cases are called, in order, **batch** or **deterministic**, **stochastic** or **online**, **minibatch** or **(minibatch) stochastic**. The number of examples for every step is also called the *batch size* and is another hyperparameter. It's clear how the **deterministic** approach and the **online** approach are just special cases of the more general **minibatch**: in the first one the *batch size* is simply equal to the length of the dataset whilst in the second case the same hyperparameter is 1 (Goodfellow et al. 2016). When trying to decide which value to choose for the *batch size* the key factors are memory usage, efficient employment of the computational architecture (Goodfellow et al. 2016) and actual statistical considerations such as regularization effects of a small batch size (Wilson and Martinez 2003).

In the following section I give a simple overview behind the princples of forward propagation and error backpropagation used to compute the gradients and update the parameters.

Forward propagation and error backpropagation

Once an architecture has been chosen and an appropriate loss is derived – thanks to the maximmum likelihood principle – the necessary training step for updating the parameters of the model follows a simple cicle of events. First, the loss $J(\boldsymbol{\theta})$ is computed through forward propagation. Second, one computes the gradient of the loss with respect to all training parameters with the backpropagation procedure. Finally, an update of the weights and biases takes place, i.e. the trainable parameters $\boldsymbol{\theta}$ following either directly a very simple solution like Eq. (2.54) or a more complex implementation that builds on it (Goodfellow et al. 2016).

1. **Forward propagation** is another way that is used to call the forward flow of information in a network with a feedforward architecture. Given an input $\boldsymbol{x}$ the model computes an output value $\hat{\boldsymbol{y}}$ which, in the case of supervised learning – in the specific case of this project, multi-class classification – the output value is compared to a target $\boldsymbol{y}$. This process can go one step further and $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$ can be used to calculate a loss $J(\boldsymbol{\theta})$ (Goodfellow et al. 2016).

2. In the second step of the process one computes the value of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Even if the architecture is fairly complex one can employ a simple algorithm, called **error backpropagation** – sometimes just **backpropagation** – to compute

all elements of the gradient of the cost function with respect to every single component of the parameter vector $\boldsymbol{\theta}$. The intuition behind this algorithm is to use the chain rule:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.55}$$

with $y = g(x)$, $z = f(g(x)) = f(y)$ and $f, g : \mathbb{R} \to \mathbb{R}$. Eq. (2.55) can be expanded to the multidimensional case and it is used to compute the necessary gradients by decomposing them in smaller simple steps (Bishop 2006).

3. The last step is the **parameter update** step where the values obtained in step 2 are used to compute the new values of the parameters. The simplest way to do the update is to use Eq. (2.54). However, this is not the most efficient algorithm (Bishop 2006) and many different algorithms are used in the machine learning literature that build upon the vanilla one. The choice of the update rule in gradient-descent optimizations is a matter of research; the most notable are the stochastig gradient descent with momentum (Qian 1999), Adagrad (Duchi et al. 2011), Adadelta (Zeiler 2012) and (Kingma and Ba 2014). In this thesis I use Adadelta (Zeiler 2012) to keep in line with Agrawal et al. (2019).

The following section is a brief explanation of the optimization algorithm Adadelta (Zeiler 2012) used in this project.

Adadelta: An Adaptive Learning Rate Method

As stated before, the objective of many machine learning algorithms is to minimize a certain function, which in my case is the loss $J(\boldsymbol{\theta})$. This is done by computing an update to the parameters $\Delta\boldsymbol{\theta}$ that in my case – since I decided to use a gradient-descent based optimization procedure – has its simplest version corresponding to the negative gradient multiplied by a learning rate γ (cf. Eq. (2.54)).

$$\boldsymbol{\theta}^{(\tau)} = \boldsymbol{\theta}^{(\tau-1)} + \Delta\boldsymbol{\theta}^{(\tau-1)} \tag{2.56}$$

Although with the use of the backpropagation algorithm one can easily compute the gradients programmatically, the learning rate hyperparameter remains to be chosen and set manually. This choice is not casual and is a delicate balance between a risk of divergence with a too high learning rate and risk of slow training for a too low value of the hyperparameter; this is one of the issues that the Adadelta optimization algorithm addresses by introducing a per-dimension dynamic-learning rate using only first-order information[14] (Zeiler 2012).

[14]Without computing second derivatives.

A very important and effective modification of the vanilla consists of computing second-order derivatives and modifying the update accordingly (Goodfellow et al. 2016) such as in the case of the so-called Newton's method (Nocedal and Wright 2006):

$$\Delta\boldsymbol{\theta}^{(\tau)} = {\boldsymbol{H}^{(\tau)}}^{-1}\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}^{(\tau)}) = {\boldsymbol{H}^{(\tau)}}^{-1}\boldsymbol{g}^{(\tau)} \tag{2.57}$$

where $\boldsymbol{H}$ is the Hessian matrix of the second-order derivatives of J. I also introduce a simplified notation for the gradient and use $\boldsymbol{g}$. However, employing second-order methods has impossible computational costs for large models (Goodfellow et al. 2016). Hence many optimization algorithms either improve the use of first-order derivatives or try to approximate second-order information.

In the hope of estimating a good learinig rate, many approaches suggest to employ learning rate schedulers that modify the learning rate when the training slows down in order to avoid an oscillation around the minimum (Goodfellow et al. 2016; Nocedal and Wright 2006). However, these methods add hyperparameters to tune the decay rate of the learning rate and do not act on a per-dimension bases because the employ a single global learning-rate.

The simplest and best known method for improving training on a per-dimension basis is the *momentum method* (Qian 1999) by accelerating progress where the gradient points consistently in the same direction and slowing it down where the sign changes often. The update step is composed of an exponential decay that stores past information. Formally,

$$\Delta\boldsymbol{\theta}^{(\tau)} = \rho\Delta\boldsymbol{\theta}^{(\tau-1)} - \gamma\boldsymbol{g}^{(\tau)} \tag{2.58}$$

where ρ is constant that controls the decay rate of the momentum term.

An additional important first-order method called Adagrad (Duchi et al. 2011) tries to approximate some properties of second order methods by only using second order information (Zeiler 2012) by dividing the global learning rate γ by the 2-norm[15] of the previous gradients for each dimension indipendently. That is,

$$\Delta\theta_i^{(\tau)} = -\frac{\gamma}{\sqrt{\sum_{t=1}^{\tau} {g_i^{(\tau)}}^2}} g_i^{(\tau)}. \tag{2.59}$$

This update rule implies that large (small) gradients are associated with a small (large) learning rate causing the progress along each dimension to even out over time as with second-order approaches (Zeiler 2012). However, this method has two main drawbacks: (1) if initial gradients are large then the learning rate remains low for the rest of the training and (2) the accumulation of squared gradients causes

[15]The p-norm or L_p-norm is defined as $(\sum_{i=1}^{n} x_i^p)^{\frac{1}{p}}$ for a vector of length n. In this case I extend the definition to a sequence of vector components and calculate the 2-norm on the time dimension.

the denominator to continuously increase, further slowing down training. Adadelta tries to overcome these flaws (Zeiler 2012).

The Adadelta method is based on two intuitions by the authors, I will briefly go over both of them:

First intuition: The sum of square gradients should be limited to a window instead of being formed out of the entire sequence of past gradients so that the denominator cannot become infintely large. However storing gradients that are inside the window is expensive therefore this window is implemented as an exponentially decaying average of squared gradients (Zeiler 2012). I have,

$$E[\boldsymbol{g}^2]^{(\tau)} = \rho E[\boldsymbol{g}^2]^{(\tau-1)} + (1-\rho){\boldsymbol{g}^{(\tau)}}^2 \tag{2.60}$$

with $E[\boldsymbol{g}^2]^{(\tau)}$ being the decaying average at step τ mentioned above and ρ the decay rate similar to Eq. (2.58). By requiring the square root of $E[\boldsymbol{g}^2]^{(\tau)}$ in the denominator to keep in line with Eq. (2.59) the result is very similar to a Root Mean Square[16] The new update rule reads,

$$\Delta\theta_i^{(\tau)} = -\frac{\gamma}{\sqrt{E[g_i^2]^{(\tau)} + \epsilon}} g_i^{(\tau)} = -\frac{\gamma}{RMS[g_i]^{(\tau)}} g_i^{(\tau)}. \tag{2.61}$$

for every element i of the various parameter and gradient vectors. ϵ is a small constant – typically in the order of 10^{-6} – introduced to improve numerical stability (Becker and Lecun 1989).

Second intuition: if the components in the update rules were to have units, the units of the various components of vanilla SGD and momentum would both be mismatched with respect to the parameters. The unit of the update step are expected to be of the same type as the parameter. However, one can see in Eq. (2.58), Eq. (2.59) and Eq. (2.61) that the update is either unitless or has units of the inverse of the parameter (Zeiler 2012). This is not the case for second order methods such as Newton's method as Eq. (2.57). By rearrangin the terms one can find the units of the term that would approximate the Hessian of Eq. (2.57). This means in terms of units,

$$\Delta\theta \propto \frac{\frac{\partial J}{\partial\theta}}{\frac{\partial^2 J}{\partial\theta^2}} \Rightarrow \frac{1}{\frac{\partial^2 J}{\partial\theta^2}} \propto \frac{\Delta\theta}{\frac{\partial J}{\partial\theta}}. \tag{2.62}$$

By comparing Eq. (2.61) with Eq. (2.62) it is clear that the part missing is the numerator (Zeiler 2012). Zeiler (2012) suggest adding an RMS similar to that of

[16]The Root Mean Square (RMS) is defined as $RMS[x] = \sqrt{\frac{1}{n}\sum_{i=1}^{n} {x_i}^2}$ for a set of n values $\{x_1, ..., x_n\}$. I abuse notation and extend the definition to include the root of the decaying average of Eq. (2.60).

Eq. (2.61) albeit delayed by one step. This provides the update rule for the Adadelta method:

$$\Delta\theta_i^{(\tau)} = -\frac{\sqrt{E[\Delta\theta_i^2]^{(\tau-1)} + \epsilon}}{\sqrt{E[g_i^2]^{(\tau)} + \epsilon}} g_i^{(\tau)} = -\frac{RMS[\Delta\theta_i]^{(\tau-1)}}{RMS[g_i]^{(\tau)}} g_i^{(\tau)}. \tag{2.63}$$

with parameters ρ and ϵ being shared between numerator and denominator. Introducing ϵ in the numerator ensures that the first iterations actually starts given that $\Delta\boldsymbol{\theta}^{(\tau=0)} = 0$ and that updates always happen even if the running average becomes small (Zeiler 2012).

With very little additional computation compared to Adagrad, Adadelta is able to remove the need for setting a manual learning rate and has the benefit of implementing dynamic larning rates on a per-dimension basis. Further, it retains the acceleration feature of momentum SGD by accumulating previous steps in the numerator. From Adagrad it inherits the ability to even out progress per-dimension with the advantage of not having a denominator that accumulates to infinty. Lastly, the algorithm is an approximation of second-order information with only first-order information (Zeiler 2012). For a precise account on how the Hessian was approximated see Becker and Lecun (1989).

Some implementations add additional terms to the original update rule of Adadelta such as reintroducing a learning rate or inserting parameter decay. For this project I employ the original adadelta optimization algorithm with parameter listed in Table 2.4.

ADADELTA Optimizer Hyperparameters		
Hyperparameter	Symbol	Value
Decay rate	ρ	0.9
Numerical stability constant	ϵ	10^{-6}

Table 2.4: Hyperparameters for the Adadelta optimization algorithm used in this thesis.

2.2.12 Regularization

A very important problem in the field of machine learning is making sure that models *generalize* well. If one wants a model to have predictive power this should have low error scores not only on the training data but also on novel data. This central issue in machine learning and more so in the case of neural networks is precisely due to the astounding number of parameters that a model has – as an example, the recent GPT-3 model by OpenAI has an impressive 175 billion parameters (Brown et al. 2020). A model that is unable to generalize is said to be *overfitting* (Bishop 2006). With a limited number of training data and a disproportionate number of parameters the model finds itself learning the examples fed to it – and their inherent noise – by "heart" without learning any actual patterns that are really present and that are shared between the training and the test data (Goodfellow et al. 2016). The strategies that are designed to reduce the test error are called, as a whole, **regularization** (Goodfellow et al. 2016; Bishop 2006). Finding good regularization techniques is a very important topic of research in the deep learning community. Following are the description of the regularization methods used for this project.

Batch Normalization

As the network parameters change during training, a layer's input distribution also changes. Changing parameters creates a change in the distribution in the input, resulting in unwanted behavior like vanishing or exploding gradients; this phenomenon is called *Internal Covariate Shift* (Ioffe and Szegedy 2015). Batch normalization tries to address this problem by normalizing a feature map to a set mean and standard deviation (normally a mean of 0 and a standard deviation of 1), both learnable parameters. Batch normalization is computed on fore every feature map indipendently. Formally for a single input it reads,

$$BN_{\gamma,\beta}(\boldsymbol{x})_k = \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\mathrm{Var}[x_k] + \epsilon}} \cdot \gamma_k + \beta_k \tag{2.64}$$

with k being the activation number in a layer. $\boldsymbol{x} = (x_1, ..., x_K)$ is the input with K elements. ϵ is a small constant for numerical staibility. γ_k and β_k are two learnable parameters (Ioffe and Szegedy 2015).

If the entire training set is used for every training step, this requires the mean and the variance to be computed on the entire set. For stochastic gradient descent this is impractical and the mean and variance on a single batch are used instead (Ioffe and Szegedy 2015). For a batch $\mathcal{B}$ of m examples and omitting the k element number (given that batch normalization is computed indipendently for every activation) the mean $\mu_{\mathcal{B}}$ and the variance $\sigma^2_{\mathcal{B}}$ of the variance are computed as follows, according to Ioffe and Szegedy (2015),

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.65}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \tag{2.66}$$

$$y_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \cdot \gamma + \beta \equiv BN_{\gamma,\beta}(x_i) \tag{2.67}$$

with y_i being the output after the Batch Normalization. The layer can relearn the original distribution by undoing the normalization if it's the convenient thing to do.

In order to keep track of past information a running average for μ and σ is computed for every mini-batch with the following updating rule (Ioffe and Szegedy 2015) before the normalization step:

$$\mu_b \leftarrow (1-\rho) \cdot \mu_{b-1} + \rho \mu_b \tag{2.68}$$

$$\sigma_b^2 \leftarrow (1-\rho) \cdot \sigma_{b-1}^2 + \rho \sigma_b^2 \tag{2.69}$$

where b is the batch number. ρ is the momentum for the update of the running mean.

As stated before, Batch Normalization is employed before a non-linear activation function to stabilize the input. Note that in the case of an affine transformation of the type

$$\boldsymbol{z} = g(\boldsymbol{W}\boldsymbol{u} + \boldsymbol{b}) \tag{2.70}$$

with an element-wise non-linearity g and learned parameters $\boldsymbol{W}$ and $\boldsymbol{b}$, if one introduces a batch normalization the bias term $\boldsymbol{b}$ is irrelevant. Mathematically,

$$\boldsymbol{z} = g(BN(\boldsymbol{W}\boldsymbol{u} + \boldsymbol{b})) = g(BN(\boldsymbol{W}\boldsymbol{u})). \tag{2.71}$$

This implies that a bias term is not needed for the layer preceding a batch normalization (Ioffe and Szegedy 2015).

Further, to adopt Batch Normalization with convolutional layers, in order to respect the parameter-sharing property, the normalization is applied jointly on elements of the same feature map. This means that the learne pair of parameters γ and β is not uinique for every activation but it is shared by an entire feature map (Ioffe and Szegedy 2015).

Lastly, Ioffe and Szegedy (2015) notes how Batch normalization improves network training time by reducing the dependence of gradients on the scale or initial values of the parameters and allowing for a larger learning rate. Batch normalization has a regularizing effect and reduces the need for other types of regularization such as dropout layers.

Early stopping

If models are sufficiently large to be prone to overfitting, a common observation is that, although the training error continues to decrease, the validation set inverts this direction at certain point in time (Goodfellow et al. 2016). To avoid this inconvenience a common and easy to implement solution is to stop the training procedure when no decrease in the validation error is detected. This method is also called **early stopping** (Prechelt 1998). Practically it is implemented by checking a performance metric of the model over the validation set at every epoch and the training is interrupted if a minimum improvement (itself a parameter) of the metric is not detected within a certain number (called patience) of past epochs.

Loshchilov and Hutter (2017) investigated the effect of having weight decay explicitly set in the update rule in the case of adaptive learning rate algorithms effectively decoupling the norm-penalty from the gradient based step. The *decoupled weight-decay*, as it is normally called, has been implemented in multiple machine learning libraries for the adaptive learning rate optimization algorithms.

2.2.13 Weights Initialization

Deep learning algorithms, being iterative, need a specification of the initial values of the parameters from which to start the training process. The model is sufficiently complex that the majority of algorithms are very sensible to the choice of initialization. Vanishing and exploding gradients are the central effect of bad initialization (Glorot and Bengio 2010). Some initialization points are extremely unstable and don't lead to a convergence of the algorithm (Goodfellow et al. 2016).

It is well accepted that rectifier networks perform better than the counterparts using smoother activation functions such as hyperbolic tangent and sigmoid (Glorot et al. 2011; Glorot and Bengio 2010). Nonetheless their performance can still be affected in a negative way by a wrongly chosen initialization scheme.

LeCun et al. (1998) suggest an initialization scheme based on a uniform distribution for sigmoid activation functions such that they find themselves in their linear range. Glorot and Bengio (2010) proposed the adoption of an initialization method based on a scaled uniform distribution similar to LeCun et al. (1998) in order to maintain constant variance of the gradients during backpropagation. This initialization scheme is called *Xavier* or *Glorot* and it is used for sigmoidal activations. Always on the principles identified by LeCun et al. (1998) and improved upon by Glorot and Bengio (2010), He et al. (2015) propose corrections for the peculiarities of ReLU activation functions and compute different boundaries for the uniform initialization distributions of the weights. This method is called *Kaiming* or *He* initialization.

2.2.14 Hyperparameter Search

As stated previously, Deep learning algorithms have normally a set of parameters, called *hyperparameters*, that influence different characteristics of the algorithm and the way it behaves. Time, memory and quality of the model's ability to make accurate predictions from novel data are the most important aspects influenced by hyperparameters (Goodfellow et al. 2016).

I will briefly discuss the some strategies that are used to choose the correct set of hyperparameters:

1. **Manual hyperparameter tuning** is easiest in terms of computational cost but requires a very good understanding of how the various hyperparameters affect the behaviour of the algorithm and the results. During manual tuning the user has to balance the capacity of the model to represent the problem, the learning algorithm's ability to reach a minimum of the cost function and the degree to which regularization is achieved. One needs also to pay constant attention to the training and validation error to spot either over or underfitting. Further, once low training error is achieved, it is generally very effective to reach a good performance on the test set by simply increasing the training set size and matching it with a proportionate model capacity (Goodfellow et al. 2016).

2. **Grid search** is a valid choice with a limited number of hyperparameters and a limited search space. For every hyperparameter a small number of values are selected and an algorithm trains the model for every combination of hyperparameter values. A clear disadvantage of this approach is the exponential growth of the computational cost with $O(n^m)$ for m hyperparameters taking n values (Goodfellow et al. 2016).

3. **Random search** is an alternative to grid search when there are more hyperparameters and more values involved. Hyperparameters that have a numeric real value are sampled from a uniform distribution or from a log-uniform distribution. Whilst hyperparameters that take on discrete values are sampled according to a Bernoulli (in case of a binary hyperparameter) or multinoulli distribution. This approach was found to yield good hyperparameter combinations much faster than grid search (Bergstra and Bengio 2012; Goodfellow et al. 2016).

For an account on other hyperparameter tuning strategies see Bergstra et al. (2011)

2.3 Metrics

Considering the formulation of the problem as stated in Section 2.2.2 I utilize classification metrics normally employed in the case of nonprobabilistic forecasts for discrete predictands following Wilks (2011). I briefly discuss the binary classification case and then extend the notions to the multi-class problem.

In classification tasks a prediction can lie within one of four groups.

- **True Positives** written also as TP is the subset of elements correctly classified as belonging to a certain class.

- **True Negatives** also referred to as TN are all the elements correctly classified as not belonging to a certain class.
- **False Positives** or FP are the group of elements that were wrongly classified as belonging to a certain class.
- **False Negatives** or FN are all the elements wrongly classified as not belonging to a certain class.

these groups can be organized in a 2x2 table that is commonly called a **confusion matrix** or with the more general term **contingency table**. The confusion matrix can be extended to multiple classes which leads to having a further subdivision of all the classification groups except for the TP, e.g. for a 3-class problem elements can be mis-classified as belonging to two other classes leading to a break down of the FN.

With different combinations of the prediction/ground-truth pairs as specified in the previous paragraph, different metrics can be obtained.

- **Accuracy**, also known as *proportion correct*, is defined as

$$\mathrm{A} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{2.72}$$

  and is the ratio of correct predictions over the entirety of predictions made. This measure is commonly monitored during training as a way to track progress. The canonical extension to the multi-class case has the disadvantage of not taking into account class imbalances.

- **Precision** is defined as

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{2.73}$$

  and is also known as *positive predictive value* or *PPV*. It's the ratio of the examples predicted as belonging to the relevant class that are also relevant in reality. Put differently, it's the probability that, given a classification to a class c, the element really belongs to that class.

- **Recall** is defined as

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2.74}$$

  and it's the ability of the classifier to correctly identify all members that really belong to a certain class c. It's very commonly known as *sensitivity* or *hit rate*. A good classifier should maximise both Precision and Recall, however in reality there is a trade-off between the two. A measure that considers both Precision and Recall is the F-1 measure.

- **F-measure** or F_1-*score* is the harmonic mean of Precision and Recall

$$\mathrm{F_1} = 2 \cdot \frac{\mathrm{P} \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}} \tag{2.75}$$

  and is a specific case of the more general F_β-*score*[17].

- **Critical Success Index** (*CSI*) or *Threat Score* (*TS*) is defined as

$$\mathrm{TS} = \mathrm{CSI} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \tag{2.76}$$

  is a useful measure when the frequency of the Negatives is much higher than the frequency of the Positives. It is similar to accuracy but it does not consider the TN.

Another common way to quantify the goodness of forecasts is to use so-called **Skill Scores**. This measure is interpreted as the improvement over a reference forecast. With a generic measure of Accuracy (different from the accuracy previously defined) the skill score is defined as

$$\mathrm{SS_{ref}} = \frac{\mathrm{A} - \mathrm{A_{ref}}}{\mathrm{A_{perf}} - \mathrm{A_{ref}}} \tag{2.77}$$

where A is the measure of accuracy and the subscripts refer to the reference forecast (ref) and the perfect forecast (perf).

- **Heidke Skill Score** is a common skill score popularly known thanks to Heidke (1926) and uses as the accuracy measure the one defined in Eq. (2.72). The reference accuracy measure is the accuracy by random forecasts independent from observations. It uses marginal probabilities inferred from the relevant contingency table. The probability of a correct attribution to a class is the multiplication of the probability that an element belongs to that class times the probability that it is observed as belonging to that class. Similarly one can derive the probability of correctly not assigning a member to a class. One then obtains

$$\mathrm{HSS} = \frac{2(\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN})}{(\mathrm{TP} + \mathrm{FN})(\mathrm{FN} + \mathrm{TN}) + (\mathrm{TP} + \mathrm{FP})(\mathrm{FP} + \mathrm{TN})} \tag{2.78}$$

  that yields 0 for no improvement over the reference forcast and 1 for a perfect forecast.

For the multiclass classification problems other than a simple comparison of the metrics computed for the individual classes, one can aggregate them to have a

[17]The F_β-*score* is defined as $\mathrm{F}_\beta = (1 - \beta^2)\frac{\mathrm{P} \cdot \mathrm{R}}{\beta^2 \cdot \mathrm{P} + \mathrm{R}}$

single value of the goodness of the model or forecast. Some metrics have their own extension such as the HSS

$$\text{HSS}_{multi} = \frac{\sum_{c=1}^{K} p(f_c, o_c) - \sum_{c=1}^{K} p(f_c)p(o_c)}{1 - \sum_{c=1}^{K} p(f_c)p(f_c)} \tag{2.79}$$

where $p(f_c, o_c)$ denotes the probability that target and forecast correspond for the class c (the accuracy of Eq. (2.72)), $p(f_c)$ is the probability that class c is forecast and $p(o_c)$ is the probability that class c is observed. These probabilities for the K classes are obtained with a frequentist approach from the K×K confusion matrix.

Another more direct way to aggregate a metric M for the various K classes is the **macro average**

$$\overline{\text{M}} = \frac{1}{\text{K}} \sum_{c=1}^{K} \text{M}_c \tag{2.80}$$

which works well when, as is the case in some multiclass classification problems, one of the classes is underrepresented in the dataset but is of high importance.

The **micro average** in turn computes a metric M globally. For multi-class classification problems, the micro average of F-score, precision, recall is equal to the accuracy. This way of averaging strongly underestimates minority classes if they appear in very low frequency.

The **weighted average** weights the metrics according to their frequency in the dataset and compensates for class imbalance. Hence, it is preferable not to use this averaging technique if the minority classes are important.

In this project I use Precison, Recall, F-Score, CSI and HSS both for every class independently and as a macro average to evaluate the performance of the model.

Chapter 3

Results

In this section I will report the statistics of the RADNET dataset leading to the choice of the weights for the cross-entropy loss. The distribution of rainfall rates is assessed. Finally, I give the final results for the relevant classification task.

3.1 Dataset Statistics

Rainfall patterns vary greatly from location to location which in turn affects the distribution and frequency of the precipitation rainfall rates picked up through radar. The RADOLAN dataset is no exception. Given that this project has as goal to replicate to some extent the model by Agrawal et al. (2019) I verify that induced choices are at least to a first glance acceptable. When it comes to dividing rainfall into classes for the classification problem Agrawal et al. (2019) decides to adopt three different thresholds for this subdivision. These are 0.1 mm/h, 1 mm/h and 2.5 mm/h. However, differences in the structure of the problem of this project and the one by Agrawal et al. (2019) exist. (1) The categorization problem by Agrawal et al. (2019) outputs images that are compared with radar derived precipitation estimates at 1 h. These estimates are outputted every 10min. Conversely, for this project both the inputs and outputs are given every at an hourly rate from each other. (2) Agrawal et al. (2019) structures the problem as multiple binary classification tasks whereas this project is structured as one single multi-class classification task. (3) Oversampling frames with precipitation is the method used by Agrawal et al. (2019) to compensate for the fact that the majority of pixels is rainless. In this project I simply ignore the scenes where all of the input frames have no rainy pixel. I also add weights to the simple cross-entropy loss.

The analysis of the dataset then yields class distributions for the different dataset subsets (training, validation, testing) as in Table 3.1 and in Fig. 3.1. A summary of the percentages of scenes used out of all scenes available and their final proportion in the dataset is given in Table 3.2.

Frequency % Classes				
Class Label	Class Interval	Training	Validation	Test
0	$[0, 0.1)$	89.4%	88.02%	87.78%
0.1	$[0.1, 1.0)$	8.34%	9.46%	9.57%
1.0	$[1.0, 2.5)$	1.81%	2.05%	2.2%
2.5	$[2.5, \infty)$	0.42%	0.47%	0.45%

Table 3.1: Frequencies of the different classes in the target channel for every subset of the dataset. The class frequencies for the training subset are used to determine the weights for the weighted cross-entropy loss. A bar chart from the data contained in this table is depicted in Fig. 3.1.

Scenes Selection			
	Training	Validation	Test
Available Scenes	63336	10176	10200
Selected Scenes	58784	7116	7074
Percentage Selected	92.8%	69.9%	69.3%
Proportion in Dataset	80.6%	9.8%	9.7%

Table 3.2: Breakdown of the selection of scenes from all the available scenes. The third row is computed by dividing the selected scenes by the available scenes. The proportion in the total dataset is computed by dividing the selected scenes by the sum of all selected scenes. A scene is discarded if and only if all the input frames are rainless.

Weights for Training

To counteract the class imbalance, using weights in the cross-entropy loss of the model is normally the preferred choice in the deep learning literature. These weights are typically the inverse of the frequency of the classes. That means, following the frequencies as reported in Table 3.1,

Class 0	$\frac{1}{89.4}$
Class 0.1	$\frac{1}{8.34}$
Class 1.0	$\frac{1}{1.81}$
Class 2.5	$\frac{1}{0.42}$

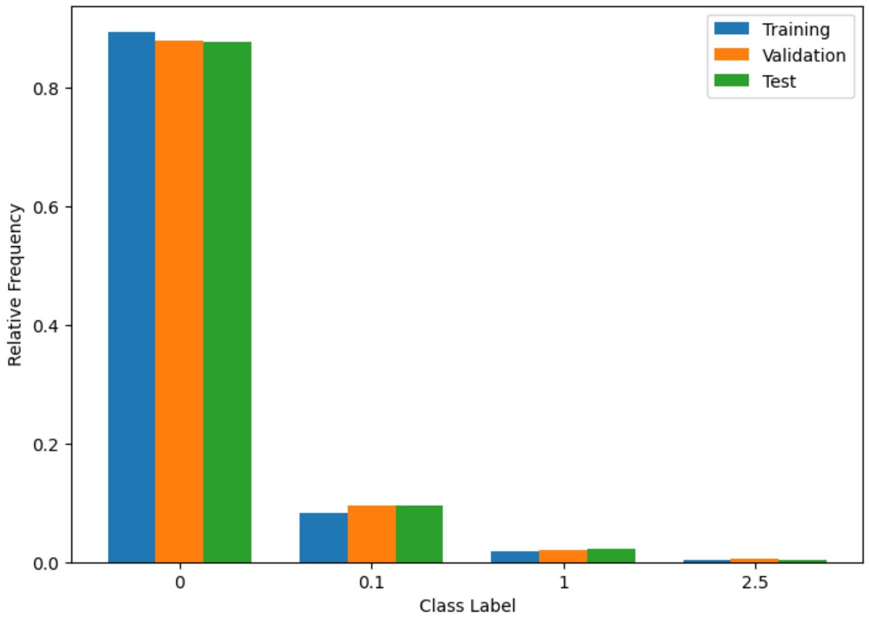

Figure 3.1: Bar chart of the precipitation class (see Table 2.1) frequencies for training, validation and test subset of the dataset. The frequencies sum to 1 for each one of the subsets. The chart is computed using data from Table 3.1.

Other methods of balancing a cross-entropy loss exist, such as the employment of an effective number of samples (Cui et al. 2019) that builds upon the reasoning that as the number of samples increases, the additional benefit of a newly added data point diminishes.

Skewness of inputs

Knowing that high precipitation rates are infrequent and that in general very skewed distributions for the layers of a neural network are not desireable, I assess the frequency and shape of the distribution of the values for the raw precipitation rates of the RADOLAN dataset as those are the ones that are fed as input to the model.

In order to make the distribution less skewed and bring it closer to a normal distribution, a common solution is to transform the data by taking the (natural) logarithm; for instance Ayzel et al. (2020) decide to transform the input to the model by taking the natural logarithm as in

$$x_{\text{transformed}} = \ln(x_{raw} + 0.01) \tag{3.1}$$

where 0.01 is added to the raw input to avoid the logarithm of zero. Fig. 3.2 shows the implication of this transformation for the distribution of the precipitation values.

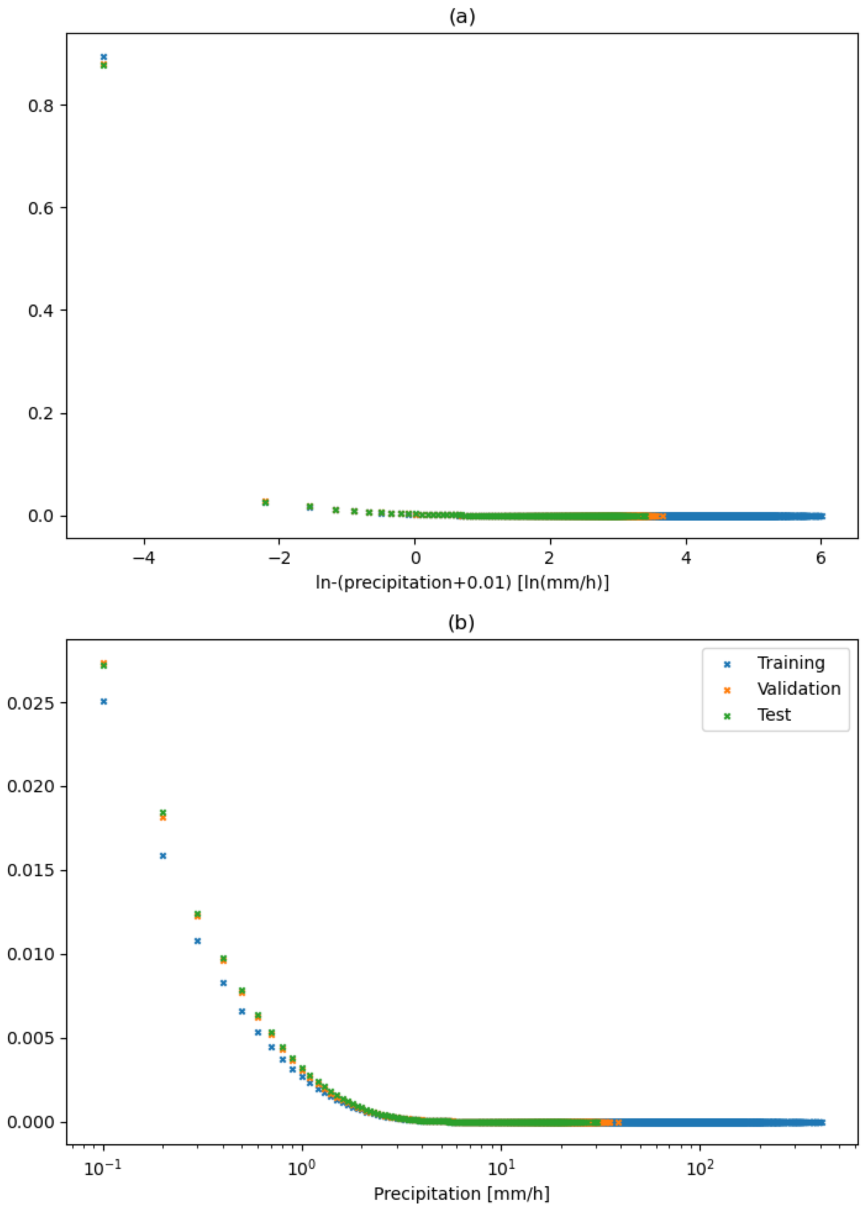

Figure 3.2: Relative frequency of all precipitation values for the pixels of the frames in the RADOLAN dataset. The distributions are computed for the training, validation ad test subsets of the dataset. (a) displays the distribution after the transformation of Eq. (3.1), hence the distribution of the values fed to the model. (b) is the distribution of the values displayed in log-scale without any addition but it does not contain the most frequent value 0 in order to better understand the distribution among the other values.

3.2 Model Performance

Hyperparameter Search

To avoid a computationally expensive procedure, the hyperparameters were divided in two groups. (1) Those manually tuned and (2) those picked by utilizing the available literature.

The manual search was done by trial and error considering all the metrics on the validation set, paying more attention to the HSS and the CSI. Table 3.3 sums up the hyperparameters found through this process.

Hyperparameters	
Name	Value
batch size	32
depth	6
epochs	10

Table 3.3: The hyperparameters found via manual tuning.

Testing

Using the model produced through the hyperparameters of Table 3.3 the metrics are computed on the testing subset. The results are summed up in Table 3.4

Metric	Cl. 0	Cl. 0.1	Cl. 1	Cl. 2.5	Macro average
Precision	**0.988**	0.346	**0.309**	0.218	0.465
	0.943	**0.433**	0.308	**0.227**	**0.478**
Recall	0.861	**0.653**	**0.585**	**0.609**	**0.677**
	0.943	0.433	0.308	0.226	0.478
F1	0.921	**0.452**	**0.404**	**0.321**	**0.525**
	0.943	0.432	0.308	0.227	0.478
CSI	0.853	**0.292**	**0.253**	**0.191**	**0.397**
	0.893	0.276	0.182	0.128	0.370
HSS	0.579	0.438	**0.404**	**0.321**	**0.436**
	0.583	**0.441**	0.314	0.229	0.392

Table 3.4: The results of the model compared to persistence applied to the test subset of the dataset. For every metric the upper row refers to the results of the model whilst the lower row refers to persistence. Higher values are better. The best values for each class and metric are highlighted.

As Table 3.4 shows the performance of the model is generally better than persistence for higher precipitation classes suggesting an influence of the weights for the loss giving higher importance to higher precipitation. The generally high values for precision and recall for the lowest precipitation class can be explained by the overall higher amount of tiles with no precipitation making it an ”easy guess”. The most significant information is to be found in the overall better prediction skill of the model compared to persistence for the remaining classes as the macro-averages testify.

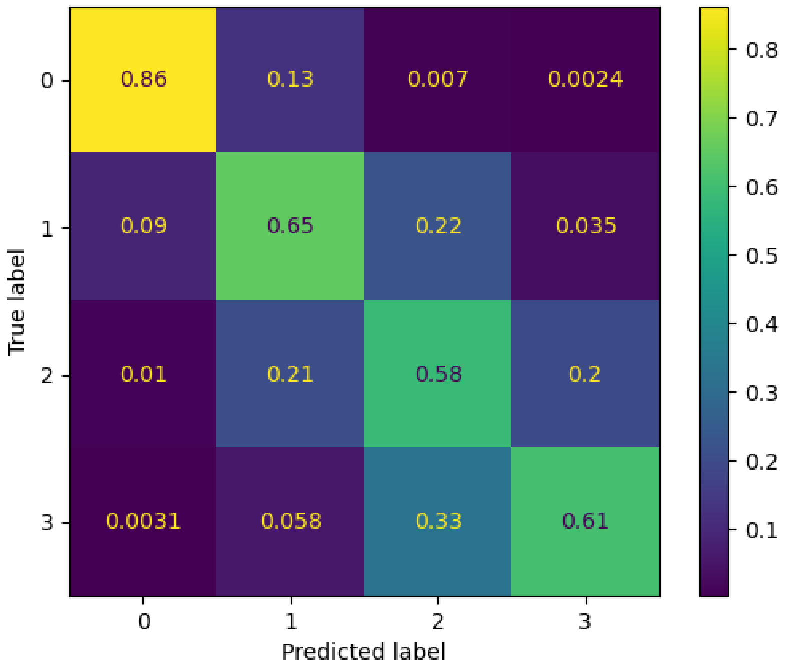

Figure 3.3: The normalized confusion matrix for the predictions made by the model on the test set. Normalization occurs on the rows. The labels 0, 1, 2, 3 stand for the classes 0, 0.1, 1, 2.5 respectively.

Fig. 3.3 is the confusion matrix for the task. As it suggests the majority of wrong classifications happen with neighbouring classes, meaning that the model is able to understand the shape of the precipitation but has difficulties in assessing the intensity correctly.

3.3 Case Studies

To visually assess the goodness of the model I analyse two case studies, the first scene is the one at 2020-02-09T1950Z and the second one is at 2021-03-11T0150Z.

3.3.1 First Case Study: 2020-02-09T1950Z

Synoptic Situation

The synoptic situation near the time of the input and output of the neural network is addressed through the use of the model results produced by NOAA's model Global Forecast System (GFS)[1] and visualized through Ertel2[2]. Geopotential height and equivalent potential temperature at 850hPa are employed. Fig. 3.4 shows the relative figure for the model output at 2020-02-10T0000Z where the presence of a cold front is evident in the northern part of Germany near the area of interest (cf. Fig. 2.6). The orientation of the front is SW-NE. Additionally, given the orientation of the isolines representing the geopotential, the direction of the geostrophic flow is easterly. I also assess the 1h accumulated precipitation associated with this front by observing the output of the ICON (EU) model[3] through Ventusky[4] at 2020-02-10T0300Z. Fig. 3.5 shows the relative image. The precipitation bands near the area of interest in northern Germany (again cf. Fig. 2.6) are oriented in a similar direction to the cold front i.e. SW-NE.

Model Prediction

Following the synoptic situation described in the previous section I analyze the actual output of the model given the input sequence. Fig. 3.6 shows the scene for 2020-02-09T1950Z. This timestamp is the timestamp of the first input image (upper left in the group of 6 images). The other 5 input images are those depicted in the group made of 6 images in Fig. 3.6 and all are of precipitation accumulated in a 1h interval. They are separated by 1h from each other. The model then applies a series of convolutions (cf Section 2.2.6) that abstract the features of the input images in a representational space that the model itself learns during the training process. The spatial resolution is reduced for every layer that the input data goes through in the encoding phase (cf. Section 2.2.10) but is later taken back to the original resolution in the decoding phase. In this last phase the model rebuilds a plausible output image using information from the encoding phase thanks to the skip connections (cf. Section 2.2.8). The timestamp of the output of the model is 6h after the first input image, hence 2020-02-10T0150Z and it is shown in the last row on the left of Fig. 3.6, it is confronted with the ground truth on the same row on the right. This allows for a straightforward comparison and shows if the model has acquired the ability to infer the spatial distribution of the precipitation correctly.

[1]https://www.ncei.noaa.gov/products/weather-climate-models/global-forecast

[2]http://ertel2.uibk.ac.at:8080/

[3]more on the ICON (EU) model to be found at
https://www.dwd.de/EN/ourservices/nwp_forecast_data/nwp_forecast_data.html

[4]https://www.ventusky.com/

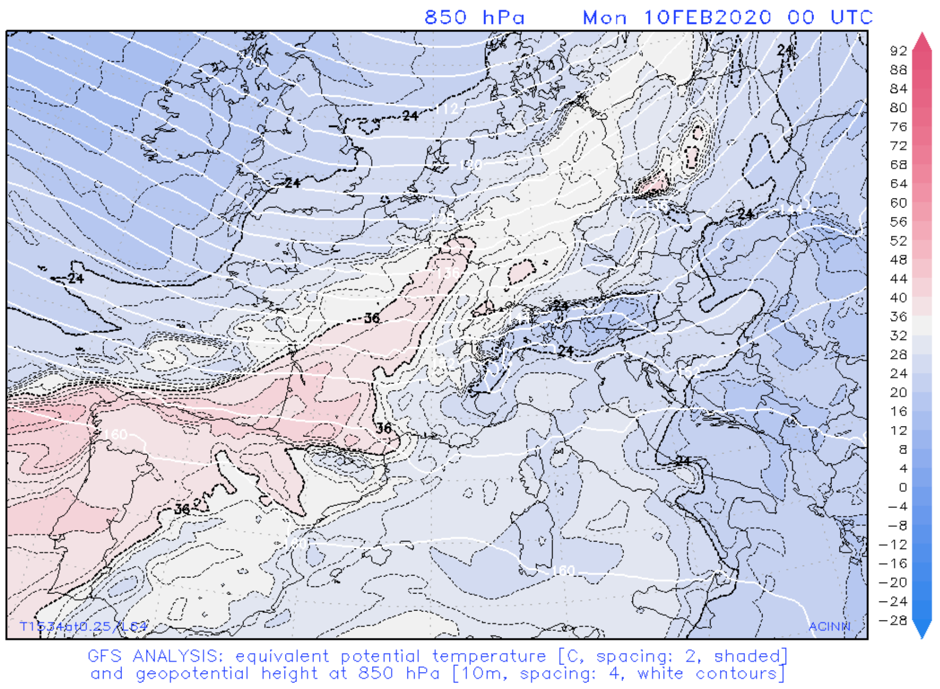
850 hPa Mon 10FEB2020 00 UTC
92
88
84
80
76
72
68
64
60
56
52
48
44
40
36
32
28
24
20
16
12
8
4
0
−4
−8
−12
−16
−20
−24
−28
GFS ANALYSIS: equivalent potential temperature [C, spacing: 2, shaded]
and geopotential height at 850 hPa [10m, spacing: 4, white contours]
ACINN

Figure 3.4: Output of the GFS model for the timestamp 2020-02-10T0000Z showing equivalent potential temperature and geopotential height at 850hPa. The equivalent potential temperature is in C, shaded and with lines at 2°C distance. The geopotential height is shown in white contours and in 10m units. The spacing between the lines is 4. The SW-NE cold front in the area of interest in norther Germany is evident. The flow is assessed through the geopotential height and is easterly. Figure produced with the help of ERTEL2.

As Fig. 3.6 shows the model is indeed able to predict precipitation rainbands going in the SW-NE direction as is seen in the ground truth and in Fig. 3.5. However, it is clear that the model predicts higher precipitation rates than are actually observed. An hypothesis for this behaviour is the influence of the weights employed for the loss that give very high importance to higher precipitation rates.

3.3.2 Second Case Study: 2021-03-11T0150Z

Synoptic Situation

Similarly to the first case study, I assess the synoptic situation at a time approximately near to the input and output of the neural network (the latter is 6h after the input image, hence 2021-03-11T0750Z) using the same tools. Fig. 3.7 shows the output of GFS and shows the presence of a warm front that is starting to enter the area of interest in northern Germany at 2021-03-11T0000Z. Its orientation is S-N. The geostrophic flow associated to the system, being parallel to the isolines of geopoten-

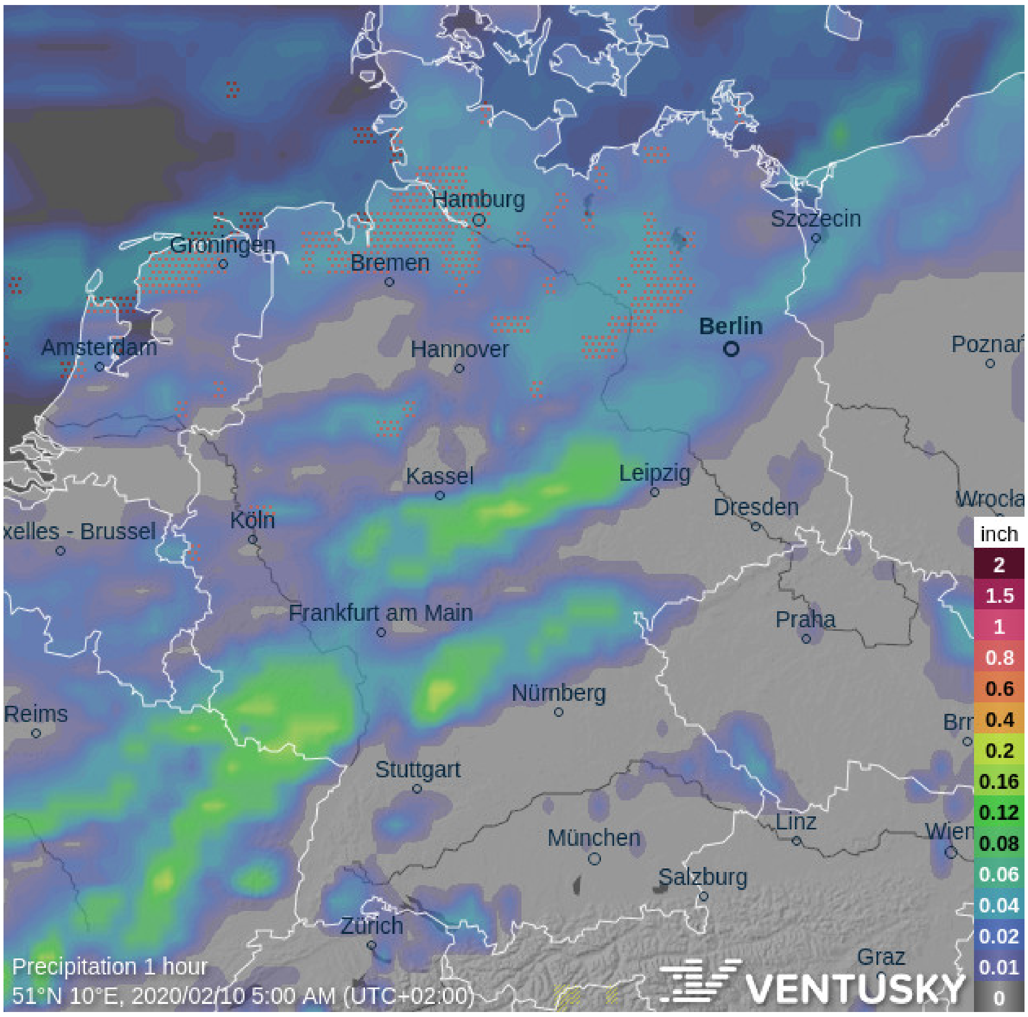

Figure 3.5: 1h precipitation over Germany at 2020-02-10T0300Z according to the ICON (EU) model. The frontal structure after it has passed over the area of interest and its approximate SW-NE orientation can be seen clearly. Figure produced with Ventusky.

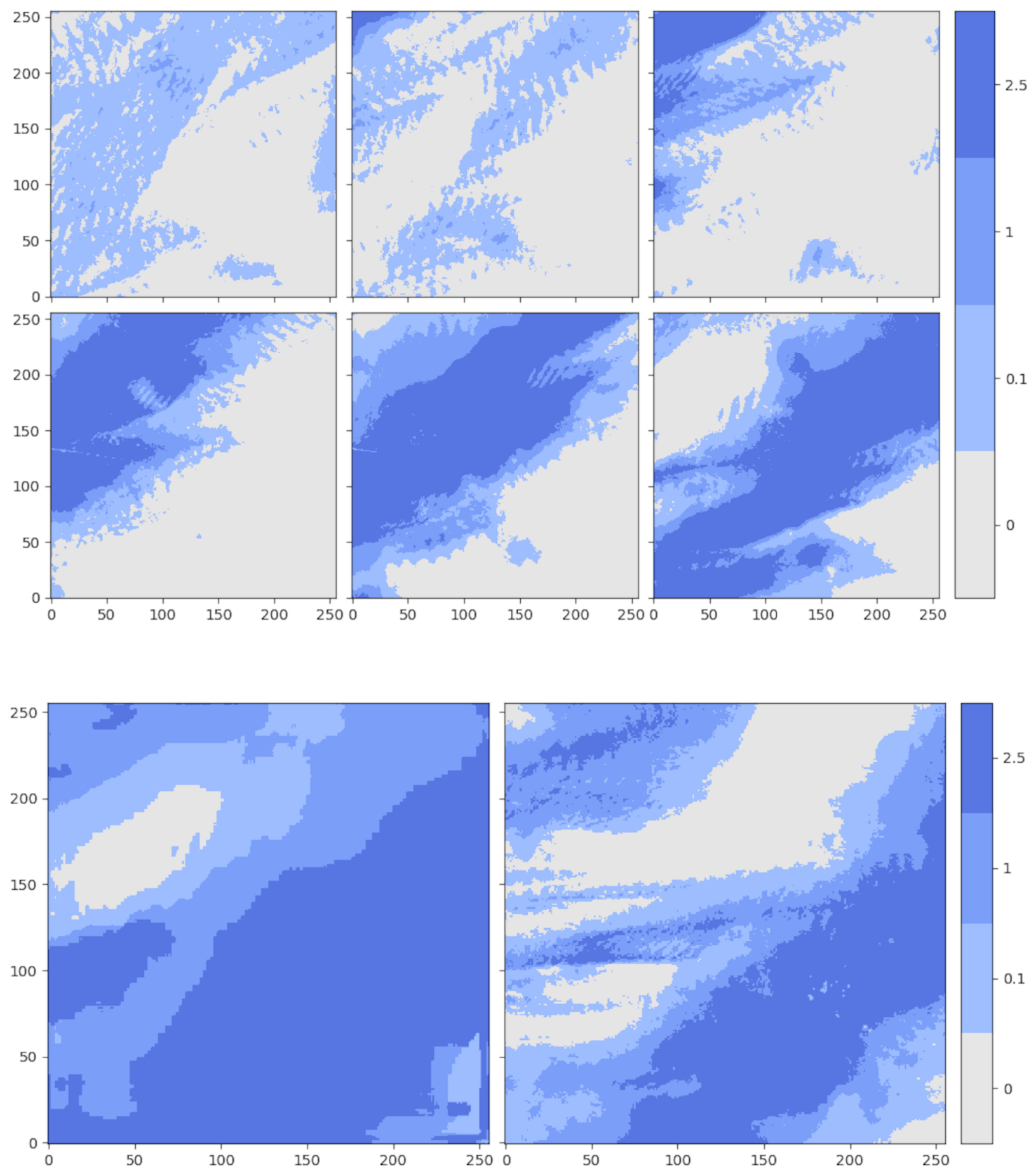

Figure 3.6: Example of input and output and ground truth for 2020-02-09T1950Z. The input part is made of the six images in the top. The prediction is the left image of the lower two and the ground truth is to be found on the right of the prediction. The input is fed to the model as continuous precipitation values but it is depicted in precipitation classes to better compare them with the output. The labels on the right refer to the precipitation classes. The coordinates are in km from the bottom left corner. The order of the inputs is left to right top to bottom. The interval between every one is one hour and all are 1h cumulative precipitation values. The model can be seen to correctly predict the orientation of the precipitation bands. At the same time it predicts higher precipitation values than the ground truth.

tial height, is found to be easterly to south-easterly. The rainbands associated with this front can be seen in the ICON (EU) model output taken at 2021-03-11T0900Z after they have left the area of interest and are in the SW-NE direction.

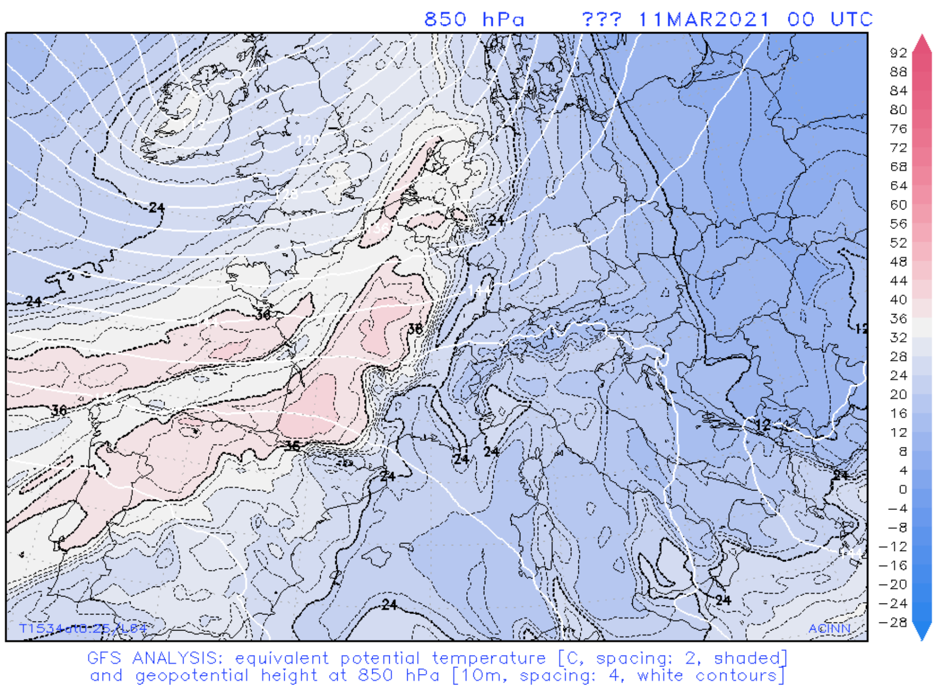
850 hPa ??? 11MAR2021 00 UTC
GFS ANALYSIS: equivalent potential temperature [C, spacing: 2, shaded]
and geopotential height at 850 hPa [10m, spacing: 4, white contours]
ACINN

Figure 3.7: Output of the GFS model for the timestamp 2020-03-11T0000Z showing equivalent potential temperature and geopotential height at 850hPa. The equivalent potential temperature is in C, shaded and with lines at 2°C distance. The geopotential height is shown in white contours and in 10m units. The spacing between the lines is 4. The N-S warm front approaching the area of interest in northern Germany is evident. The geostrophic flow, parallel to the isolines of geopotential height, is easterly to south-easterly. Figure produced with the help of ERTEL2.

Model Prediction

Following the synoptic situation, I assess again the ability of the model to predict high level features such as the direction of the precipitation bands. I find in Fig. 3.9, similarly to the first case, that the model is able to predict the direction of the precipitation bands but outputs higher precipitation rates than the ground truth.

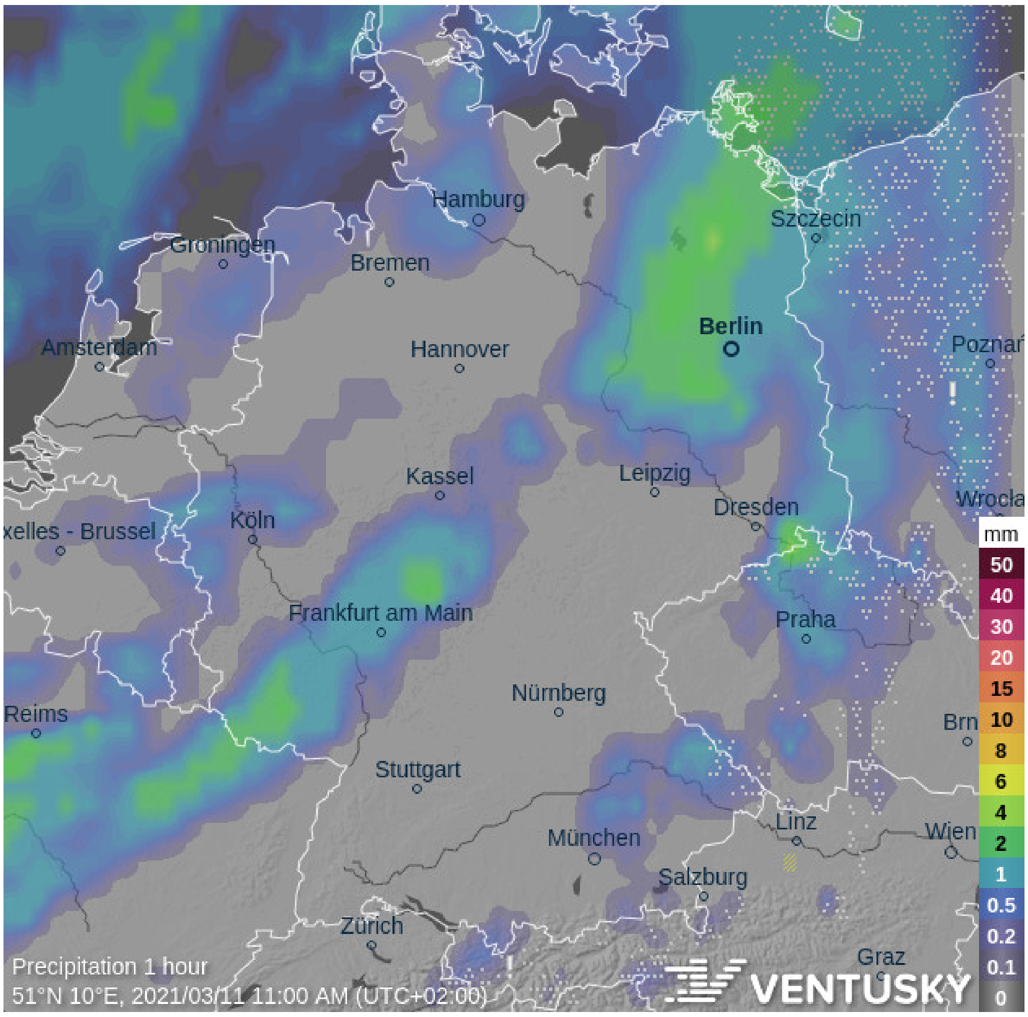

Figure 3.8: 1h precipitation over Germany at 2021-03-11T0900Z according to the ICON (EU) model. The frontal structure after it has passed over the area of interest can be seen over northern Germany. Its orientation is SW-NE. Figure produced with Ventusky.

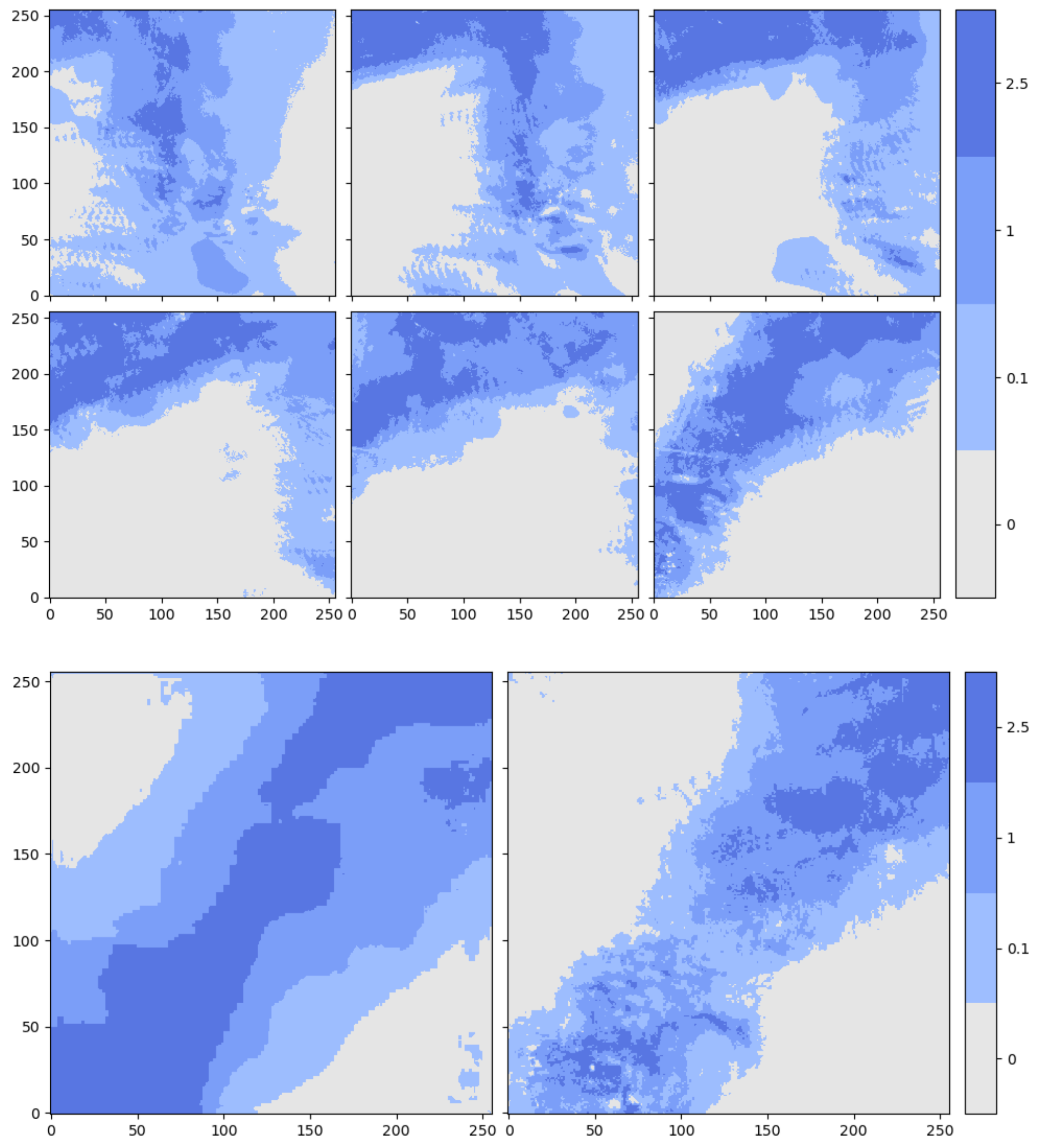

Figure 3.9: Example of input and output and ground truth for 2021-03-11T0150Z. The input part is made of the six images in the top. The prediction is the left image of the lower two and the ground truth is to be found on the right of the prediction. The input is fed to the model as continuous precipitation values but it is depicted in precipitation classes to better compare them with the output. The labels on the right refer to the precipitation classes. The coordinates are in km from the bottom left corner. The order of the inputs is left to right top to bottom. The interval between every one is one hour and all are 1h cumulative precipitation values. The model can be seen to correctly predict the orientation of the precipitation bands. At the same time it predicts higher precipitation values than the ground truth.

Chapter 4

Discussion and Conclusions

In this chapter I will discuss the results by answering the research questions followed by a part on the decision taken for the methodology. Finally, I will compare the results with similar studies and state some possible developments.

4.1 Results

The answers to the research questions of Section 1.3 follow:

1. **Does the employed neural network outperform persistence?** For most of the metrics and classes the neural network outperforms persistence. This is also what the macro-averages indicate. The macro averaged HSS for the neural network is 0.436 compared to 0.392 for persistence. The macro-averaged CSI is 0.397 for the neural network compared to 0.370 for persistence.

2. **How does the model's performance depend on the rainfall intensity?** The performance of the model, in absolute terms decreases with rainfall intensity but gets generally better compared to persistence for increasing rainfall intensity.

3. **The model of Agrawal et al. (2019) after which this project is inspired outperformed persistence for all precipitation classes. Can this result be confirmed?** The model of this thesis was also found to generally outperform persistence albeit not to the same degree as the original paper. However, the ability of the model to abstract features as indicated by Figs. 3.6 and 3.9 suggests that with further hyparameter tuning a similar performance could be achieved.

4.2 Discussion of the Methodology

In this section I will discuss the decisions taken as regards to the methodology.

4.2.1 Choice of the Network Architecture

The network architecture of this project is chosen to resemble the ubiquitous U-Net architecture introduced by Ronneberger et al. (2015) that was found to perform well on segmentation tasks. As for the particular details of the various layers composing the network, I followed Agrawal et al. (2019). I treated the model depth as a hyper-parameter given that Ayzel et al. (2020), trying to solve a similar problem, adopts a depth of 4 (instead of 7 as in Agrawal et al. (2019). An advantage of lower depth is decreased memory footprint and computational requirements. However, not enough parameters can affect the model performance. Unfortunately, Agrawal et al. (2019) are not exhaustive in their description of the architecture. The lacking information was compensated by adopting solutions available in the general deep learning literature. The architecture of short-range skip connections when downsampling blocks are applied and the number of input feature maps to the blocks are different from the number of output feature maps (Drozdzal et al. 2016).

4.2.2 Changes in the RADOLAN Routine

The RADOLAN calibration routine described in Section 2.1.2 is subject to constant changes that are brought into the operational procedure. Weigl (2021a) keeps track of these changes. Some are as as simple as additional ground rain gauges or the dismissal and addition of radar stations. Other changes affect the algorithm of the routine itself, such as better corrections for the shadowing of radar signals by the orography. The DWD has until now reprocessed the RADOLAN database twice, these data are available and could be used to improve the model performance.

4.2.3 Choice of the Training, Validation and Test Sets

It is common practice in the artificial intelligence literature to subdivide the dataset into three subsets, the training set used for optimizing the model, the validation set to compare the different hyperparameter choices and the test set to produce the final data later reported as the actual performance of the model. The proportions 80:10:10 are common choice for deep learning tasks. The choice of chronological order was taken partly also considering the changes in the RADOLAN calibration procedure, hence allowing the final performance measure to be closer to the performance if the model were to be applied to very recent data. A significant drawback of this approach is that it relies on the assumption that the time-series is stationary, i.e. no changes occurred to the underlying processes, in this case the climate. An alternative way to asses model performance avoiding the use of the computationally expensive cross validation, would be to choose the training set by randomly selecting members of the dataset.

4.2.4 Area and Time-Resolution Selection

Orography is known to introduce error in the final product derived from radar reflectivity measurements. For simplicity and to remain within the investigative and non-operational nature of the project I decided to select a limited area in a flat region of Germany completely within the RADOLAN RW product's domain. As it pertains to the choice of 1hr cumulative precipitation data, this was driven by the lack of literature applying deep learning weather prediction at those timescales. Agrawal et al. (2019) uses 7 images at 10min distance from each other taken from the multi radar multi sensor (MRMS) dataset and predicts the MRMS image at 1h after the last image of the input. Ayzel et al. (2020) in contrast uses 4 images at 5min interval from the RADOLAN RY product to predict a similar image 5min after the last one of the input series.

4.2.5 Alternative Comparisons

The model in this thesis was compared only with the so-called persistence prediction that works by assuming that the forecast is equal to the data at the last available time before the forecast. Despite its simplicity, persistence is not a trivial prediction to outperform and serves as first intuitive benchmark to understand the performance of a prediction method. Agrawal et al. (2019) after which this thesis is partly inspired, compared their model with both persistence and the HRRR model. Ayzel et al. (2020) too compared their model against persistence. They also compared it with the results of the `rainymotion` (Ayzel et al. 2019) optical flow library albeit at a 5min time resolution. The `rainymotion` library was shown to outperform the RADVOR (Deutscher Wetterdienst 2022c) operational nowcast model of the DWD (Ayzel et al. 2019). Given the similarity of the dataset between this study and Ayzel et al. (2020) further comparisons could include the accumulated hourly product produced by the RADVOR nowcast. The application of an optical flow algorithm such as `rainymotion` to the 1h accumulated rainfall data needs to be investigated.

4.2.6 Hyperparameters Optimization

For the optimization of the hyperparameters of the model, I applied the manual search using knowledge guided by the literature as regards to the most plausible hyperparameter choice. As discussed in Section 2.2.14 different methods to find the best combination of hyperparameters exist. It is likely that the application of an algorithm such as random search Bergstra and Bengio (2012) could improve the results. However, considering the investigative purpose of this project, the increased amount of time and resources needed to train the model with multiple hyperparameters combinations would be beyond the current scope.

4.2.7 Class Imbalances

An imbalanced dataset is a common problem in classification tasks. The relevance is even more pronounced if the less frequent class is also of higher importance. For this project, as shown in Table 3.1 the high precipitation classes are vastly underrepresented in the dataset compared to the dominant no-rainfall class. The solution I employed, following a common practice in the deep learning community is to assign weights proportional to the inverse of the frequency of the classes for the purpose of computing the loss. However, this choice seems to lead to an overprediction of high rainfall classes. The choice of weights can be considered as part of hyperparameter tuning. A possible alternative way to address the class imbalance problem would be to adopt so-called effective number of samples Cui et al. (2019) that tries to to capture the diminishing marginal benefits by using more data points of a class allowing for less compensation of the imbalance.

4.3 Comparison with Similar Studies

The literature on precipitation nowcasting is fairly recent and therefore the available studies are somewhat limited in number and scope. Nonetheless, some studies that focus on a task similar to the one addressed in this thesis exist. In these studies the precipitation nowcasting problem is almost always addressed at a time resolution of 5min between each input frame. Agrawal et al. (2019) and Ayzel et al. (2020) propose a network architecture (U-Net) and study setup most similar to the one of this project. They both found better performance of their respective models for their tasks compared to persistence, this result remained for increasing lead times in the case of Ayzel et al. (2020) where the output of the model was used as input for subsequent frames. The two studies compared the U-Net to the HRRR model and optical flow respectively with input frames 10min and 5min apart. Such a comparison (i.e. a comparision with a a model used in operational settings) could not be made for the neural network used in this thesis but it is reasonable to assume that with better hyperparameter tuning and small changes to the architecture a similar result could be achieved.

Shi et al. (2017) propose a recurrent architecture based on long and short term memory (LSTM) called TrajGRU. The peculiarity of the model is to learn the connection structure between the hidden states. It utilizes the current input and the previous state to generate a local neighbourhood that should influence a certain location at a every timestamp. The model was compared to traditional Convolutional Neural Networks and to the predecessor of TrajGRU: ConvGRU. In an offline setting a comparison with persistence and the model was made. In all cases the absolute performance of the model decreases as the precipitation rate increases. TrajGRU was found to always outperform all other models as well as persistence.

In the study by Ravuri et al. (2021) the authors test a Generative Adversarial Network (GAN) architecture against three competitive models: a U-Net implementation, PySTEPS (Pulkkinen et al. 2019) and an implementation of MetNet (Sønderby et al. 2020). The GAN consists of two parts, a generator and a discriminator. The generator is responsible for creating images (given an input) that should be as similar as possible to the ones in the dataset. The discriminator has to learn to distinguish between a real example and a generated one. Ravuri et al. (2021) showed that the GAN is chosen by expert forecasters from the Met Office as the one with the highest usefulness compared to the alternative proposed models.

Overall the studies suggest that the deep learning perspective is a promising approach to the precipitation nowcasting problem. In this regard this thesis confirms the identified trajectory and warrants further investigation into the application of neural networks to the 1h time scale. Possible extensions include the study of the behaviour of a U-Net with additional input frames, additional physical variables other than precipitation, additional output frames, the utilization of the model outputs as inputs for extending the lead time of the prediction (as in Ayzel et al. (2020)). All these developments have as a precondition the betterment of the current model, specifically with better hyperparameter tuning and a different weighting strategy for the various precipitation classes.

Appendix A

Software and Hardware

The software was written in *Python 3.8*. The RADOLAN dataset was handled through *wradlib* (Mühlbauer et al. 2022), *xarray* (Hoyer et al. 2022), *dask* (Dask Development Team 2016) and *zarr* (Miles et al. 2020). The confusion matrix was done with the help of *scikit-learn* (Pedregosa et al. 2011). The majority of the plots are produced with *matplotlib* (Caswell et al. 2019) unless specified otherwise. The deep learning library used is *Pytorch* (Paszke et al. 2019) wrapped within *Pytorch-Lightning* (Falcon et al. 2020). The computations partly happend on the Vienna Scientific Cluster[1] 4 (VSC4) compute nodes, partly on VSC3 using the GPU nodes and partly on Google Compute Cloud.

[1]https://vsc.ac.at//systems/

Bibliography

Agrawal, S., L. Barrington, C. Bromberg, J. Burge, C. Gazen, and J. Hickey, 2019: Machine Learning for Precipitation Nowcasting from Radar Images. URL http://arxiv.org/abs/1912.12132.

Aljaafari, N., 2018: Ichthyoplankton Classification Tool using Generative Adversarial Networks and Transfer Learning. Ph.D. thesis.

Ayzel, G., M. Heistermann, and T. Winterrath, 2019: Optical flow models as an open benchmark for radar-based precipitation nowcasting (rainymotion v0.1). *Geoscientific Model Development*, **12 (4)**, 1387–1402, doi:10.5194/GMD-12-1387-2019.

Ayzel, G., T. Scheffer, and M. Heistermann, 2020: RainNet v1.0: a convolutional neural network for radar-based precipitation nowcasting. *Geosci. Model Dev*, **13**, 2631–2644, doi:10.25932/publishup-47294, URL https://doi.org/10.25932/publishup-47294.

Badrinarayanan, V., A. Kendall, and R. Cipolla, 2017: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **39 (12)**, 2481–2495, doi:10.1109/TPAMI.2016.2644615.

Becker, S. and Y. Lecun, 1989: Improving the Convergence of Back-Propagation Learning with Second-Order Methods.

Bergstra, J., R. Bardenet, Y. Bengio, and B. Kégl, 2011: Algorithms for Hyper-Parameter Optimization. *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., Curran Associates, Inc., Vol. 24, URL https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Bergstra, J. and Y. Bengio, 2012: Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, **13 (10)**, 281–305, URL http://jmlr.org/papers/v13/bergstra12a.html.

Bishop, C. M., 2006: *Pattern recognition and machine learning*, Vol. 4. Springer, URL https://www.microsoft.com/en-us/research/people/cmbishop/prml-book/.

Brown, T. B., et al., 2020: Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, **2020-December**, doi:10.48550/arxiv.2005.14165, URL https://arxiv.org/abs/2005.14165v4.

Campbell, M., A. J. Hoane, and F. H. Hsu, 2002: Deep Blue. *Artificial Intelligence*, **134 (1-2)**, 57–83, doi:10.1016/S0004-3702(01)00129-1.

Caswell, T. A., et al., 2019: matplotlib/matplotlib: REL: v3.1.1. doi:10.5281/ZENODO.3264781, URL https://zenodo.org/record/3264781.

Cover, T. M. and J. A. Thomas, 2006: *Elements of Information Theory*. 2d ed., Wiley-Interscience.

Cui, Y., M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, 2019: Class-Balanced Loss Based on Effective Number of Samples. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dask Development Team, 2016: Dask: Library for dynamic task scheduling. URL https://dask.org, URL https://dask.org.

Deutscher Wetterdienst, 2022a: KONRAD - KONvektive Entfwicklung in RADarprodukten. URL https://www.dwd.de/DE/forschung/wettervorhersage/met_fachverfahren/nowcasting/konrad_node.html, URL https://www.dwd.de/DE/forschung/wettervorhersage/met_fachverfahren/nowcasting/konrad_node.html.

Deutscher Wetterdienst, 2022b: RADOLAN - RADAR OnLine ANeichung. URL https://www.dwd.de/DE/leistungen/radolan/radolan.html, URL https://www.dwd.de/DE/leistungen/radolan/radolan.html.

Deutscher Wetterdienst, 2022c: RADVOR (Radar-Niederschlagsvorhersage). URL https://www.dwd.de/DE/leistungen/radvor/radvor.html, URL https://www.dwd.de/DE/leistungen/radvor/radvor.html.

Deutscher Wetterdienst, 2022d: Website of the Deutscher Wetterdienst (DWD). URL https://www.dwd.de/, URL https://www.dwd.de/.

Drozdzal, M., E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal, 2016: The Importance of Skip Connections in Biomedical Image Segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artifi-*

cial Intelligence and Lecture Notes in Bioinformatics), **10008 LNCS**, 179–187, doi:10.1007/978-3-319-46976-8{_}19, URL https://link.springer.com/chapter/10.1007/978-3-319-46976-8_19.

Duchi, J., E. Hazan, and Y. Singer, 2011: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, **12 (61)**, 2121–2159, URL http://jmlr.org/papers/v12/duchi11a.html.

Falcon, W., et al., 2020: PyTorchLightning/pytorch-lightning: 0.7.6 release. doi:10.5281/ZENODO.3828935, URL https://zenodo.org/record/3828935.

Glorot, X. and Y. Bengio, 2010: Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh and M. Titterington, Eds., PMLR, Chia Laguna Resort, Sardinia, Italy, Proceedings of Machine Learning Research, Vol. 9, 249–256, URL https://proceedings.mlr.press/v9/glorot10a.html.

Glorot, X., A. Bordes, and Y. Bengio, 2011: Deep Sparse Rectifier Neural Networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., PMLR, Fort Lauderdale, FL, USA, Proceedings of Machine Learning Research, Vol. 15, 315–323, URL https://proceedings.mlr.press/v15/glorot11a.html.

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

He, K. and J. Sun, 2015: Convolutional neural networks at constrained time cost. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, **07-12-June-2015**, 5353–5360, doi:10.1109/CVPR.2015.7299173.

He, K., X. Zhang, S. Ren, and J. Sun, 2015: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Proceedings of the IEEE International Conference on Computer Vision*, **2015**, 1026–1034, doi:10.48550/arxiv.1502.01852, URL https://arxiv.org/abs/1502.01852v1.

He, K., X. Zhang, S. Ren, and J. Sun, 2016a: Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, **2016-December**, 770–778, doi:10.1109/CVPR.2016.90.

He, K., X. Zhang, S. Ren, and J. Sun, 2016b: Identity Mappings in Deep Residual Networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, **9908 LNCS**, 630–645, doi:10.1007/978-3-319-46493-0{_}38, URL https://link.springer.com/chapter/10.1007/978-3-319-46493-0_38.

Heidke, P., 1926: Calculation of the success and goodness of strong wind forecasts in the storm warning service. *Geogr. Ann. Stockholm*, **8 (1926)**, 301–349.

Hossain, M., B. Rekabdar, S. J. Louis, and S. Dascalu, 2015: Forecasting the weather of Nevada: A deep learning approach. *Proceedings of the International Joint Conference on Neural Networks*, **2015-September**, doi:10.1109/IJCNN.2015.7280812.

Hoyer, S., et al., 2022: xarray. doi:10.5281/ZENODO.6323468, URL https://zenodo.org/record/6323468.

Ioffe, S. and C. Szegedy, 2015: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., PMLR, Lille, France, Proceedings of Machine Learning Research, Vol. 37, 448–456, URL https://proceedings.mlr.press/v37/ioffe15.html.

Karevan, Z. and J. A. K. Suykens, 2018: Spatio-temporal Stacked LSTM for Temperature Prediction in Weather Forecasting. doi:10.48550/arxiv.1811.06341, URL https://arxiv.org/abs/1811.06341v1.

Kingma, D. P. and J. L. Ba, 2014: Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, doi:10.48550/arxiv.1412.6980, URL https://arxiv.org/abs/1412.6980v9.

LeCun, Y., Y. Bengio, and G. Hinton, 2015: Deep learning. *Nature 2015 521:7553*, **521 (7553)**, 436–444, doi:10.1038/nature14539, URL https://www.nature.com/articles/nature14539.

LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller, 1998: Efficient BackProp. *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, Springer-Verlag, Berlin, Heidelberg, 9–50.

Lin, S. Y., C. C. Chiang, J. B. Li, Z. S. Hung, and K. M. Chao, 2018: Dynamic fine-tuning stacked auto-encoder neural network for weather forecast. *Future Generation Computer Systems*, **89**, 446–454, doi:10.1016/J.FUTURE.2018.06.052.

Loshchilov, I. and F. Hutter, 2017: Decoupled Weight Decay Regularization. *7th International Conference on Learning Representations, ICLR 2019*, doi:10.48550/arxiv.1711.05101, URL https://arxiv.org/abs/1711.05101v3.

Maas, A. L., A. Y. Hannun, and A. Y. Ng, 2013: Rectifier nonlinearities improve neural network acoustic models. *IN ICML WORKSHOP ON DEEP LEARNING FOR AUDIO, SPEECH AND LANGUAGE PROCESSING*, URL https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.693.1422.

McCulloch, W. S. and W. Pitts, 1943: A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 1943 5:4*, **5 (4)**, 115–133, doi:10.1007/BF02478259, URL https://link.springer.com/article/10.1007/BF02478259.

Miles, A., et al., 2020: zarr-developers/zarr-python: v2.4.0. doi:10.5281/ZENODO.3773450, URL https://zenodo.org/record/3773450.

Mitchell, T. M., 1997: *Machine Learning*. McGraw-Hill.

Mor-Yosef, S., A. Samueloff, B. Modan, D. Navot, and J. G. Schenker, 1990: Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study. *Obstetrics and Gynecology*, **75 (6)**, 944–947, URL https://europepmc.org/article/med/2342742.

Mühlbauer, K., et al., 2022: wradlib/wradlib: wradlib-release v1.15.0. doi:10.5281/ZENODO.6449103, URL https://zenodo.org/record/6449103.

NOAA, 2021: High-Resolution Rapid Refresh model. URL https://rapidrefresh.noaa.gov/hrrr/, URL https://rapidrefresh.noaa.gov/hrrr/.

Nocedal, J. and S. J. Wright, 2006: *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering, Springer New York, doi:10.1007/978-0-387-40065-5, URL http://link.springer.com/10.1007/978-0-387-40065-5.

Papers with Code, 2022: An Overview of Activation Functions — Papers With Code. URL https://paperswithcode.com/methods/category/activation-functions, URL https://paperswithcode.com/methods/category/activation-functions.

Paszke, A., et al., 2019: PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 8024–8035, URL http://papers.neurips.cc/paper/

9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pedregosa, F., et al., 2011: Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, **12 (85)**, 2825–2830, URL http://jmlr.org/papers/v12/pedregosa11a.html.

Prechelt, L., 1998: Early Stopping - But When? *Neural Networks: Tricks of the Trade*, K.-R. Orr Genevieve B.
}and Müller, Ed., Springer Berlin Heidelberg, Berlin, Heidelberg, 55–69, doi:10.1007/3-540-49430-8{_}3, URL https://doi.org/10.1007/3-540-49430-8_3.

Pulkkinen, S., D. Nerini, A. A. Pérez Hortal, C. Velasco-Forero, A. Seed, U. Germann, and L. Foresti, 2019: Pysteps: An open-source Python library for probabilistic precipitation nowcasting (v1.0). *Geoscientific Model Development*, **12 (10)**, 4185–4219, doi:10.5194/GMD-12-4185-2019.

Qian, N., 1999: On the momentum term in gradient descent learning algorithms. *Neural Networks*, **12 (1)**, 145–151, doi:10.1016/S0893-6080(98)00116-6.

Qiu, M., P. Zhao, K. Zhang, J. Huang, X. Shi, X. Wang, and W. Chu, 2017: A short-term rainfall prediction model using multi-task convolutional neural networks. *Proceedings - IEEE International Conference on Data Mining, ICDM*, **2017-November**, 395–404, doi:10.1109/ICDM.2017.49.

Ravuri, S., et al., 2021: Skillful Precipitation Nowcasting using Deep Generative Models of Radar. URL http://arxiv.org/abs/2104.00954.

Reichstein, M., G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, and Prabhat, 2019: Deep learning and process understanding for data-driven Earth system science. *Nature*, **566 (7743)**, 195–204, doi:10.1038/s41586-019-0912-1, URL https://doi.org/10.1038/s41586-019-0912-1.

Ren, X., X. Li, K. Ren, J. Song, Z. Xu, K. Deng, and X. Wang, 2021: Deep Learning-Based Weather Prediction: A Survey. *Big Data Research*, **23**, 100 178, doi:10.1016/j.bdr.2020.100178.

Ronneberger, O., P. Fischer, and T. Brox, 2015: U-Net: Convolutional Networks for Biomedical Image Segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, **9351**, 234–241, URL https://arxiv.org/abs/1505.04597v1.

Rosenblatt, F., 1958: The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, **65 (6)**, 386–408, doi:10.1037/H0042519, URL /record/1959-09865-001.

Rossi, R. J., 2018: *Mathematical statistics : an introduction to likelihood based inference.*

Shelhamer, E., J. Long, and T. Darrell, 2017: Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **39 (4)**, 640–651, doi:10.1109/TPAMI.2016.2572683.

Shi, X., Z. Chen, H. Wang, D. Y. Yeung, W. K. Wong, and W. C. Woo, 2015: Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *Advances in Neural Information Processing Systems*, **2015-January**, 802–810, doi:10.48550/arxiv.1506.04214, URL https://arxiv.org/abs/1506.04214v2.

Shi, X., Z. Gao, L. Lausen, H. Wang, D.-Y. Yeung, W.-K. Wong, W.-C. Woo, and H. Kong Observatory, 2017: Deep Learning for Precipitation Nowcasting: A Benchmark and A New Model. Tech. rep. URL https://arxiv.org/abs/1706.03458.

Sønderby, C. K., et al., 2020: MetNet: A Neural Weather Model for Precipitation Forecasting. URL http://arxiv.org/abs/2003.12140.

Srivastava, R. K., K. Greff, and J. Schmidhuber, 2015: Highway Networks. doi:10.48550/arxiv.1505.00387, URL https://arxiv.org/abs/1505.00387v2.

Steffenel, L. A., V. Anabor, D. Kirsch Pinheiro, L. Guzman, G. Dornelles Bittencourt, and H. Bencherif, 2021: Forecasting upper atmospheric scalars advection using deep learning: an O3 experiment. *Machine Learning*, 1–24, doi:10.1007/s10994-020-05944-x, URL https://doi.org/10.1007/s10994-020-05944-x.

Wang, C. and Y. Hong, 2018: Application of spatiotemporal predictive learning in precipitation nowcasting. *AGU Fall Meeting Abstracts.*

Wang, Y., Z. Gao, M. Long, J. Wang, and P. S. Yu, 2018: PredRNN++: Towards A Resolution of the Deep-in-Time Dilemma in Spatiotemporal Predictive Learning. Tech. rep., 5123–5132 pp. URL http://proceedings.mlr.press/v80/wang18b.html.

Wang, Y., M. Long, J. Wang, Z. Gao, and P. S. Yu, 2017: PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs. *Advances* *in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., Vol. 30, URL https://proceedings.neurips.cc/paper/2017/file/e5f6ad6ce374177eef023bf5d0c018b6-Paper.pdf.

Weigl, E., 2015: Radarniederschlag - Prinzip der Niederschlags-bestimmung mit Radar inkl. Umrechnung der Radarreflektivitäten in Momentanwerte des Niederschlages. Tech. rep., Deutscher Wetterdienst. URL https://www.dwd.de/DE/leistungen/radarniederschlag/rn_info/download_niederschlagsbestimmung.pdf?__blob=publicationFile&v=4.

Weigl, E., 2021a: RADOLAN Änderungen der operationellen Routine. Tech. rep., Deutscher Wetterdienst. URL https://www.dwd.de/DE/leistungen/radolan/radolan_info/radolan_aenderungen_pdf.pdf?__blob=publicationFile&v=17.

Weigl, E., 2021b: RADOLAN/RADVOR Hoch aufgelöste Niederschlagsanalyse und –vorhersage auf der Basis quantitativer Radar- und Ombrometerdaten für grenzüberschreitende Fluss-Einzugsgebiete von Deutsch-land im Echtzeitbetrieb Beschreibung des Kompositformats Version 2.5.3. Tech. rep., Deutscher Wetterdienst. URL https://www.dwd.de/DE/leistungen/radolan/radolan_info/radolan_radvor_op_komposit_format_pdf.pdf?__blob=publicationFile&v=16.

Weigl, E., T. Reich, P. Lang, A. Wagner, O. Kohler, and N. Gerlach, 2004: Projekt RADOLAN - Routineverfahren zur Online-Aneichung der Radarniederschlagsdaten mit Hilfe von automatischen Bodenniederschlagsstationen (Ombrometer). Tech. rep., Deutscher Wetterdienst. URL https://www.dwd.de/DE/leistungen/radolan/radolan_info/abschlussbericht_pdf.pdf?__blob=publicationFile&v=2.

Wilks, D. S., 2011: Statistics. *Statistical Methods in the Atmospheric Sciences*, **100**, 100.

Wilson, D. R. and T. R. Martinez, 2003: The general inefficiency of batch training for gradient descent learning. *Neural Networks*, **16 (10)**, 1429–1451, doi:10.1016/S0893-6080(03)00138-2.

Xu, B., N. Wang, H. Kong, T. Chen, and M. Li, 2015: Empirical Evaluation of Rectified Activations in Convolutional Network. doi:10.48550/arxiv.1505.00853, URL https://arxiv.org/abs/1505.00853v2.

Zeiler, M. D., 2012: ADADELTA: An Adaptive Learning Rate Method. doi:10.48550/arxiv.1212.5701, URL https://arxiv.org/abs/1212.5701v1.

Acknowledgments

I would like to thank my incredible supervisor Prof. Georg Mayr for the continuous help and advice on every part of the thesis. Thank you for your support. I'd also like to thank Matthias Göbel, Daniel Frisinghelli and Prof. Haltmeier for the suggestions on the neural network. Thanks to all my classmates, especially to Michele Giurato, for having shared with me this very important part of my life. I'm grateful to my family, for them being by my side. Most importantly thank you to Gabi, for having helped me in finding new energy when everything seemed lost.