VRIJE
UNIVERSITEIT
BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Applied Sciences and Engineering: Applied Computer Science

NOWCASTING OF SEVERE THUNDERSTORMS OVER LAKE VICTORIA

Prediction of nightly thunderstorm activity over Lake Victoria through deep learning on satellite-derived images

Seppe Lampe

August 2021

Supervisor: Inne Vanderkelen
Co-promotors: dr. Steven Dewitte and prof. dr. Adrian Munteanu
Promotor: prof. dr. Wim Thiery

Engineering

Acknowledgements

Writing this dissertation was a group effort, I was able to write and complete this thesis with the support of many people. It would have never existed and/or reached its current state without each of these heroes. Thank you!

First and foremost, Prof. Dr. Wim Thiery has provided me with this exciting opportunity and allowed me to work as a MSc thesis student under his supervision this year. Throughout the last year we had countless of online meetings to discuss progress, difficulties and any questions I had. Moreover, Wim offered me the opportunity to obtain a grant and continue working on his team next year as a PhD Candidate. I sincerely hope the funding comes through and we can continue our work together in the coming years. Thank you Wim for the opportunity and the trust you placed and continue to place in me.

However, I not only have Wim to thank but also Inne Vanderkelen, my supervisor for this thesis. Inne has joined Wim and me in every meeting and provided us with much help throughout this thesis. Inne has been my go-to contact point all year and has spent a lot of time helping me write this dissertation. Inne has brought structure, transparency and refinement to this thesis. Thank you Inne for the countless hours you have spent guiding me in this project!

Of course I cannot forget my co-promotors Dr. Steven Dewitte and Prof. Dr. Adrian Munteanu who have provided us with help when needed. Thank you both.

Besides these direct contributors, many others have aided me throughout the last couple of years. First, I want to thank all professors that I encountered on my path. I owe a lot of my insights, skills and interests to enthusiastic and helpful teachers who inspired me to expand my horizon and dig deeper. Without them, there would be no MACS, there would be no thesis and I would not have become the scientist I am today. Thank you!

Furthermore, a lot of practical exercise is required to become a computer scientist. Therefore, the MACS programme contains many exercise sessions and projects, most of which are guided by teaching assistants (TAs). Almost all exercise sessions were clearly organised and sometimes even offered multiple times per week to ensure the best personal guidance for all students. This, combined with constructive feedback on lab/project reports, is one of the main ways we grew as computer scientists. I realise that this requires a lot of effort, therefore thank you to all TAs for all your support and the time you invested in us!

Throughout the last two years I also had the pleasure of going through the MACS programme with an amazing group of classmates. Even though we only had ~one semester of in-person classes most of us have kept close contact, helping each other where and whenever we could. Thank you to all my classmates, you have been a source of support, information and camaraderie in these strange times.

I am also lucky to have two fantastic, understanding parents rooting for me 100% of the time. My mom and dad have supported me in numerous ways, they cheer me up, push me forward and are always ready when I need them the most. My gratitude towards them is too large to put into words. Thank you sincerely for everything you mean to me. You are the best.

Besides a fantastic family, I also have a wonderful group of friends. Supporting, understanding and a bright light in times of need, you all make life so much more worth it. Even though recently I haven't seen most of you as often as I should, you all kept supporting me (and counting down the days to this deadline). Thank you for keeping my spirits up and see you all soon!

Lastly, I want to thank my girlfriend, Cynthia, for keeping me sane throughout this Covid-ridden year full of social distancing, lockdowns and isolation. I also haven't been able to spend as much time with you as I wanted to these last couple of months. I promise that will change now and I look forward to moving into our new place together. I love you with all my heart, thank you for being you.

I would like to end my gratitudes with a playful quote from one of my closest friends, which makes me smile (and agree more) every time I think about it.

Writing your MSc thesis drives you crazy, wanting to write a second proves you are. - P.T.

Abstract

Every year 1000-5000 fishermen die on Lake Victoria as a consequence of intense nighttime thunderstorms. The lack of a well-functioning weather observation network prohibits the operation of traditional high-resolution weather forecasting systems. Previous studies uncovered a relationship between the afternoon storm intensity around Lake Victoria and the storm intensity over the lake in the following night. The Lake Victoria Intense storm Early Warning System (VIEWS) project exploits this dependency by forecasting the occurrence of a 99^{th} percentile thunderstorm-intense night based on the preceding afternoon weather conditions. VIEWS operates on a satellite-derived dataset of overshooting tops, a proxy for thunderstorms, which was designed by NASA in 2010. This overshooting top detection algorithm received substantial upgrades in 2015 and 2021. Here, we evaluate whether the VIEWS project still provides reliable forecasts on this updated dataset and what the effects are on its performance. Furthermore, do we investigate the possibility of improving and extending VIEWS through machine and deep learning methods.

First, a correlation-based analysis is performed to determine when and where the relationship between the afternoon thunderstorm activity and subsequent nighttime thunderstorm intensity is maximal. Then, we replicate the univariate logistic regression model of VIEWS but use the 2021 version of the overshooting top algorithm. This yields a reduced performance on the upgraded dataset. Next, the same univariate logistic regression is performed considering only the highly-correlated input features derived in the first step. This preprocessing significantly improves the performance of the univariate classifier. Next, the highly-correlated features are summed into a number of aggregates, based on their time interval or location. These temporal and spatial aggregates are subsequently used in different multivariate logistic regression models. These classifiers outperform the univariate model, yielding a performance that is close to the original effectiveness of VIEWS. Additionally, we evaluate the use of neural networks to perform the same classification. Individual neural networks show promising results for this task. However, their ensembles performed significantly worse in detecting extreme events, most likely due to a lack in diversity of the considered models. Lastly, convolutional neural networks were designed to make spatially-explicit predictions of nightly thunderstorm intensity over the lake. Preliminary results have not yet shown satisfactory performance of these models, but their full potential remains to be investigated.

Table of Contents

Prefix

Abbreviations

AI Artificial Intelligence.

ANN Artificial Neural Network.

AUC Area Under the Curve.

BT Brightness Temperature.

BTTD Brightness Temperature minus Tropopause Difference.

CLI Command Line Interface.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

DFID UK Department for International Development.

DL Deep Learning.

EAT East African Time.

ELU Exponential Linear Unit.

GPU Graphics Processing Unit.

HDF5 Hierarchical Data Format version 5.

HIGHWAY HIGH impact Weather lAke sYstem.

HIWC High Ice Water Content.

HPC High-Performance Computing.

HRV High-Resolution Visible.

IR Infrared.

IRBT Infrared Brightness Temperature.

ITCZ Intertropical Convergence Zone.

LaRC Langley Research Center.

MERRA-2 Modern-Era Retrospective analysis for Research and Applications, Version 2.

ML Machine Learning.

MSE Mean Squared Error.

netCDF4 Network Common Data Format version 4.

NN Neural Network.

NWP Numerical Weather Prediction.

OR Odds Ratio.

OT Overshooting Top.

POSIX Portable Operating System Interface.

RAM Random Access Memory.

ReLU Rectified Linear Unit.

RMI Belgian Royal Meteorological Institute.

RNN Recurrent Neural Network.

ROC Receiver Operating Characteristic.

rpm rotations per minute.

SEVIRI Spinning Enhanced Visible and InfraRed Imager.

TA4 Tropical Africa 4.4km Model.

tanh hyperbolic tangent.

TPU Tensor Processing Unit.

TT Tropopause Temperature.

VIEWS Lake Victoria Intense storm Early Warning System.

VNIR Visible and Near InfraRed.

WMO World Metereological Organisation.

List of Figures

Chapter 1

Introduction

Lake Victoria is a shallow (<100 m) freshwater lake located in East-Africa at the border of Tanzania, Uganda & Kenya (Figure 1.1). Covering ~68 000 km^2, it is Africa's largest lake and second-largest freshwater lake in the world (Kite, 1982; Vanderkelen et al., 2018; Sichangi & Makokha, 2017). The fishing industry of Lake Victoria feeds 30 million people, hosts 200 000 fishermen and accounts for ~2.8%, 2.5% and 0.5% of the GDP of Uganda, Tanzania and Kenya respectively (Njiru et al., 2018; Cannon, 2014; LVFO, 2016). Lake Victoria's fishermen mainly sail out to go fishing at night (Kudhongania & Cordone, 1974; Kwena et al., 2012; Mills et al., 2014). However, according to the Red Cross and the UK Met Office, 1000-5000 of these fishermen die each year because of boating accidents during severe thunderstorms (Semazzi, 2011; Watkiss et al., 2020). The high wave activity and lightning associated with the thunderstorms are the primary cause of these life-threatening boating accidents (Semazzi, 2011; Cannon, 2014; Watkiss et al., 2020). For each deceased fisherman, on average eight relatives are left behind without an income (Semazzi, 2011).

In the coming decade, urbanisation will continue to spread along the lakefront, which will lead to an increase of fishing communities exposed to these natural hazards (Seto et al., 2012). Furthermore, climate projections show a strong increase in thunderstorm intensity over Lake Victoria in the near-future (Thiery et al., 2016). Currently, these thunderstorm events already cause an unacceptably large number of fatalities, and have substantial adverse impacts on the socio-economic development of the communities around the lake. In the coming decade(s) more people will be affected by these thunderstorms, which will increase in intensity. An early warning system for thunderstorms would severely benefit this region, to prevent fishermen to sail out on nights with severe storm activity. The lack of an elaborate weather observation network however hampers traditional forecasting methods such as Numerical Weather Prediction (NWP) forecasts (IOC et al., 2019).

Recently, the HIGH impact Weather lAke sYstem (HIGHWAY) project started providing spatio-temporal storm forecasts twice per day to the communities around Lake Victoria (Watkiss et al., 2020). HIGHWAY is managed by the World Metereological Organisation (WMO) and funded by the UK Department for International Development (DFID). The project employs a NWP system to generate spatio-temporal storm prediction maps. NWPs rely on in-situ measurements and accurate mathematical models to simulate future weather conditions based on the current weather (Bauer et al., 2015). However, Lake Victoria and its surrounding region lack sufficient and appropriate weather monitoring stations (IOC et al., 2019). HIGHWAY has made efforts

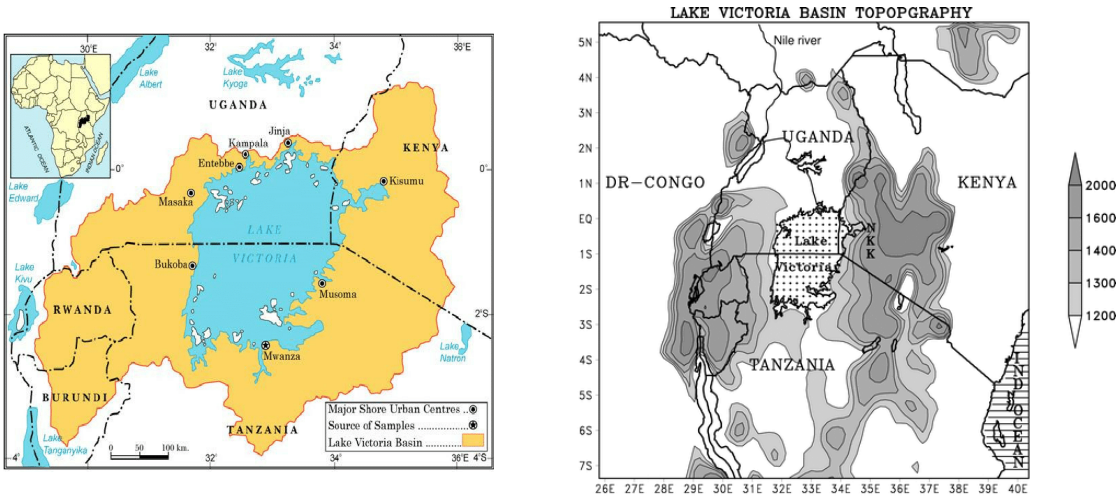


Figure 1.1: (left) Geographical setting of Lake Victoria and (right) the topography of the region surrounding Lake Victoria (Adapted from Anyah et al., 2006; Gumisiriza et al., 2009).

to improve this situation through a field campaign but there is no indication on how many measuring stations the project has set-up. HIGHWAY estimates that it is currently reducing the fatalities of thunderstorms over Lake Victoria by ~30%, showing the high impact of the project (Watkiss et al., 2020). However, only ~70% of the storms are currently being predicted, leaving room for improvement (Watkiss et al., 2020; Hanley et al., 2021).

The Lake Victoria Intense storm Early Warning System (VIEWS) project takes a different approach (Thiery et al., 2017). Lake Victoria has a distinct diurnal weather cycle, where most afternoon thunderstorms occur over land, while at night thunderstorm activity is concentrated over the lake (Figure 1.2; Thiery et al., 2015; Thiery et al., 2016). Thiery et al. (2016) revealed and explained a link between afternoon storm intensity and the subsequent nightly storm intensity. This afternoon control on the nighttime storm activity indicates an inherent predictability in the weather system. VIEWS aims to predict the nighttime storm activity over Lake Victoria by exploiting its relationship with the preceding afternoon thunderstorm intensity (Thiery et al., 2017). VIEWS employs logistic regression on a satellite-derived thunderstorm proxy called Overshooting Tops (OTs) to predict whether the following night will be an extreme thunderstorm night, defined as a 1% most thunderstorm-intense night. However, as this system only focuses on the 1% most extreme thunderstorm events and delivers no spatially explicit information, no practical implementation of VIEWS is currently in use. Recently, researchers of NASA's Langley Research Center (LaRC) released a new and improved version of the OT detection algorithm (Khlopenkov et al., 2021).

A reliable storm forecast or early warning system would be highly beneficial to the communities around Lake Victoria and boost the region's economic development. Currently, two systems exist, the NWP-based HIGHWAY and satellite-derived VIEWS. Improvements to these systems would further reduce the substantial death rate on Lake Victoria and provide socio-economic benefits to the entire region.

Thanks to recent scientific progress, including improved satellite-derived storm detection and the development of (new) deep learning techniques, it is now possible to investigate new possibilities and push the limits of storm forecasting over Lake Victoria forward (Schön & Dittrich, 2019;

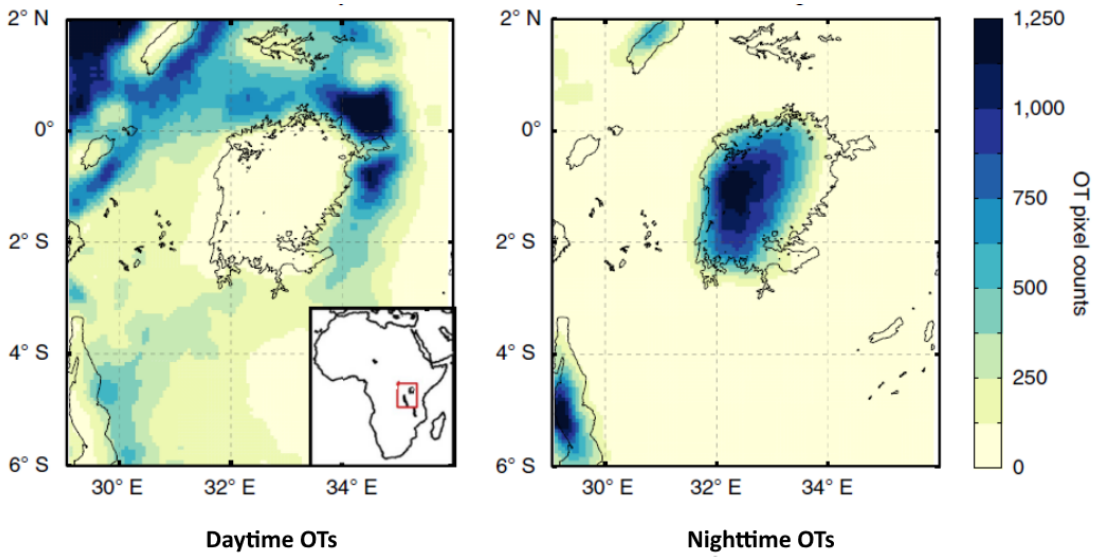


Figure 1.2: Diurnal weather cycle for the region around Lake Victoria, where afternoon thunderstorms mainly occur over land, while the nighttime thunderstorm activity is concentrated over the lake. Overshooting Tops (OTs) are a proxy for thunderstorms (Adapted from Thiery et al., 2016).

K. Zhou et al., 2019; K. Zhou et al., 2020; Liu et al., 2021). In this thesis, we evaluate the effect of the updated OT detection algorithm on VIEWS and consider the possibility of improving and extending the VIEWS project through machine and deep learning. First, we empirically review the effect of the new OT detection scheme on the performance of VIEWS. Then, we consider improving the thunderstorm predictions via a correlation-based filtering of the input features. Subsequently, we investigate whether multivariate logistic regression can outperform the univariate logistic regression of VIEWS. Lastly, we evaluate the possibility of introducing neural networks to improve the performance of VIEWS and to extend the project by including spatially-explicit predictions for the nighttime thunderstorms over Lake Victoria.

Chapter 2

Theoretical Framework

This thesis combines scientific knowledge from atmospheric science, deep learning and earth observation. This chapter provides the necessary background information on the used concepts and methods from the different disciplines. First, the mesoscale weather dynamics over Lake Victoria are explained. Then, OTs and their detection algorithm are discussed, which is followed by an overview of the state of the art in forecasting thunderstorms over Lake Victoria. In the last section, the machine and deep learning concepts relevant to this thesis are introduced.

2.1 Lake Victoria and its Weather Dynamics

This section describes the weather patterns of the region around Lake Victoria along with its future predictions. The diurnal weather cycle is discussed in high detail, as it is crucial for the system developed in this thesis.

Lake Victoria is located between -3° and 0.5° latitude and has a tropical climate (Verschuren et al., 2002). On average, annual precipitation over the lake is ~1500–2000 mm, while the land around Lake Victoria receives ~1000–1500 mm rainfall per year (Figure 2.1; Kizza et al., 2012; Thiery et al., 2015). The seasonal climate around Lake Victoria is governed by the transition of the Intertropical Convergence Zone (ITCZ). From March to May, when the ITCZ is located north of the region, a first rainfall season (long rains) occurs (Figure 2.1). A second, less intense rainfall season (short rains) happens during October-December, when the ITCZ is positioned in the south over Kenya (Mungai, 1984; Song et al., 2004; Kizza et al., 2009).

Precipitation and thunderstorms exhibit a clear diurnal cycle over Lake Victoria and its surrounding region (Figure 1.2). This diurnal cycle is the result of land and lake breezes, which are driven by the diurnal thermal gradient between the lake's water surface and the surrounding land (Savijärvi, 1997; Thiery et al., 2015). During the day, land heats up faster than water, which causes relatively more air to rise, creating a lower atmospheric pressure over land. This pressure difference generates a lake breeze with winds coming from the lake to the land. This induces convection over the land during the day. At night, the land cools faster than the lake surface, reversing the lake breeze into a land breeze, going from land to the lake, leading to convergence over the lake. In addition, the mountains east of the lake strengthen the convergence by generating upslope (anabatic) winds during the day and downslope (katabatic) winds at night (Anyah et al., 2006; Williams et al., 2015; Van de Walle et al., 2020). Anabatic winds are

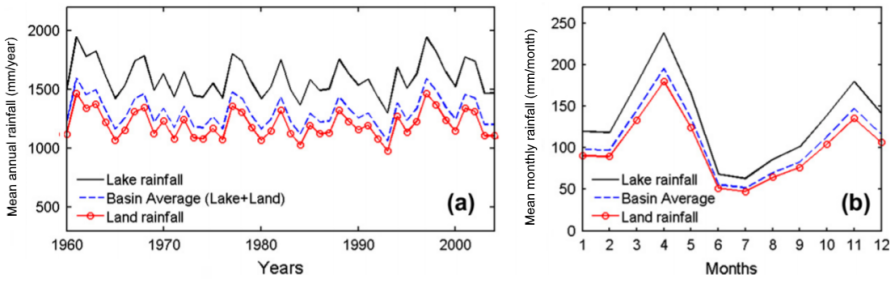


Figure 2.1: Annual and monthly precipitation over Lake Victoria (Adapted from Kizza et al., 2012).

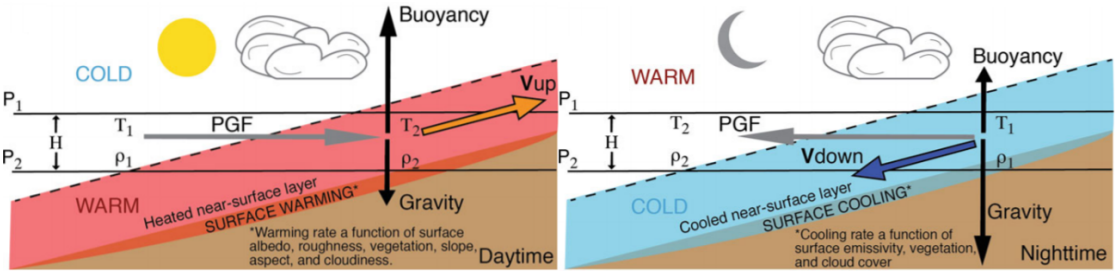


Figure 2.2: Mountain winds (left) upslope (anabatic) during the day, (right) downslope (katabatic) during the night (Adapted from Hatchett et al., 2019).

caused by the warming of the slope surface, which heats the overlying air and causes it to raise. Oppositely, during the night, the cool slope surface cools the overlying air, causing it to descend downslope (Figure 2.2). This convergence forces the upward movement of moisture-carrying air, which subsequently reaches lower atmospheric pressures and undergoes adiabatic decompression, exhibiting work on its environment. As a consequence, the air loses part of its energy and cools down. Colder air can hold less moisture, ultimately leading to condensation and the formation of water droplets (Byers & Rodebush, 1948; C.-C. Wang et al., 2019; Virts & Goodman, 2020).

Peak convection over the lake occurs between 4 and 9 a.m. (Ba & Nicholson, 1998; Song et al., 2004; Thiery et al., 2017). The nightly thunderstorms over Lake Victoria are the result of convergence (74%) and moisture advection (26%) of the nighttime land breeze over the lake (Thiery et al., 2016). Furthermore, there exists a strong connection between the intensity of the nighttime lake storms and the preceding daytime land storm activity (Thiery et al., 2015; Thiery et al., 2016). Intense afternoon land storms induce a moisture anomaly in the lower atmosphere and cool the land surface, which reduces the lake breeze and associated moisture divergence from over the lake. Additionally, the colder land surface not only weakens the lake breeze but also strengthens the nighttime land breeze, increasing nighttime convergence, which, along with the higher moisture availability, strengthens the thunderstorm activity over the lake (Thiery et al., 2016).

A future climate projection for the Lake Victoria basin shows that mean precipitation will decrease, while extreme precipitation will increase (Thiery et al., 2016). Thiery et al. (2016) analyses a 7km resolution, coupled lake-land-atmosphere climate projection of the region around Lake Victoria and coarser-scale ensemble projections for the end of the 21st century. They find that mean precipitation will decrease by 6% but extreme precipitation will increase under a high-

emission scenario. Additionally, the increase of extreme precipitation over the lake is estimated to be 2.4 times higher than that of the land surrounding the lake. A present-day 1-in-15-year precipitation event (thunderstorm) would become a 1-in-1.5-year (or possibly even 1-in-0.8-year) event. Thiery et al. (2016) relate the decrease in mean precipitation to the changes in mesoscale dynamics caused by faster warming land surfaces. The increase in extreme rainfall events is attributed to the increase in moisture availability over the lake. This predicted intensification of extreme precipitation events further increases the importance of a well-functioning early warning system for the region.

2.2 Overshooting Tops

OTs are dome-shaped protrusions on top of a cumulonimbus anvil (Figure 2.3). OTs are formed by intense updraughts through the tropopause into the lower stratosphere (Proud, 2015). These intense updraughts occur in convective thunderstorms which often cause hazardous weather at the surface (Reynolds, 1980; Negri & Adler, 1981; K. Bedka et al., 2010). OTs can be well recognized in satellite images, as they are isolated areas of lower Infrared Brightness Temperatures (IRBTs) relative to the warmer surrounding anvil cloud (K. Bedka et al., 2010). Therefore, OTs are a relevant proxy to asses thunderstorm activity from satellite products. In this thesis, OTs are used to estimate thunderstorm activity and form the basis for the predictions of our models.

As OTs are fundamental to this thesis, we spend sufficient time explaining how they are obtained from Spinning Enhanced Visible and InfraRed Imager (SEVIRI) satellite data. SEVIRI is a geostationary satellite centered at 0°longitude, which collects data at 3km resolution every 15 minutes. A simple OT detection scheme, which evaluates each pixel separately based on a threshold or signature, can never be effective for the following reasons: (i) a pixel in a convective OT can have an identical spectral signature as a pixel in an ordinary cirrus cloud and (ii) no single threshold is effective for the detection of convection across a wide range of latitudes and environmental conditions (K. Bedka et al., 2010; Setvák et al., 2013; Khlopenkov et al., 2021). Therefore, OT detection algorithms are required to consider a wide variety of local and regional characteristics to determine if a pixel is an OT.

In 2010, NASA's LaRC designed an automated OT detection algorithm for detecting OTs from i.a., SEVIRI (K. Bedka et al., 2010). This algorithm operates in four steps. First, 'cold pixels'

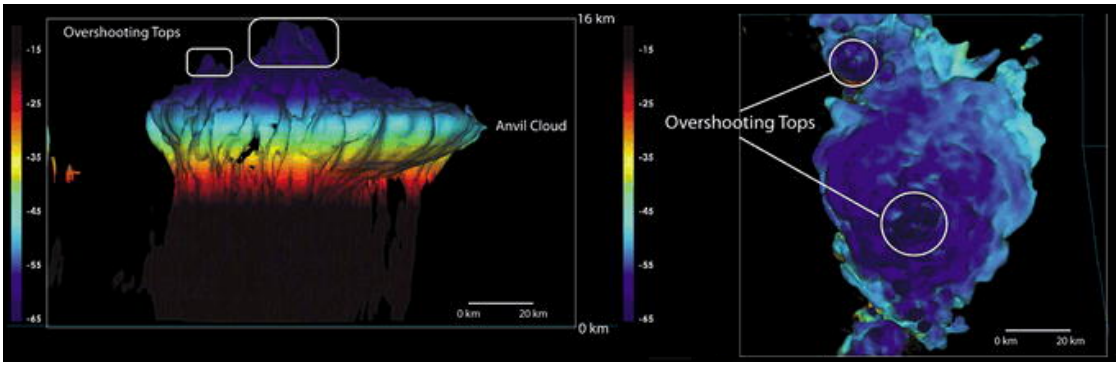


Figure 2.3: Example of an overshooting top (Adapted from K. Bedka et al., 2010). The tropopause is located at the anvil cloud.

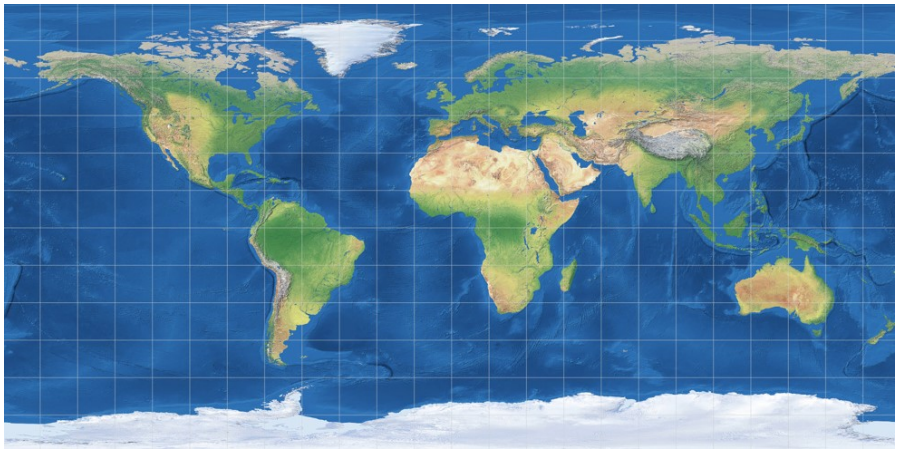


Figure 2.4: Equirectangular projection of the world (by Tobias Jung).

are selected from the satellite's Infrared (IR) band if they have an IRBT $\leq$ 215K, which is contemporarily lower than the surrounding tropopause. Secondly, the pixels are ordered based on their IRBT, with the lowest coming first. Thirdly, the pixels that are within 15km of a colder pixel are discarded, as well as those that are not part of an anvil cloud. Lastly, the remaining pixels are considered OTs if they are at least 6.5K colder than the surrounding anvil IRBT.

In 2016, this algorithm received a significant upgrade, most importantly was the introduction of probabilistic prediction for OTs instead of the previous binary prediction method (K. Bedka & Khlopenkov, 2016). Recently, the same authors have published the third version of the OT detection algorithm, which further improves the performance of this OT detection scheme (Khlopenkov et al., 2021). The Lake Victoria OT dataset used in this thesis is compiled using the newest version of NASA's OT detection algorithm.

The LaRC OT detection algorithms operate on satellite images having an IR channel with a central wavelength of 10.3-11.5 µm like, SEVIRI. As the algorithms depend on spatial context, it is important that the shape of clouds is preserved in the satellite image. Therefore, the input satellite maps are reprojected to a conformal map projection, which is shape-preserving. The algorithm reprojects the input images to the equirectangular projection, which projects longitudinal meridians to vertical lines and circles of latitude to straight horizontal lines (Figure 2.4). Although this projection is non-conformal by nature, it can be regarded as conformal at low to mid latitudes, where most storms occur and our region of interest is located (Khlopenkov et al., 2021). Furthermore, the popularity of the equirectangular projection allows to simply cross-match against other meteorological products. For SEVIRI images, the output is eventually reprojected to 28 pixels/degree (0.036°) latitude and longitude (Khlopenkov et al., 2021).

Summarized, the newest version of the algorithm works as follows: the IRBT obtained from SEVIRI is first transformed to its relative value compared to the Tropopause Temperature (TT). From this, anvil clouds are detected and based on that, cold spots in the anvil clouds. Around these cold spots a spatial analysis quantifies a number of parameters such as anvil area, anvil temperature, prominence of cold spots, etc., which are combined to estimate the probability that a cold spot is an OT. The sections below describe the algorithm of Khlopenkov et al., 2021 in more detail.

In the first step, the IRBT from SEVIRI is compared to the TT. Thus, an estimation of the

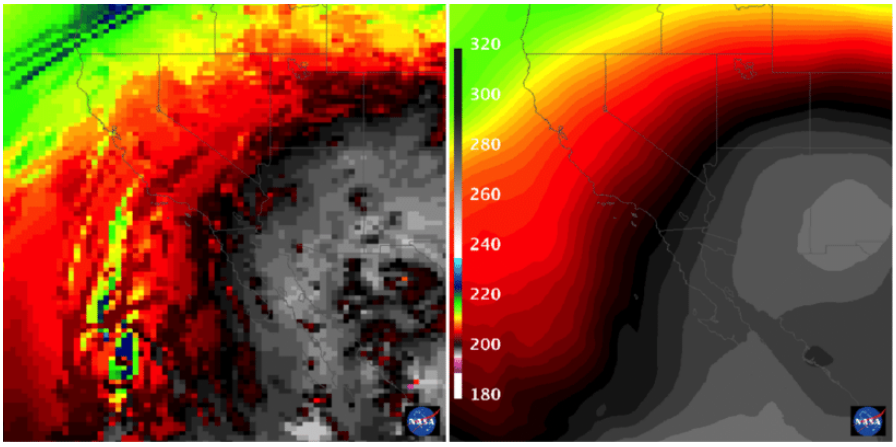


Figure 2.5: Smoothing of the TT with the original MERRA-2 tropopause product on the left, and the smoothened version on the right (Khlopenkov et al., 2021).

TT needs to be obtained, which varies spatially and temporally. Modern-Era Retrospective analysis for Research and Applications, Version 2 (MERRA-2) contains a reanalysis of the TT (Gelaro et al., 2017). However, this product often yields unrealistic values in single pixels, which is detrimental for storm detection. Therefore, a spatial filtering is applied to this layer to smoothen the TT field (Figure 2.5). Subsequently, a so-called Brightness Temperature (BT) score is calculated via the formula in Equation 2.1, which scales the input map from SEVIRI according to the Brightness Temperature minus Tropopause Difference (BTTD) (Figure 2.6).

$$BT = (60 - BTTD) \cdot 340 \quad \text{with} \quad BTTD = IRBT - TT \tag{2.1}$$

The following step generates an anvil mask. First, a 22 km circular disk is analysed for every other pixel in every other line. The BT-score values of this disk are binned into 32 groups and the histogram is inspected. BTs in anvils have uniformly spread, cold temperatures, which results in a sharp peak in the histogram. The anvil rating, a value between 0 and 255, indicates how likely a pixel belongs to an anvil and is calculated via Equation 2.2.

$$r_{anvil} = C_H \cdot H_i \cdot i \cdot (2N + 8 - i) \tag{2.2}$$

Where r_{anvil} is the anvil rating, i the bin number, C_H a normalization coefficient equal to $0.35/D^2$ with D equal to the amount of pixels in the diameter of the histogram window, H_i the height (number of counts) in bin i and N the number of bins ($N = 32$). This however results in an unsatisfactory non-uniform pattern (Figure 2.7a, which is caused by the OTs in the anvil. As OTs are colder than the anvil, the presence of an OT will cause the peak to be distributed along several bins and reduce the anvil rating of the OT pixels but also of the surrounding pixels. This issue can be solved by considering the n most filled bins (Equation 2.2). Empirically a value of 3 has been defined as optimal for n. This results in Equation 2.3, which uses a normalization coefficient equal to $0.22/D^2$ (Figure 2.7b). The white border on Figure 2.7b indicates the anvil boundary using a BT-score of 16000 (T difference of 13K). The orange region almost reaches the white border but the pixels at the edge of the anvil have a distinct lower BT-score. This is easily explained by the relatively large fraction of non-anvil pixels in the disk of these pixels,

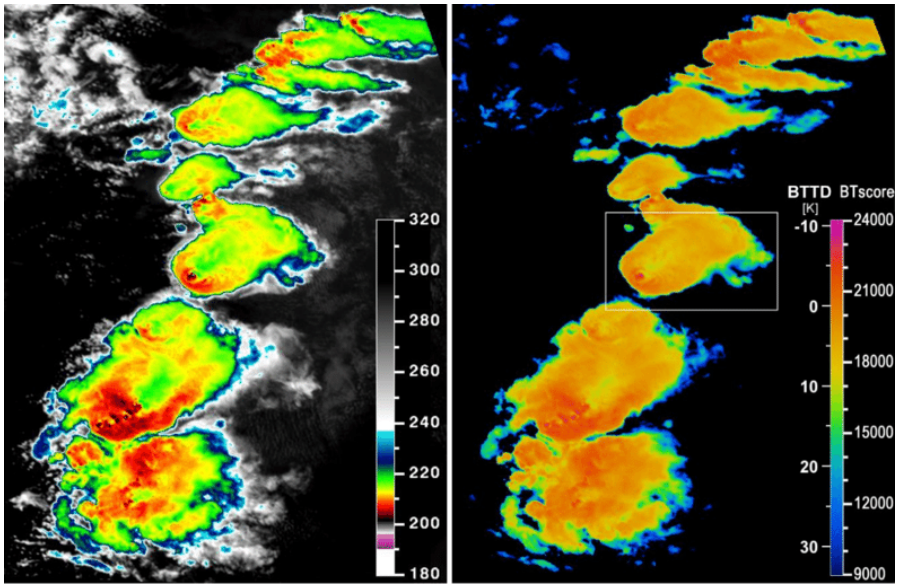
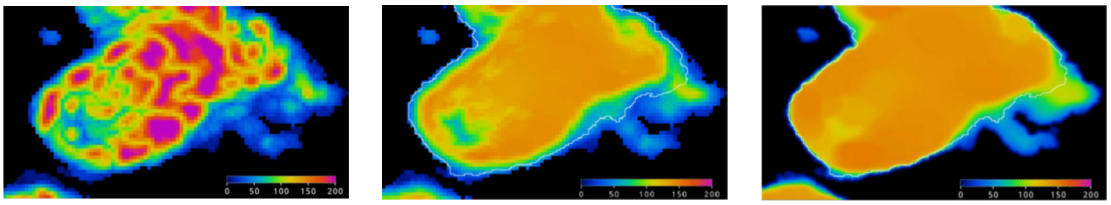


Figure 2.6: The original IRBT (left) and BTTD (right; Khlopenkov et al., 2021).



(a) no summation (b) summation (c) summation and refinement

Figure 2.7: Anvil rating of the white section marked in Figure 2.6 (Khlopenkov et al., 2021).

resulting in lower bin heights. Therefore, some extra refinements steps are included to overcome these artefacts (Figure 2.7c).

$$r_{anvil} = C_H \cdot \sum H_i \cdot i \cdot (2N + 8 - i) \tag{2.3}$$

The next step is to select OT candidates. OTs are localized cold areas with diameters of less than 15 km within an anvil cloud (K. Bedka et al., 2010). Therefore, the first action is to detect all cold spots followed by distance review. Firstly, the image is divided into 3x3 subsets and only the pixel with the highest BT-score is considered. Then, for each maximum the 11x11 pixel neighbourhood is scanned for other maxima and again only the highest value is retained. Then, the proximity check makes use of Equation 2.4. Where A and B are BT-scores of two OT candidates, D_{eff} is the effective distance, if the effective geometrical pixel distance is smaller than this D_{eff} then the smallest OT candidate will be discarded. Z is the Rectified Linear Unit (ReLU) activation function, which is 0 when its content is negative and equals the content if the content is positive. L is the minimal desired distance between two OTs (default 4 km). This value will be altered by the two ReLU activation functions. The first term increases D_{eff} if the BT-scores of the two OT candidates differs considerably, meaning that the weaker one can only

survive it is spatially further away from the stronger one. The second activation function only increases D_{eff} if at least one of the two candidates is weak, allowing for stronger candidates to be closer together.

$$D_{eff} = L \cdot \left(1 + Z\big(10 \cdot \sqrt{\frac{|A-B|}{A+B}}\big) + Z\big(\frac{17000 - min(A,B)}{170}\big)\right) \tag{2.4}$$

In the last step, the OT candidates are given a rating based on four parameters. Ideal OTs have high BT-scores, and are located in broad, cold and uniform anvils. The four parameters are (i) a tropopause factor, indicating how cold the OT candidate is compared to the tropopause, (ii) a prominence factor, which accounts for the temperature difference between the candidate and the surrounding anvil, (iii) an anvil area factor, which considers the anvil size around the OT and (iv) an anvil rating factor, taking into account the uniformity of the anvil. How each of these four factors is calculated is out of the scope of this work, but further detailed in Khlopenkov et al. (2021). Equation 2.5 shows how the probability is calculated for each of the OT candidates. $TropopauseF$ is the tropopause factor explained above, its value is between 0 and 1. λ contains the three other factors, which all have values between 0 and 1 as well. The motivation is that the tropopause factor is by far the most deciding factor but can be cancelled out if the candidate is not part of an anvil. If λ approaches 0 then $(\frac{1}{\lambda} - 1)$ will be large, since $TropopauseF$ has a value between 0 and 1, this will result in a small OT_{prob}. Contrarily, if λ approaches 1 then the exponent will approach 0 and the OT_{prob} will become close to 1.

$$OT_{prob} = TropopauseF^{0.6(\frac{1}{\lambda}-1)} \tag{2.5}$$

2.3 Forecasting Thunderstorms over Lake Victoria

This section introduces the current efforts in forecasting thunderstorms and early warning systems over Lake Victoria. This thesis complements and build further upon these efforts. Here we outline how these systems work and what their limitations are.

2.3.1 Forecasting Thunderstorms with NWP

NWPs are the physical models that are used to forecast weather patterns and are the basis of traditional weather forecasting applications. In this thesis, we do not apply a NWP system. Nonetheless, the following paragraph includes a brief description of these models to demonstrate the difficulties surrounding weather forecasting in the Lake Victoria region and explain why we take a different approach.

NWP is a forecasting method based on physical weather observations and computational modelling of fluid dynamics. The atmosphere is a physical fluid and can therefore be modelled via fluid dynamics from a start state (Holton, 1973; R. A. Brown, 1991). These models rely on meteorological observations to initialize the start state i.e., to build a 3D gridded representation of the current atmosphere (Haltiner & Williams, 1980; Kalnay, 2003). NWPs then attempt to mathematically model the evolution of numerous characteristics of the atmosphere by numerically solving a set of partial differential equations (Kalnay, 2003; Tribbia & Baumhefner, 2004; Bauer et al., 2015). These equations always include a number of primitive equations. First, Newton's second law (conservation of momentum; Equation 2.6), where dv/dt is the change in

three-dimensional speed of an air parcel over time, α the specific volume, $-\alpha\nabla p$ is the pressure gradient force, $\nabla\phi$ represents the geopotential (the potential of Earth's gravity field), F is frictional force and $-2\Omega v$ is the Coriolis force (Kalnay, 2003; Tribbia & Baumhefner, 2004; Coiffier, 2011). Second, the continuity equation (conservation of mass; Equation 2.7), where $\partial\rho/\partial t$ is the change of density over time, which is equal to the divergence ∇ of mass in/out ρv of the system (Kalnay, 2003). Third, the first law of thermodynamics (conservation of energy; Equation 2.8), which states that heat applied to an air parcel at rate Q will result in an increase in thermal energy $C_v T$ and/or produce expansion (work). This can be transformed to Equation 2.9, where the rate of change in specific entropy of a parcel s is ds/dt, which is equal to Q/T. θ represents $T(p_0/p)^{R/C_p}$, where p_0 is a reference pressure (1000 hPa) and $C_p = C_v + R$ (Kalnay, 2003; Tribbia & Baumhefner, 2004). Fourth, The equation for conservation of water mass (Equation 2.10), where E and C are Evaporation and Condensation respectively. Lastly, the equation of state for ideal gases (Equation 2.11) relates the atmosphere's density ρ and volume α with its temperature T and the gas constant R (Kalnay, 2003; Tribbia & Baumhefner, 2004; Coiffier, 2011).

$$\frac{dv}{dt} = -\alpha\nabla p - \nabla\phi + F - 2\Omega v \tag{2.6}$$

$$\frac{\partial\rho}{\partial t} = -\nabla\rho v \tag{2.7}$$

$$Q = C_v\frac{dT}{dt} + p\frac{d\alpha}{dt} \tag{2.8}$$

$$\frac{ds}{dt} = C_p\frac{1}{\theta}\frac{d\theta}{dt} = \frac{Q}{T} \tag{2.9}$$

$$\frac{dq}{dt} = E - C \tag{2.10}$$

$$\rho\alpha = RT \tag{2.11}$$

However, numerous small-scale and large-scale processes affect the atmosphere such as sea breezes, snow bands and thunderstorms (Figure 2.8; Tribbia & Baumhefner, 2004; Bauer et al., 2015). All these factors influence the state of the atmosphere and can have significant influence on the large-scale dynamics of the atmosphere through upscale growth (Tribbia & Baumhefner, 2004; Stensrud, 2009). NWPs models attempt to represent these processes via parameterizations, which are simplifications of mechanisms via either numeric parameters or simplified equations (Kalnay, 2003; Tribbia & Baumhefner, 2004). Parameterizations are necessary, in particular for processes for which no physical equation has been found that accurately represents the process. In addition, some of these small-scale mechanisms require NWPs to operate at high spatial and/or temporal resolutions, which can supersede the current computational capabilities (Haltiner & Williams, 1980; Coiffier, 2011).

Convection is of significant importance for forecasting thunderstorms. However, convection is a relatively small-scale (<1 km) process for which the hydrostatic equilibrium assumption is not longer valid, and is therefore parameterized in coarse-scale NWPs (Stensrud, 2009; Hanley et al., 2021). These models are generally unable to produce high-intensity precipitation events and overestimate light rain events (Stephens et al., 2010). High-resolution 1km NWPs, which include processes for convection, no longer rely on parameterization for representing convection. However, these models are still not able to fully represent convection as e.g., convective updraft in clouds, requires even higher resolutions (Bryan et al., 2003; Hanley et al., 2015; Hanley et al., 2021). Therefore, these NWPs are called convective-permitting, instead of convective-resolving.

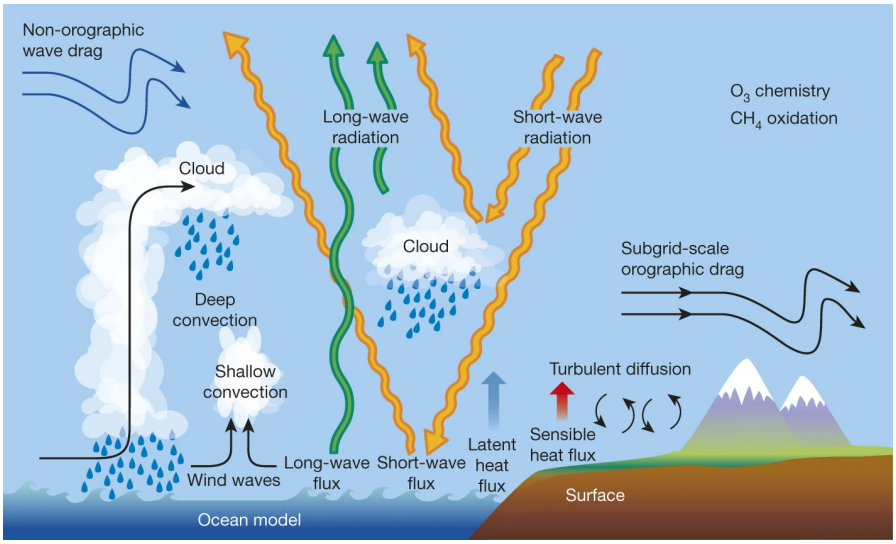


Figure 2.8: A number of atmospheric processes that are frequently parameterized (Bauer et al., 2015).

Since 2011, the UK Met Office runs a 4.4km resolution NWP for the East-Africa region (Chamberlain et al., 2014). The first version was the East-Africa 4.4km model, which was replaced in 2019 by the Tropical Africa 4.4km Model (TA4). Both these models are convective-permitting, which results in sub-optimal convective cells that are too large and circular (Hanley et al., 2015; Hanley et al., 2021). As a consequence, these models overestimate intense rain and underestimate light rain. TA4 is able to accurately predict afternoon precipitation in the region but has lower performance for nightly rains, when most boating accidents occur. Moreover, in the long rains, the model underestimates night-time storms over Lake Victoria and predicts them too late (Hanley et al., 2021).

Part of the sub-optimal performance of these NWPs is the due to the quality of the meteorological observations. The East-African weather observation network is insufficient for traditional high-resolution NWP weather modelling due to a combination of reasons (IOC et al., 2019). First, in some areas, there are no weather monitoring stations leading to gaps in coverage. Next, many stations do not report data internationally or do not report at the required hourly interval. In addition, the high cost of maintenance and too little trained operators complicate operating weather stations. Finally, instead of being automated, some stations require significant manual intervention (IOC et al., 2019).

Since 2019, the HIGHWAY project provides weather forecasts to the communities around the lake twice per day. These weather forecasts are derived from TA4 (Watkiss et al., 2020; Hanley et al., 2021). Additionally, HIGHWAY improved the weather observation infrastructure of the region to allow for better operation of NWPs and raised awareness to the communities around the lake of their weather service (Watkiss et al., 2020). TA4 is estimated to predict ~70% of the storms (Watkiss et al., 2020). However, TA4 has better performance in the afternoon than at night and underestimates night-time storms over Lake Victoria during the long rains. This indicates that the prediction of nightly thunderstorms over the lake is likely below ~70%, leaving ample room for improvement.

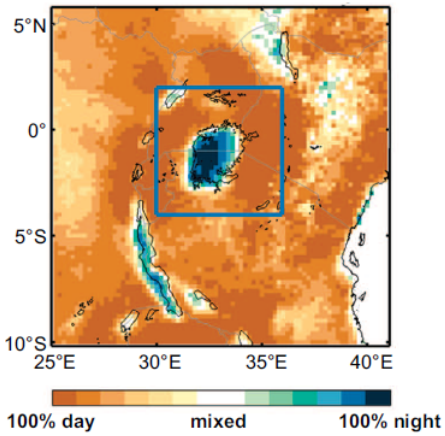


Figure 2.9: A visualization of the diurnal cycle of thunderstorms over Lake Victoria. The blue rectangle represents the window that was selected by Thiery et al. (2017) to count the daytime OTs (Adapted from Thiery et al., 2017).

2.3.2 Forecasting Thunderstorms with Satellite Observations

Another way to forecast heavy thunderstorms over Lake Victoria is to utilize the relation between afternoon en overnight storm activity, which can be measured by near real-time satellite imagery. The VIEWS project aims to model the probability of extreme thunderstorm nights as a function of afternoon conditions over the surrounding land. It performs a binomial regression on the total afternoon OT counts, derived from the 2010 OT dataset, to predict whether the subsequent night will be an extreme thunderstorm night (Thiery et al., 2017). Thiery et al. (2017) defines these extreme thunderstorm nights as nights where the total number of OTs over Lake Victoria during 00:00-–12:00 East African Time (EAT; 21:00—09:00 UTC) exceeds the 99th percentile, which equals 2236 OTs. The 99th percentile of most extreme thunderstorm nights corresponds to the 3–4 nights with most storms per year.

Thiery et al. (2017) models the relationship via Equation 2.12, where $P(ex)$ is the probability of an extreme night and. OT_{day} is the total number of OTs during the preceding day. This is defined as the sum of all OT pixels detected between 12:00 and 18:00 East African Time (EAT) (09:00–15:00 UTC) in an area enclosed by the rectangle in Figure 2.9. β_0 and β_1 are two parameters fit by logistic regression. β_0 is the intercept, which sets probability for an extreme thunderstorm night when no OTs were observed during the preceding day and β_1 is the regression coefficient.

$$ln(\frac{P(ex)}{1-P(ex)}) = \beta_0 + \beta_1 * OT_{day} \tag{2.12}$$

The VIEWS model predicts the probability of having a 99^{th} percentile thunderstorm-intense night (Thiery et al., 2017). The practical implementation requires a probability threshold (θ) to determine when an early warning should be issued. Once a θ is selected, warnings are automatically derived from this statistical model. The issued warnings will capture only a fraction of all actual extreme nights. This fraction is the hit rate (H), which will be anti-correlated to θ i.e., a lower probability threshold leads to a higher hit rate. Oppositely, a false alarm occurs

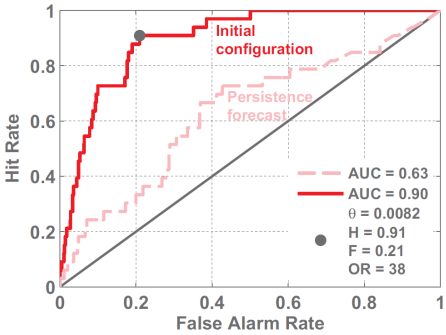


Figure 2.10: ROC curve of the non-optimized VIEWS model (Thiery et al., 2017). Area Under the Curve (AUC) is a performance metric, where random guessing has an AUC = 0.5, and a perfect model has an AUC = 1.

when a warning is issued when there is actually no extreme night occurring. The fraction of non-extreme events for which a warning is issued is the false alarm rate (F). F increases when θ is decreased. Thus, increasing H inevitably leads to an increase in F, and vice versa. Depending on the objective of the warning system, acceptable values of H and/or F can substantially vary. Therefore, it is not possible to determine one unique and most suitable threshold probability (Thiery et al., 2017). The Receiver Operating Characteristic (ROC) curve in Figure 2.10 visualizes H and F for different values of θ. ROC curves above the 1:1 line (in gray) indicate that the forecast performance supersedes random guessing, and the closer the ROC curve approaches the upper-left corner, the better the forecasts (Wilks, 2011; Thiery et al., 2017).

This model was optimized by considering various values for three parameters influencing OT_{day} i.e., (i) the forecast lead time λ (in h), (ii) the length of the daytime window μ (in h), and (iii) a critical rank correlation threshold r_{crit}, which controls the spatial window (Equation 2.13). λ was varied between 3h to 11h, the minimum lead time of 3h ensures that there would be sufficient warning transmission time in an practical system. Secondly, μ was varied from 2h to 14h and r_{crit} from 0.10 to 0.30 (Thiery et al., 2017). r_{crit} is the threshold for the Spearman rank correlation between time series of the daytime OT pixels and the time series of the nighttime OT pixels over Lake Victoria. For each combination of λ and μ a Spearman rank correlation coefficient will be calculated for each pixel. This coefficient indicates how strong each daytime pixel correlates with the number of total nighttime OTs. By selecting the pixels with a Spearman rank correlation coefficient above a certain threshold, only those pixels which contribute significantly to the thunderstorm dynamics are considered (Thiery et al., 2017). Area Under the Curve (AUC) was used as performance metric, where an AUC = 0.5 equals that of random guessing and AUC = 1 is a perfect model.

$$OT_{day} = \sum_{i=0}^{\mu} OT_{p,\lambda-\mu+i} \tag{2.13}$$

Figure 2.11 shows the results of these optimizations. An optimal lead time λ of 3h, aggregation time μ of 14h, r_{crit} of 0.15 and θ of 0.0129 result in a model with AUC = 0.93. This model has an H and F of 0.85 and 0.13, respectively. Lastly, the Odds Ratio (OR) is 36. OR is defined as the relative chance that a day with $P(ex) > \theta$ is followed by an extreme night compared to the chance a day with $P(ex) \leq \theta$ is followed by an extreme night i.e., a day with $P(ex) > 0.0129$ is

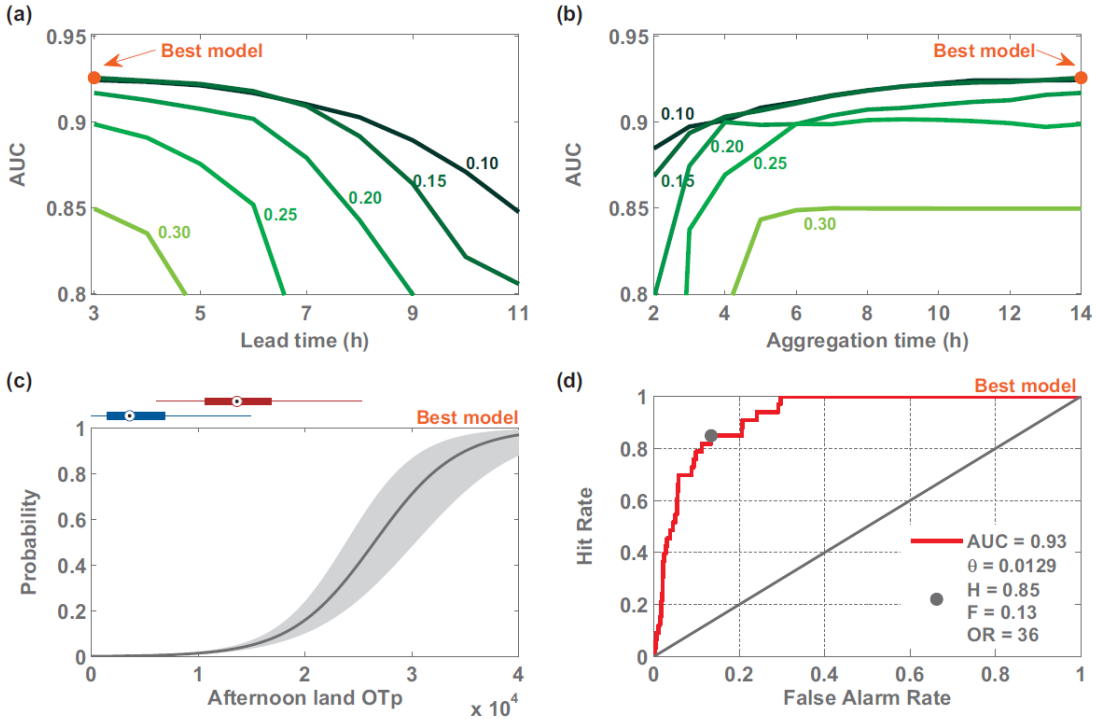


Figure 2.11: The optimized parameters of VIEWS (Thiery et al., 2017). a) and b) represent the performance of different lead times and aggregation times respectively, the green curves are different values of r_{crit}. c) The prbability of an extreme night, based on the preceding OT_{day} count, and d) the ROC curve of the optimal configuration.

36 times more likely to be followed by an extreme night than a day with $P(ex) \leq 0.0129$. Even though, the best model has an aggregation time of 14h, its performance is ~equal to the model with 10h aggregation time. AUC only increases marginally after the 10h mark.

The false alarm rate (F=0.13) is still considerably large in the optimal model, as a false alarm is issued ~once per week. It is possible to increase θ to obtain a lower hit rate, where H is 0.5 i.e., 50% of all extreme events are predicted. Then, a false alarm would occur ~once per month. However, false alarms generally capture intense storm nights rather than calm nights e.g., 30%, 50% and 70% of all false alarms correspond to nights above the 90th, 83th and 72th percentile, respectively (Figure 2.12). Thus, while false alarm nights are not in the 1% most intense thunderstorm nights, they nonetheless often are nights with strong thunderstorm activity. Overall this indicates that false alarms contain valuable information for a practical thunderstorm early warning system (Thiery et al., 2017).

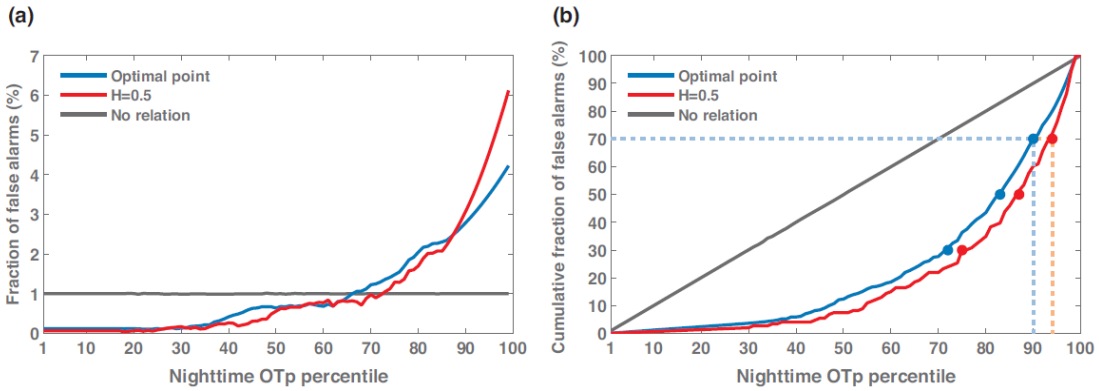


Figure 2.12: Characteristics of false alarms in VIEWS (Thiery et al., 2017).

2.4 Machine & Deep Learning

The following section gives an introduction to the general concepts of Machine Learning (ML) and Deep Learning (DL), along with their best practises and implications. The goal is to provide information on (i) what ML and DL are, (ii) what both terms exactly refer to, (iii) what kind of problems can be solved with these techniques, (iv) what the basic principles behind Neural Networks (NNs) are, and finally (v) what some of the good practises are in designing deep learning architectures.

2.4.1 Machine Learning

ML and its sub-field DL deal with the concept of letting algorithms learn from data and are part of the Artificial Intelligence (AI) field. In general, ML methods train a model based on sample data. There are three distinct approaches in ML i.e., (i) supervised learning, (ii) unsupervised learning, and (iii) reinforcement learning (Jordan & Mitchell, 2015). Each approach handles specific types of problems and has its own methods for obtaining solutions. However, blends of these three base types also exist. For example semi-supervised and discriminative learning combine different aspects of supervised and unsupervised learning (Jordan & Mitchell, 2015; Mohri et al., 2018).

The goal of supervised learning is to predict Y from X, where Y is called the label (or target) and X the feature set. Supervised learning requires labelled training data i.e., data consisting of feature-label pairs. All supervised learning models have at their core a mapping function, which translates X into Y (Hastie et al., 2009; Harrington, 2012; Jordan & Mitchell, 2015). Depending on the type of label data, different applications exist e.g., classification, regression and ranking (Mohri et al., 2018). Classification handles those problems where the desired output belongs to a distinct group of possible outcomes. One example is a simple spam classifier, where the feature data consists of emails and the labels are binary values indicating spam or no spam. This is called a binary classifier, multi-class classifiers also exist and are quite common (Harrington, 2012; Jordan & Mitchell, 2015; Mohri et al., 2018). Regression methods are used when the label is a continuous value e.g., the price of a house. In this example, a good feature set could include surface size, building year, EPC, number of bedrooms, etc. Ranking problems require a list of

objects to be ordered. A simplified example would be to rank web pages based on their relevance to the search query, popularity and update frequency (Hastie et al., 2009; Harrington, 2012; Jordan & Mitchell, 2015; Mohri et al., 2018).

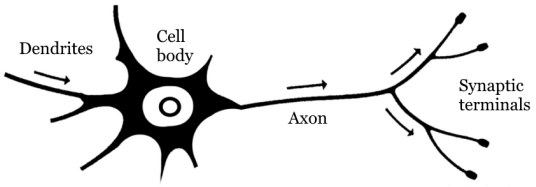
Supervised learning methods follow a common learning approach (Harrington, 2012; LeCun et al., 2015; Jiang et al., 2020). First, the available data is split into a training and test set, where the training set is generally much larger than the test set (e.g., 85-15%). Then, the relevant features need to be selected from the dataset. Subsequently, the model is fitted with the training data. Now the performance of the model can be measured by passing the test set through an error function (Harrington, 2012; LeCun et al., 2015; Akalin, 2020; Jiang et al., 2020; L. Wang et al., 2020). The performance is measured with a separate test set to prevent overfitting, a concept which is handled at the end this section.

Secondly, unsupervised learning operates on unlabelled training data, that is data with only features. Where supervised learning dealt with the task of predicting Y from X, unsupervised learning aims to extract specific information (properties) from the data e.g., which groups of data points are present or which dimensions hold the most information. Depending on the goal of the algorithm, different applications exist e.g., clustering, dimensionality reduction and anomaly detection (Harrington, 2012; Shalev-Shwartz & Ben-David, 2014; Mohri et al., 2018).

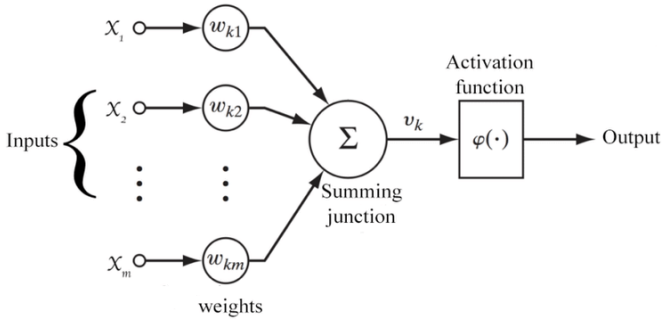
Lastly, reinforcement learning methods attempt to achieve a long-term goal by interacting with the environment, knowing that their current decisions affect the probabilities of future possible outcomes. Here, situations are mapped to actions i.e., in this *circumstance* what is the best *action* to undertake to maximize the chances of achieving the *goal*. Reinforcement learning has many uses in game theory, multi-agent systems and robotics (Szepesvári, 2010; Wiering & Van Otterlo, 2012; Sutton & Barto, 2018). Although it is a very exciting field of research, we do not employ reinforcement learning in this thesis.

2.4.2 Deep Learning

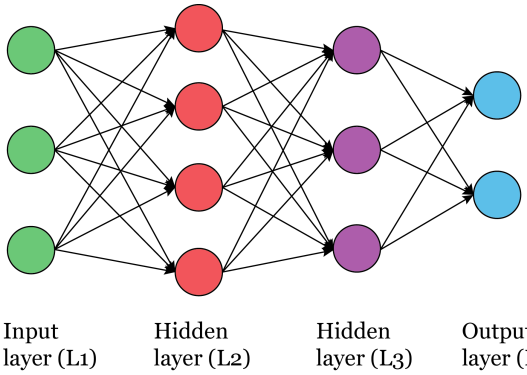
DL represents the subset of ML approaches that apply Artificial Neural Networks (ANNs) to tackle problems. Here, we will focus on ANNs for supervised learning. ANNs are often simply called NNs and originated from the idea to computationally mimic the brain, which consists of a network of densely-connected neurons (Figure 2.13; Maass, 1997; LeCun et al., 2015; Schmidhuber, 2015). In NNs, a neuron is represented as a simple mapping function (Figure 2.13b). Each neuron receives its input from a number of other neurons and stores a unique weight for each of these connections. The output values of the input neurons are multiplied by their weight, and are then summed. Then, an activation function is applied over this sum. The activation function allows for non-linear behaviour of the model. Usually, neurons are grouped together in layers and receive their input from all neurons in the previous layer, while their output serves as input for all the neurons in the next layer. The first layer in the network is data (model input), while the last layer is the output of the network (Maass, 1997; LeCun et al., 2015; Schmidhuber, 2015; Goodfellow et al., 2016). As opposed to ML, feature selection can be included in NNs i.e., instead of manually selecting and creating features, the model can learn which features in the data are most useful or extract higher level features autonomously. Secondly, the 'fitting' of the model is replaced by a training phase, which the following subsection explains in detail (Hastie et al., 2009; Harrington, 2012).



(a) Simplified schematic representation of a neurological neuron (Adapted from Akgün & Demir, 2018).



(b) Representation of a neuron in ANNs (Adapted from Akgün & Demir, 2018).



(c) Example of a NN (after Alapatt et al., 2020)

Figure 2.13: Example of a neuron, the representation of a neuron in NNs and a NN.

2.4.3 Training Deep Learning Models

Figure 2.14 shows an example of a simple NN architecture, where the bottom layer represents the input and the topmost layer the output. In-between these layers are two hidden layers of the network. The input layer of the network is a data sample, which can be anything that holds information e.g., numbers, the pixel values of an image or even text after it has been pre-processed (Lai et al., 2015; C. Zhou et al., 2015). The data is regarded as being the output of the neurons in the input layer. The values of the neurons in the first hidden layer are calculated in two steps. First, for each neuron the total weighted sum of its inputs is obtained via Equation 2.14a. Where z_j represents the weighted sum of inputs for neuron j in a layer L, i is the index of a neuron in layer $L-1$, which has n neurons. w_{ij} is the weight by which the output of neuron i is multiplied for neuron j and a_i is the activation value (output) of neuron i. In the second step, an activation function is applied over z_j to obtain the output of that neuron y_j to the next layer (Equation 2.14b; Hecht-Nielsen, 1992; Anthony & Bartlett, 2009; Hastie et al., 2009). In the next layer $(L+1)$, the same process takes place but the input comes from the neurons of layer L instead of the $L-1$. This mechanism is repeated for the neurons in the output layer, which produce the output of the network. In this example there are two neurons in the final layer, so two values will be predicted. Subsequently, the loss function will quantify the error/loss of this prediction compared to the actual label. The choice of loss function depends on the type of problem (Anthony & Bartlett, 2009; Shalev-Shwartz & Ben-David, 2014; Mohri et al., 2018). One of the most simple loss functions is Mean Squared Error (MSE), which is appropriate for regression problems.

$$z_j = \sum_{i=1}^{n} w_{ij} \cdot a_i \tag{2.14a}$$

$$a_j = f(z_j) \tag{2.14b}$$

The first derivative of a function corresponds to the slope of that function. For a multivariate function f, this slope is called gradient ∇ and is equal to the vector of partial derivatives of f with respect to each of the variables in f Equation 2.15. Thus, for a NN, the gradient of its loss function can be obtained by calculating the partial derivatives of the loss function with respect to the model parameters (weights). Altering the weights in the negative direction of their slope will reduce the value of the loss function. This process can be repeated until convergence of the loss function i.e., until the loss function reaches a (local) minimum. This step-wise minimization of the loss function is the concept behind the group of 'gradient descent' optimization algorithms and corresponds to the actual learning of the model (Anthony & Bartlett, 2009; Hastie et al., 2009; Ruder, 2016). The following paragraphs demonstrate how the partial derivatives of the loss function, with respect to each weight, can be obtained.

$$\nabla f(x_1, x_2, \ldots, x_n) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1, x_2, \ldots, x_n) \\ \frac{\partial f}{\partial x_2}(x_1, x_2, \ldots, x_n) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x_1, x_2, \ldots, x_n) \end{bmatrix} \tag{2.15}$$

Loss functions can be described as multivariate functions, which include all of the weights and

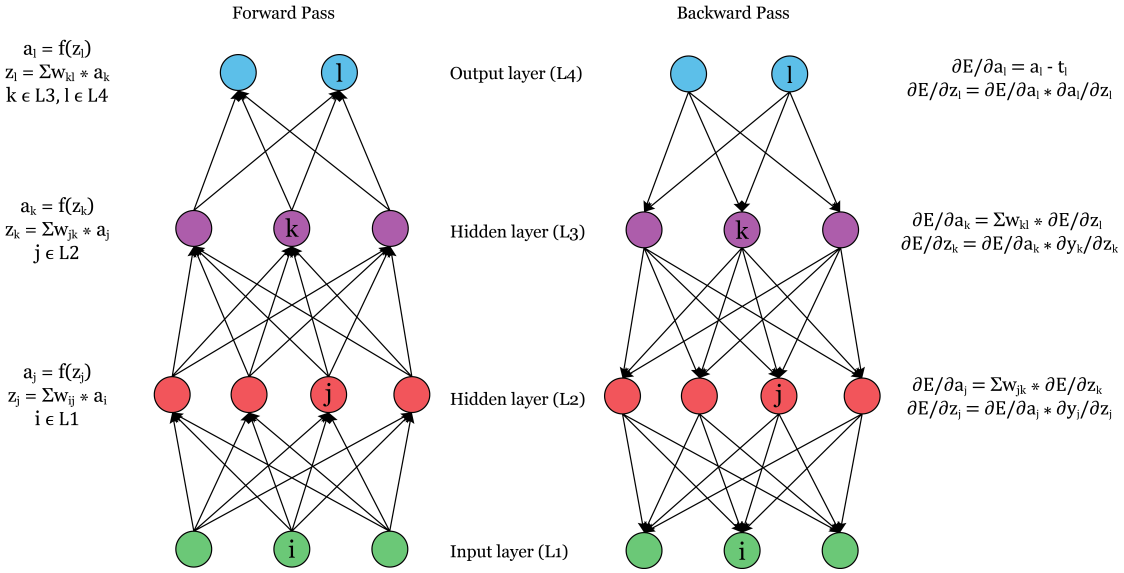


Figure 2.14: Forward pass and backward pass (backpropagation) in a neural network (after Sezer et al., 2020).

inputs of the NN they are scoring. For example, the architecture in Figure 2.14 has a loss equal to $\sum_{j=1}^{1}(a_j - y_j)^2$, where j is the index of a neuron in the output layer. This can be rewritten as the sum of the errors of the two output neurons i.e., $(a_1 - y_1)^2 + (a_2 - y_2)^2$. Applying Equations 2.14a and 2.14b results in $(f(\sum_{k=1}^{k_{max}} w_{k1} \cdot a_k) - y_1)^2 + f(\sum_{k=1}^{k_{max}} w_{k2} \cdot a_k) - y_2)^2$, which equals $(f(w_{11} \cdot a_1 + w_{21} \cdot a_2 + w_{31} \cdot a_3) - y_1)^2 + f(w_{12} \cdot a_1 + w_{22} \cdot a_2 + w_{32} \cdot a_3) - y_2)^2$. This process can be applied repeatedly for all activation values in the formula. Each iteration, moves one layer back to the front of the network and generates an expression for the loss function in terms of the activation values in that layer and the weights in all higher layers.

The chain rule (Equation 2.16) defines how the derivative of a variable y with respect to x can be calculated if y depends on u and u depends on x. By applying the chain rule twice, Equation 2.17 shows that to obtain the partial derivative for any weight, three components are required i.e., (i) how a weight affects the total input z_j of its neuron, (ii) how the input of that neuron affects its output a_j, and (iii) how the output of the neuron affects the loss function (Harrington, 2012; Shalev-Shwartz & Ben-David, 2014; Ruder, 2016). Equation 2.17b demonstrates that the first term, $\partial z_j/\partial w_{ij}$, equals a_j, the activation value of neuron j, which is in the layer before j. The middle term, $\partial a_j/\partial z_j$, depends on the activation function and can be quantified if the activation function is differentiable. Lastly, $\partial C/\partial a_j$ needs to be obtained. For neurons in the last layer, this is straightforward, Equation 2.18a demonstrates this with MSE as loss function. However, neurons in non-output layers require a different approach. In the previous paragraph, the loss function was rewritten in function of the activation values of a specific layer and the weights in all higher layers. Here, this method will be applied to expand the loss function to layer $L + 1$, where L is the layer of neuron a_j (Equation 2.18b). The derivative of this loss function with respect to a_j is solely dependent on the activation values (of the next layer). By applying the chain rule for partial derivatives (Equation 2.16b), the second step of Equation 2.18c is obtained, where a_k is the activation value of neuron k in layer $L + 1$. Applying the chain rule again on $\partial a_k/\partial a_j$ results in the third equation. This contains the derivative of z_k with respect to a_k,

which is equal to w_{jk}. The final equation in Equation 2.18c is recursive i.e., the derivative of the loss with respect to neuron a_j depends on the sum of the derivatives of all neurons in the next layer.

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} \tag{2.16a}$$

$$\frac{\partial y}{\partial x_i} = \frac{\partial y}{\partial (u_1, u_2, \ldots, u_n)}\frac{\partial (u_1, u_2, \ldots, u_n)}{\partial x_i} = \sum_{l=1}^{m}\frac{\partial y}{\partial u_l}\frac{\partial u_l}{\partial x_i} \tag{2.16b}$$

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial a_j}{\partial z_j}\frac{\partial C}{\partial a_j} \tag{2.17a}$$

$$\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial \sum_{l=0}^{n} w_{lj} \cdot a_l}{\partial w_{ij}} = \frac{\partial w_{ij} \cdot a_i}{\partial w_{ij}} = a_i \tag{2.17b}$$

$$\frac{\partial C}{\partial a_j} = \frac{\partial (a_j - y_j)^2}{\partial a_j} = 2(a_j - y) \tag{2.18a}$$

$$\frac{\partial C(a_j)}{\partial a_j} = \frac{\partial C(a_1, a_2, \ldots, a_m)}{\partial a_j} \qquad a_j \in L, \quad a_1, a_2, \ldots, a_m \in L+1 \tag{2.18b}$$

$$\frac{\partial C(a_1, a_2, \ldots, a_m)}{\partial a_j} = \sum_{k=1}^{m}\frac{\partial C}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_{k=1}^{m}\frac{\partial C}{\partial a_k}\frac{\partial a_k}{\partial z_k}\frac{\partial z_k}{\partial a_j} = \sum_{k=1}^{m}\frac{\partial C}{\partial a_k}\frac{\partial a_k}{\partial z_k}w_{jk} \tag{2.18c}$$

Equation 2.18 allows to calculate all partial derivatives step-wise, starting from the last layer (the output layer). The partial derivatives of the output layer L are obtained via Equation 2.18a. These partial derivatives are then used in Equation 2.18c to obtain the derivatives of the neurons in layer $L-1$, which can be used for layer $L-2$, and so on. This process is known as back-propagation (Hecht-Nielsen, 1992; Chauvin & Rumelhart, 2013). These partial derivatives are then often multiplied with a learning rate factor α ($0 < \alpha < 1$) or any other operation, depending on the optimization algorithm (R. A. Jacobs, 1988; Harrington, 2012; Zeiler, 2012; Mohri et al., 2018). Finally, the weights are updated with these correction factors. Summarized, in the training phase, data is first passed through the model in the forward step. As a result, the output neurons will produce a value (prediction), which the loss function will assign an error (loss) to based on the difference with the correct label. This loss will be sent back through the model from back to front, allowing to calculate the contribution of each weight to this error layer per layer. These partial derivatives correspond to the gradient of the loss function. By updating the weights opposite to this slope and with a step relative to the magnitude of the slope, the value of the loss function decreases, improving the performance of the NN.

2.4.4 Key Concepts in Deep Learning

The previous paragraphs handled the basic concepts behind NNs. In this section, the concepts of feature selection, hyperparameters, gradient decent, activation functions, overfitting, and batch normalization are discussed in more detail. Finally, two subtypes of NN are briefly described.

In DL, feature selection can be part of a NN's learning process and does not require human intervention (Shaheen et al., 2016; Dara & Tumma, 2018; Odi & Nguyen, 2018). This has

Figure 2.15: The effect of the size of the learning rate. θ are the model parameters (weights) and $J(\theta)$ is the loss function (Adapted from www.jeremyjordan.me).

a number of advantages and disadvantages. Firstly, it increases automation and significantly reduces human interaction. If enough training data is available, the features extracted by DL will be better than manually selected features. This propagates further into improved performance of the model (Xu et al., 2014; Zou et al., 2015; Semwal et al., 2017; Dara & Tumma, 2018). Furthermore, in many DL applications it is unclear what good, let alone the best features are, making manual selection difficult. However, by including feature extraction in the NN, the abstraction of the model increases, which makes the model even less human-interpretable. This can raise concerns in systems where DL algorithms are in charge of taking decisions that have significant impacts on individuals e.g., in court rulings or loan approvals. DL methods can be biased towards certain population groups, which of course leads to unwanted and unethical AI behaviour (Sandvig et al., 2015; Lee et al., 2018; Hutson, 2021). Secondly, as DL networks have more degrees of freedom, they also require larger training datasets, which causes issues when data availability is limited. Therefore, a coarse feature selection is often included during the pre-processing, which the NN then further refines.

Hyperparameters are adjustable parameters of the model, which cannot be learned. These hyperparameters can be 'tuned', i.e. adjusted 'manually' until an optimal or satisfactory configuration is obtained. Examples of hyperparameters are the number of hidden layers, the amount of neurons in each layer and the learning rate α. If the learning rate of a NN is too large or too small, a NN will not be able to learn properly Figure 2.15. In practical implementations, the learning rate will be decreased by an algorithm during training (Yu & Chen, 1997; Zeiler, 2012). Two common methods for doing this are (i) to progressively decrease it with a fixed, non-linear rate and (ii) to decrease it by a certain factor once the loss function hits a plateau.

In regular (batch) gradient descent, the gradients (partial derivatives) are accumulated in a matrix and the weight update is performed after an epoch, which represents one pass of all training samples through the network. This guarantees that the NN will move in the right direction at every update but requires many epochs before convergence is reached (Wilson & Martinez, 2003; Bottou, 2012). In stochastic gradient descent, updates are performed after each sample. This results in a model, which on average decreases the loss (Bottou, 2010; Ruder, 2016). Finally, mini-batch gradient descent combines batch gradient descent and stochastic gradient descent and performs updates after every 'mini-batch', where a mini-batch consists out of a fixed number of samples (Hinton et al., 2012). In practical applications, mini-batch gradient descent is used as it results in the fastest convergence (Hinton et al., 2012; Li et al., 2014).

Figure 2.16 depicts four of the most commonly used activation functions. The goal of these functions is to introduce non-linearity into the model (Elliott, 1993; Sharma & Sharma, 2017).

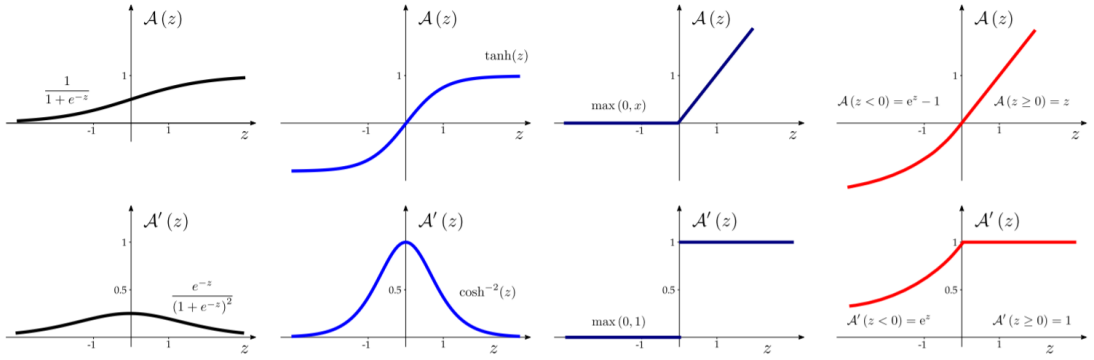


Figure 2.16: Visualization of a number of activation functions (top) and their gradients (bottom) that are frequently used in NNs. From left to right, the sigmoid, hyperbolic tangent (tanh), ReLU and Exponential Linear Unit (ELU) (Masi et al., 2021).

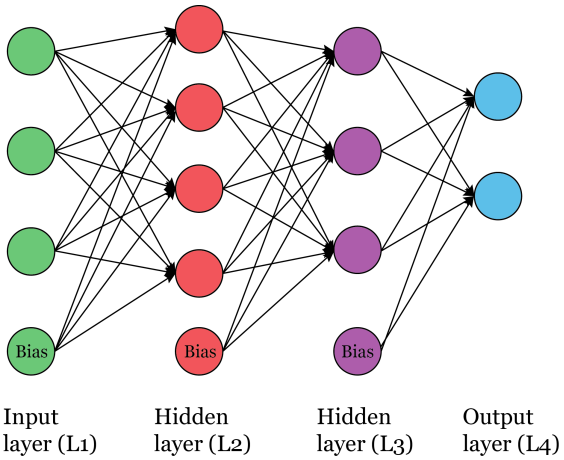


Figure 2.17: Neural Network with bias.

However, by solely using the interconnected neurons it is not possible to shift these functions up or down. Therefore, a bias neuron is generally included in each layer (Figure 2.17). Each neuron in the next layer considers this bias neuron as a regular neuron and has its own weight for it. The optimal value for these bias neurons is an additional learnable parameter of the network (Rumelhart et al., 1994).

Overfitting is one of the core concepts in DL. The goal of ML and DL is to make models that generalize the patterns in data. Overfitting is the action of fitting the model too closely to the training data, Figure 2.18 gives a visual example of overfitting (Tetko et al., 1995; Sarle et al., 1996). Overfitting will lead to low loss scores (high performance) on the training set but the performance on the test set (and subsequent practical implementation) will be significantly lower. There are a number of actions that can be undertaken to prevent or reduce overfitting in ML and DL. First, loss functions can be regularized. Regularized loss functions consist of a sum of two components i.e., the scoring function (e.g., MSE) and a regularization parameter. This regularization parameter will penalize the model based on its complexity (Adeli & Wu, 1998;

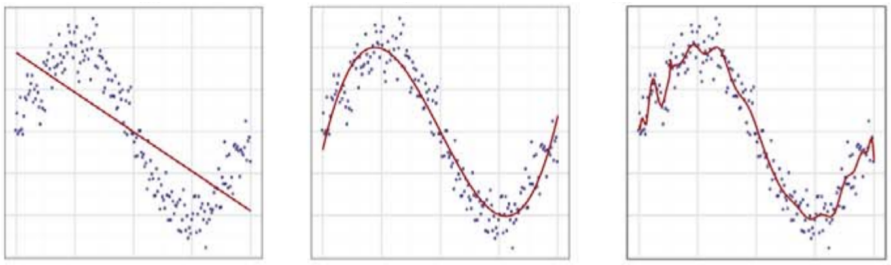


Figure 2.18: Example of (left) underfitting, (middle) good fitting and (right) overfitting (Adapted from Parpart et al., 2018)

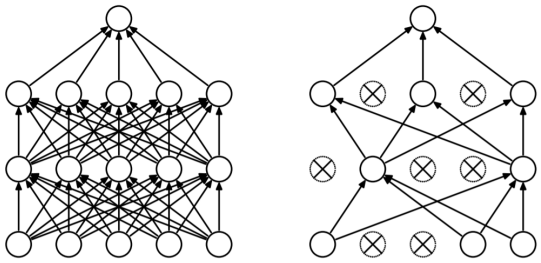


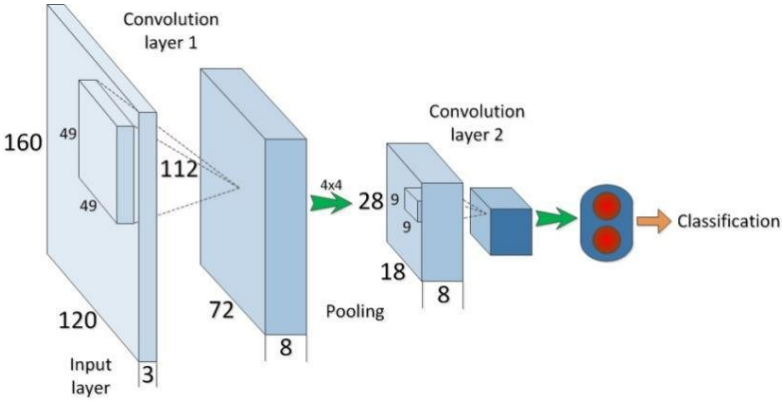
Figure 2.19: (left) a regular NN and (right) a NN with dropout (Srivastava et al., 2014).

Phaisangittisagul, 2016; Murugan & Durairaj, 2017). Often this is translated to either the L1 norm (regular sum) or L2 norm (sum of squares) of the weights in the network, multiplied by a factor λ. This λ is a hyperparameter of the NN, a too large λ will prevent the model from learning, as assigning a small weight will result in a large value for the loss function. Oppositely, a too small λ will not be able to regularize, and thus compress the total energy and reduce the complexity of the model well.

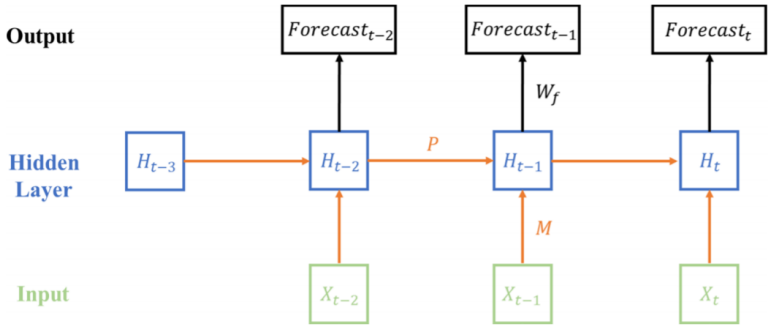
Second, instead of splitting the data into only two datasets (training and test set), often a third set is created. This validation set is then used to tune some of the hyperparameters and is not used for the actual learning. During the learning phase, the model's performance on the validation set can periodically be tested against the performance of the training set. If at a certain point, the performance of the training set keeps decreasing, while the performance of the validation set stagnates or increases, the training can be stopped early. This is called early-stopping and prevents extreme overfitting (Sarle et al., 1996; Prechelt, 1998; Caruana et al., 2001).

A third method to prevent overfitting is dropout, which can be applied to layers of a NN. Dropout randomly and temporarily removes a number of neurons from the designated layers Figure 2.19. This is equal to setting their output to 0. By introducing this randomness, a somewhat different version of the model will be trained in each training iteration. As a result, neurons cannot rely completely on the output of other neurons and become better regulated. Furthermore, the model becomes a sort of ensemble of numerous different versions, which encourages generalization (Srivastava et al., 2014).

The final concept, batch normalization, is used in NN to normalize the output of layers in the network. In the normalization process the mean of a layer's output is set to 0 and its standard deviation to 1. The output values of this layer are then multiplied by γ and summed with β, two



(a) Example of a CNN, pooling corresponds to grouping together pixels in the 2D channels. This is usually done by selecting the average or max value of that group, which is a 4 by 4 grid in this example (Y. Zhou et al., 2015).



(b) Example of an RNN (Sun et al., 2020).

Figure 2.20: Examples of a CNN and RNN.

learnable parameters for each layer. Normalization speeds up the rate by which the NN learns and combats overfitting. The reason why is still a topic of debate (Ioffe & Szegedy, 2015; Bjorck et al., 2018; Santurkar et al., 2018).

The type of NNs discussed here are dense (fully-connected) neural networks in which each layer is a 1D array of neurons, all connected to the neurons in the next layer. Alternatively, Convolutional Neural Network (CNN) layers perform N-dimensional convolutions on its input. In CNNs the input is not restricted to one dimension, which allows to retain and exploit more information from the input data, e.g. the spatial information in images. This input is multiplied by a number filters, in which each of these filters produces an output of a pre-defined dimension (Figure 2.20a; LeCun et al., 1989; LeCun, Bengio, et al., 1995). Recurrent Neural Networks (RNNs) are specialized for analysing time-series, where the output of previous time samples is fed into the input of the network (Figure 2.20b; Pineda, 1987; Sak et al., 2014).

Chapter 3

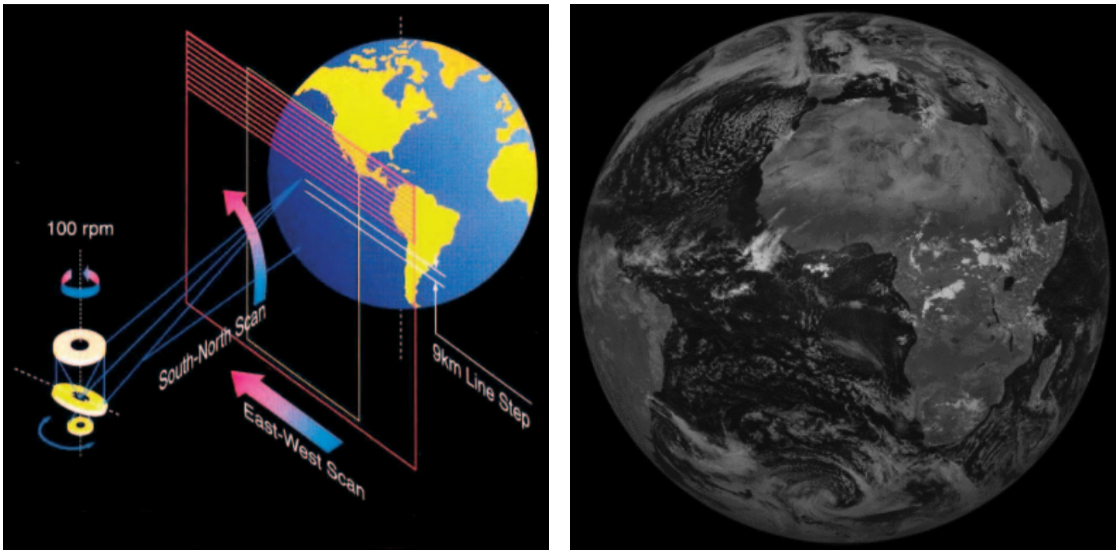
Materials & Methods

3.1 Materials

In this thesis, we apply statistical analysis, ML and DL on a dataset, which we derived from NASA's LaRC OT dataset. This OT dataset is obtained from SEVIRI satellite images via the algorithm described in Section 2.2.

3.1.1 The SEVIRI instrument

The SEVIRI instrument is a line by line scanning radiometer with a 50 cm diameter aperture, onboard of Meteosat Second Generation, a geostationary satellite. The spectral channels in which SEVIRI provide images, consist of four Visible and Near InfraRed (VNIR) bands, one of which is the High-Resolution Visible (HRV) channel, and eight IR bands (Schmid, 2000; Aminou, 2002). SEVIRI scans the Earth north-to-south line by line, where a line is east-west oriented and 9 km separated from the previous line. The system rotates around its axis at 100 rotations per minute (rpm) and the line of sight is moved for each rotation by a scanning mirror (Figure 3.1a). A complete disk image, which ranges from -81° to 81° latitude and -79° to 79° longitude, contains 1249 lines (Figure 3.1b). This image is acquired in ∼12.5 minutes, which is followed by a ∼2 minute waiting and re-calibration time to obtain an operational 15 minute repeat cycle (Schmid, 2000; Aminou, 2002). The channels have a 3 km spatial resolution in both directions, except for the HRV band, which is available at 1 km resolution. The SEVIRI product is publicly available through EUMETSAT from 19/01/2004 until present.



(a) Scanning principle of the SEVIRI instrument (Schmid, 2000).

(b) A VIS 0.8 band of the SEVIRI instrument showing the total scan range (www.eumetsat.int).

Figure 3.1: The scanning principle (a) and total scan range (b) of the SEVIRI instrument.

3.1.2 OT Dataset

The OT dataset was derived from the SEVIRI dataset by NASA's LaRC 2021 OT detection algorithm and spans from 01/01/2005 to 31/12/2015, as provided by NASA. The OT dataset has a quarter-hourly temporal resolution and contains 22 variables related to the OT derivation algorithm (see Section 2.2). Figure 3.2 depicts eight of these variables for 10/12/2015 at 14:00 UTC. The data is shown for a spatial window of 18°S to 12°N latitude and 22°E to 52°E longitude, and contains 28 pixels per degree latitude and longitude (840 by 840 in total), corresponding to a spatial resolution of ~4 km. The OT dataset contains one subdirectory for each day, and are numbered sequentially per year. Each subdirectory contains 96 OT data files corresponding to the 15-minute intervals of that day in netCDF4 (.nc) format. Network Common Data Format version 4 (netCDF4) is a file format system built on top of Hierarchical Data Format version 5 (HDF5), HDF5 allows to structure data in a hierarchical, scalable manner (Group, 2000; Folk et al., 2011). NetCDF4 extends HDF5 by offering extra utilities to array-oriented (scientific) data and is a common file format for satellite products (Rew et al., 2006; NASA, 2011; EUMETSAT, n.d.).

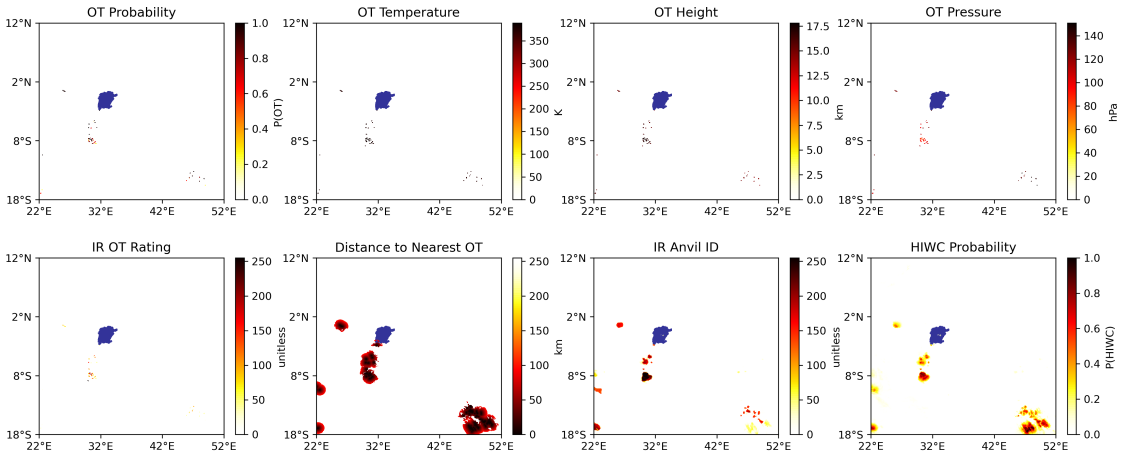


Figure 3.2: Eight variables from the OT dataset of 10/31/2015 at 14:00. The fields are, from top to bottom and left to right, (i) OT probability, (ii) OT peak potential temperature, (iii) OT peak height, (iv) OT peak pressure, (v) OT IR pattern recognition rating, (vi) distance to nearest overshooting cloud top detection pixel, (vii) anvil cloud detection id, and (viii) High Ice Water Content (HIWC) probability.

3.2 Methods

This section covers the implementation details of the experiments performed in this thesis. First, we describe the various preprocessing steps that are applied on the LaRC OT dataset to derive a functional dataset. Then, we report how the access to this dataset is optimized for different purposes, and how feature selection is performed based on correlation ranking. Fourth, the employed ML methods are covered, which is followed by the documentation of the DL experiments. The scripts of these implementations have been added as Appendices to this thesis and have also been made available on *GitHub*.

3.2.1 Dataset Preprocessing

Following Thiery et al. (2017), we define a data sample as a day-night tuple, where the night is the time period of 00:00–12:00 EAT (21:00–09:00 UTC). The preceding day is defined via the optimal configuration of Thiery et al. (2017) i.e., 07:00–21:00 EAT (04:00–18:00 UTC). There are 4017 days in the 01/01/2005–31/12/2015 time period, of which four days are missing in the OT dataset. The sample of the preceding day requires the first nine hours of these missing days. Therefore, eight days of the study period are not included in the final input dataset. For similar reasons the last day (31/12/2015) is not used. However, ~half of the remaining days are incomplete and ~180 files have a shape smaller than 840 by 840, which is inherent to the OT dataset received and likely due to inconsistencies of the processing algorithm. The malshaped files are removed and the missing files are duplicated from the time slot right before or after the missing file. If three or more consecutive files are missing, that day is excluded from the dataset. The final dataset contains 3288 day-night tuples, which is ~900 GB in size stored as netCDF4 files.

The storage and processing for this thesis is performed on Hydra, a High-Performance Comput-

ing (HPC) cluster available to VUB and ULB members, which has different types of Central Processing Unit (CPU) and Graphics Processing Unit (GPU) nodes. Training NNs requires many forward and backward passes of the dataset through the network. These passes are a repetition of the same, simple tasks performed at each neuron. This can easily be parallelized in a GPU or even a dedicated Tensor Processing Unit (TPU), which significantly reduces the time required to train these networks compared to CPUs (Y. E. Wang et al., 2019). Our dataset is too large to fit in the Random Access Memory (RAM) of a CPU or GPU node. This implies that during training of a NN, the samples will have to be fetched continuously from the secondary storage (disk). These read operations can become a bottleneck and slow the whole system down. Furthermore, we not only apply DL but also ML and regular statistical analysis, all of which require loading the dataset at least once. Therefore, we used two efficient data loading schemes.

3.2.2 Dataset Access

For the data access, we combine two approaches, depending on the size of the required subset, the frequency the subset will be used and the desired flexibility. If a subset is required for multiple analyses and is small enough to be stored in the main memory of a CPU/GPU node when decompressed, then the subset will be stored in a tensor and saved as a *.pt* file. This file can then be loaded any time it needs to be accessed, this method provides the most time-efficient access possible. However, when the subset does not fit in main memory when decompressed or when the subset needs to be flexible, for example when we want the possibility to change the variable(s) of interest (Figure 3.2), the temporal range or spatial range on the fly, we use *WebDataset*.

WebDataset is a Python package, offering efficient and scalable access to datasets, and is optimized for usage in combination with *PyTorch* (Aizman et al., 2019). *PyTorch* is an other Python library, which is built for ML and DL applications by Facebook's AI research lab (Paszke et al., 2017; Paszke et al., 2019). Although *PyTorch* only launched in 2017, it is widely used in many applications e.g, Tesla's autopilot, Lyft and Salesforce, and is adopted for the DL implementation in this thesis. Currently, efforts are undertaken to incorporate *WebDataset* in *PyTorch*. However, until this is completed, both libraries are available as separate packages (Aizman et al., 2020).

WebDataset requires data to be stored in Portable Operating System Interface (POSIX) tar archives. Tar archives allow for sequential data access, which is significantly faster than random access (A. Jacobs, 2009). *WebDataset* ensures and optimizes this sequential data-stream and connects it to DataLoader objects in *PyTorch*, which allows for efficient fetching of the dataset. Tar archives can be created from any file/directory via Command Line Interface (CLI) or via the ShardWriter class of *WebDataset*. However, as this tar archive serves as input to the DataLoaders, the OT dataset is split into day-night tuples before convertion. Appendix A contains the code that is created for this task.

The script is written on a per subdirectory basis to allow for parallel multi-processing in Hydra's multi-core CPU nodes. First, the presence of the next-day folder is verified, if the following day is present, the names of the .nc files belonging to the afternoon and night (including those of the next day folder) are stored in two separate lists. Then, the files are opened into a netCDF4 Dataset object to verify their shape, the small number of files (~180) that are not 840 by 840 are discarded. Then, missing files are replaced by the time frame right before or after them, if three consecutive files of the day-night tuple are missing, the tuple will be discarded. Otherwise, the files are concatenated and stored under two similar names, which is a requirement of *WebDataset*

to allow for efficient retrieval of the input-label tuples. Once the day-night tuples are created, the tar archive can be composed. The script in Appendix B shows how the tuples are converted to bytes, stored as 'afternoon' and 'night' and are then written to tar shards. A tar shard is a piece of the tar archive. Here, each tar shard contains ten data samples, corresponding to ten day-night tuples.

The resulting dataset allows *WebDataset* to efficiently load any variable or combination of variables from the dataset. Throughout this thesis, the *OT Probability* variable is used most frequent. Loading the entire dataset every time to only extract this variable is inefficient. Therefore, Appendix C iterates through the tuples generated by Appendix A and extracts the *OT Probability* variable via *nctoolkit*, a library for editing netCDF4 files. This results in an *OT Probability* dataset of only ~1 GB, because of the efficient netCDF4 compression.

3.2.3 Correlation-based Feature Selection

Here, we investigate the correlation between the daytime OTs and the subsequent nighttime OTs. Before analysing the nighttime OTs over Lake Victoria, a mask of the lake is required. This is constructed based on the *Visible Reflectance Image* variable of the netCDF4 OT file from 02/01/2005 at 12:45 (Figure 3.3). First, the image is cropped around Lake Victoria and subsequently thresholded. The resulting binary image is eroded once to remove small pixels cluster that are selected over land. Then, a fourfold dilation is performed to ensure the whole lake (and the coast) is included in the mask. Finally, this binary crop is placed in an empty matrix with the same size as the original image (840 by 840) to obtain the actual mask. This mask will be applied throughout this thesis on the nighttime OT maps to indicate which pixels belong to Lake Victoria. Furthermore, a subset for the label was generated based on this mask, which sums, for each night, the *OT Probability* over all lake-pixels and all time slices. This yields a tensor of shape (3288, 1), which is saved in a .pt file.

To analyse this statistical dependency, a rank correlation coefficient is calculated. For non-normally distributed datasets, the non-parametrical Spearman rank correlation coefficient is preferred (Sedgwick, 2014; Rebekić et al., 2015). Spearman's rank correlation coefficient (Equation 3.1) ranks the relationship between two datasets, giving scores between -1 and 1, where 0 indicates no correlation, -1 perfect anti-correlation and 1 represents perfect correlation. In Equation 3.1, r_s represents the Spearman correlation coefficient, $rg(X_i)$ and $rg(Y_i)$ are the two ranks of each observation i and n is the amount of observations.

$$r_s = 1 - \frac{6 \sum (rg(X_i) - rg(Y_i))^2}{n(n^2 - 1)} \tag{3.1}$$

The comparison of summed daytime OTs and the summed nighttime OTs includes 3288 observations (valid day-night tuples), which result in one Spearman rank correlation coefficient. In addition, we sum the daytime OT probabilities once over the spatial domain into a 3288 by 56 temporal tensor, and once over the temporal domain to obtain a 3288 by 840^2 tensor. Then, we apply the Spearman rank correlation algorithm to the temporal distribution, which yields 56 Spearman rank correlation coefficients, one for each time interval of the daytime period, and to the spatial distribution of OTs, obtaining 840 by 840 Spearman rank correlation coefficients. To efficiently compute the Spearman rank coefficients, Scipy's *spearmanr* function is used, which calculates the Spearman rank correlation between two equal-length series along with the p-value for the hypothesis that both series are uncorrelated (Virtanen et al., 2020). The spatial and

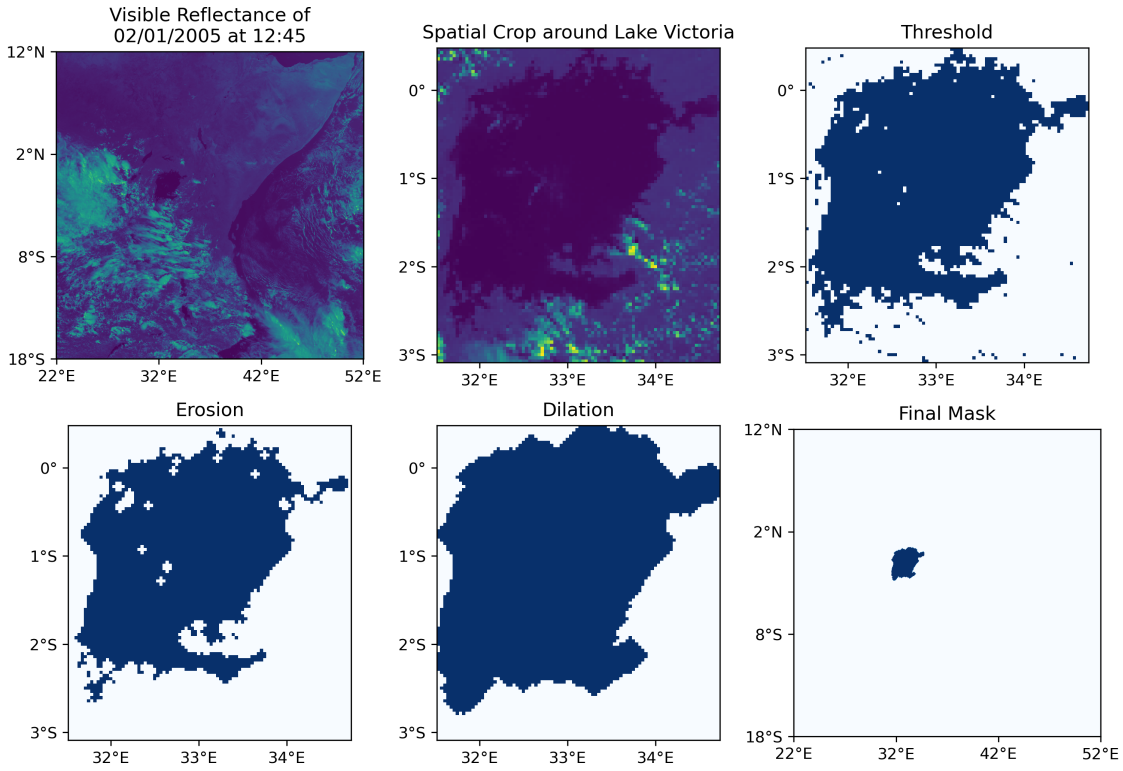


Figure 3.3: The generation of the mask for Lake Victoria based on the *Visible Reflectance* band of the OT file from 02/01/2005 at 12:45. The x-axis is the longitude, while the y-axis represents latitude.

temporal Spearman coefficients were subsequently used to perform a selection of the most correlated pixels and time periods. This selection is used in the ML and DL experiments to reduce the amount of redundant input information and parameters for each model. Furthermore, two subsets are generated: (i) the temporal subset, which contains one value, the sum of all highly-correlated ($r_s > 0.15$) pixels, for each highly-correlated time slice, and (ii) the spatial subset, which contains one value, the sum of all time slices, for each highly-correlated pixel. These subsets are stored as tensors in a .pt file.

3.2.4 Machine Learning Analysis

Similar to Thiery et al. (2017), we apply binary logistic regression to predict whether a 1% extreme night will occur based on the afternoon OT data. First, we summed the nighttime OT probabilities over Lake Victoria for each sample and ranked them to obtain the 1% extreme threshold, which is 5379 OTs. Nights with more than 5379 OTs are classified as 1, the others as 0. To perform the logistic regression, *scikit-learn*'s *LogisticRegressionCV* classifier is used (Pedregosa et al., 2011). The *LogisticRegressionCV* classifier implements logistic regression with a built-in Cross-Validation (CV) to to automatically determine the optimal hyper-parameters for each classifier. *LogisticRegressionCV* allows to manually specify a large number of classifier parameters such as loss function, the maximum number of iterations and the class weights. In our case, a balanced classifier requires a weight of 1 for the 0-class and a weight of 99 for the 1-class (1, 99). Here, we evaluated 75 different weight configurations ranging from (1, 50) to (1, 200) with a step-size of 2. For each configuration, 1000 models are trained with a training set of 80% of the samples, and tested with a test set of 20%. For each model, the dataset is divided into a different 80% training and 20% testing configuration.

This set-up was applied nine-fold, each time on a different set of input variables. First, the method of Thiery et al. (2017) is followed, which sums all daytime pixels in the 4°S–2°N, 30°E–36°E domain with an r_s above 0.15 for the whole daytime (04:00–18:00 EAT) into one OT count per afternoon and employ logistic regression with this one variable as input. Next, we perform the same analysis but considering the highly-correlated spatial and temporal input features in the whole domain (18°S–12°N, 22°E–52°E; Section 3.2.3). Then, the temporal selection of OT counts, described in Section 3.2.3), is summed into two, three, four and six aggregates. These aggregates are formed based on equally-sized splits e.g., for the temporal two-variable aggregate, the 09:00–18:00 time period is split into 09:00–13:30 and 13:30–18:00, which are each summed to obtain the two variables. Each of these four aggregate designs is used as input dataset for the *LogisticRegressionCV* classifier. Similarly, the spatial subset was grouped into two, three and four aggregates, which are used as input features for separate *LogisticRegressionCV* classifiers.

3.2.5 Deep Learning Analysis

In this thesis, DL is used for two goals. First, we attempt to improve the performance of the binary prediction in VIEWS by introducing dense neural networks. Secondly, we evaluate the possibility of extending VIEWS with spatially explicit predictions for nighttime thunderstorms over Lake Victoria based on CNNs.

Framework

Both DL tasks are carried out via the same modular approach. Here, we use *PyTorch Lightning* as framework of preference. *PyTorch Lightning* is an open-source Python library on top of *PyTorch*, which facilitates DL projects by offering a structured approach to building DL applications (Falcon, 2019). *PyTorch Lightning* projects consist out of (i) a dataset handler, which creates the training, validation and test subsets and handles their access, (ii) a classifier, which specifies what actions should happen during training, validation and testing and how models should be optimised, (iii) a (NN) model which will be trained, and (iv) a trainer, which controls communication between the dataset handler, classifier and model. The data handler, classifier and model are each defined in a separate file (Appendices D-F), while the trainer is invoked in a file where all customisable parameters are organised (Appendix F).

Data Handler

Two classes were created for handling access to the dataset, depending on whether the dataset fits in memory in Tensor format. The *TensorDataset* class handles datasets that fits in the RAM of a node, while *NetcdfDataset* handles the datasets that are too large. Both classes have three methods i.e., *training_dataloader*, *val_dataloader* and *test_dataloader*, which return *DataLoader* objects for training, validating and testing respectively. *DataLoaders* are iterables over a dataset and are the common method for looping through data in *PyTorch*. The constructor of the *TensorDataset* expects two tensors as input (*input_tensor* and *label_tensor*), where the first dimension should represent the different samples. It will divide this tensor based on its indices into training, validation and test sets. The constructor *NetcdfDataset* class requires a so-called *bucket* i.e., a list containing the locations of the .tar shards. These shards are then assigned for training, validation or testing. Furthermore, the *NetcdfDataset* class employs the *WebDataset* version of *PyTorch*'s *Dataset* and *DataLoader*, along with a function *decode_netcdf*, which transforms the byte stream tuples in each shard into two netCDF4 Dataset objects. Appendix D contains the code for these classes, along with a number of classes that can be used as pre-processing transforms.

Classifier

The *Classifier* class defines what should happen before/during/after each batch and epoch for the training, validation and testing of models and defines how the models should be optimized, it inherits from *PyTorch Lightning*'s *LightningModule*. *PyTorch Lightning* offers a structured but flexible approach on how to handle these actions. The constructor requires a loss function and a (NN) model to be passed to create an instance of the *Classifier* class (Appendix E). This model should have a *forward* method, which defines how data passes through it. Defining the training, validation and test loops follows an identical syntax. Methods that are called *X_step*, where X is training, validation or test, define what action should be undertaken for each (mini-)batch. *X_epoch_end* methods define what should happen at the end of every epoch. Here, only *X_step* and *X_epoch_end* methods are defined but *PyTorch Lightning* offers to define more possibilities such as *X_step_end* and *X_epoch_start*.

At every training step (*training_step*), the input of a batch will be passed through the model via its forward method, resulting in a predicted output. This output will be passed, along with the actual label, to the loss function. The loss function will return a score, which will be logged by the system. At every step in the validation loop, the forward method of the model will also be called, however, the loss will only be logged at the end of the epoch. This results in one mean loss for the entire validation set at every epoch. Training will stop once a predefined number of training epochs has passed. The operations for the test loop are identical to those of the

validation loop.

The last method, *configure_optimizers*, implements an optimizer and a scheduler. Optimizers define how the update values for the model's weights should be calculated. In practise, non-optimized gradient descent converges relatively slowly i.e., it requires many epochs to reach convergence. Therefore, optimizers have been built, which speed up the learning process significantly. The scheduler regulates when and how the learning rate should decrease during training.

NN models

The models (NNs) are handled by a separate CNN class, which constructs and manages dense NNs and CNNs (Appendix F). The general network architecture consists out of n convolutional layers, followed by j linear layers. The class offers flexibility and allows to easily build and alter CNNs or regular NNs, while preventing many of the common user errors. The constructor requires two parameters defining the shape of the data and output i.e., *input_size* and *output_size*, and accepts a large number of optional parameters. Each optional parameter can be defined as a single value or list/tuple. The optional parameters describe the characteristics of the 2d and 1d layers in the model e.g., layer size, kernel size, activation function, etc. If a tuple or list is given then its length must match the length of the *size_2d* or *size_1d* parameter, depending on whether the parameter applies to 1d or 2d operations. For each two-dimensional convolution operation, an output size (*size_2d*) should be defined, which indicates the number of channels in the next layer. The kernel size (*kernel_2d*), *stride_2d*, padding, groups and dilation can also be defined. *nn.Conv2d* automates the 2d convolution of an input layer of size *input_size* to the next convolution layer of size *output_size*. The *groups* parameter is an integer, which defines how the channels of the input layer should be connected to each channel in the output layer. At *groups* = 1, each input channel is connected to all output channels, while the maximum *groups* value equals *input_size* (the amount of input channels) and ensures that each input channel is only connected to *output_size* over *input_size* output channels.

The model allows to further specify whether a batch normalization (see Section 2.4) should occur after each convolution. This is followed by an activation function and the possibility to perform a pooling operation. A 2D pooling operation reduces the height and width of the channels, by combining (taking the maximum or average) several pixels together. The type of pooling (pool_2d), size of the window (pool_size_2d), stride (pool_2d_stride) and padding (pool_2d_pad) can be defined manually but have the identical default values as *PyTorch*'s *nn.MaxPool2d* function. The size of the linear layers can be defined via the *size_1d* parameter. The first linear layer has an input size equal to the output size of the last convolutional layer, this is managed automatically by the model. Each linear layer can be succeeded by a batch normalization specified in the *batchnorm_1d* parameter, which is followed by an activation function.

During the initialisation of a CNN object, the parameters will first be verified by the *validate_input_parameters*, which validates the type and length of each input parameter. Then, the 2d convolutional layers will be constructed in the *build_2d_layers* method, and the 1d linear layers in the *build_1d_layers* method. Lastly, the *forward* method defines what sequence of actions should happen in a forward pass i.e., pass the data through the 2d convolution layers, flatten the output, pass it through the 1d linear layers and return the output in a shape of (-1, *output_size*). *output_size* is the *output_size* value passed to the constructor and -1 is a 'free' value, which will be equal to the batch size.

Trainer

The *Trainer* combines the data handler, classifier and CNN in one implementation. At the top of this small script, a number of system variables are defined. These variables indicate where the logs

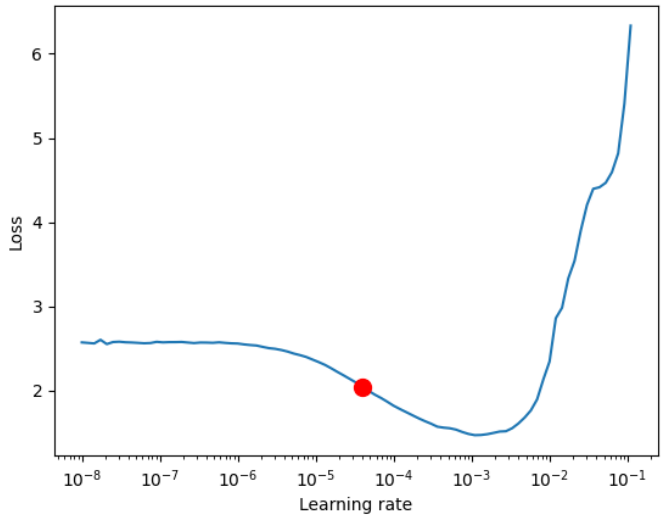


Figure 3.4: The empirically determined learning rate, which *PyTorch Lightning* uses to choose an appropriate starting value (Adapted from wwww.pytorch-lightning.readthedocs.io). The algorithm does not select the value with the lowest loss but deliberately chooses a value at the middle of the sharpest downward slope.

and models should be stored, where the input data is located, what the tunable hyperparameters of the model are (e.g., number of epochs and batch size) and what pre-processing steps should be performed. In the *CNN_PARAMS* dictionary, the NN/CNN parameters of choice can be defined (Appendix G). When this script is executed, a data handler and CNN will be initialised based on the defined parameters. The CNN will be passed as *model* for the initialisation of a *Classifier* object. The interaction of the *Classifier* with the data handler is regulated by *PyTorch Lightning*'s Trainer class. Furthermore, *PyTorch Lightning* has a time-saving automatic learning rate finder, which empirically determines at what learning rate training should begin (Figure 3.4). Here, the automatic learning rate finder is enabled viathe *auto_lr_find* parameter of the Trainer.

Training starts by calling the *fit* method of the Trainer object with the instances of *Classifier* and the data handler as its arguments. The Trainer ensures that the methods defined for the training, validation and test loop in *Classifier* are called at the right moment and that weights are only updated during evaluation of the *training_step* method. After training, the performance of the model is obtained via *test* method of the Trainer. Lastly, the model is given a name based on the current date and time and is then saved via the *torch.save* function.

Binary Classification

To perform the binary classification a number of ensemble models are created, both on the temporal as the spatial subset. Via an empirical analysis it is determined that NNs with two to five hidden layers containing 100 to 1000 neurons each yield optimal performance on the temporal subset, while four to six hidden layers, with 100 to 1000 neurons are best for the spatial subset.

The performance measure of the NNs is severely influenced by the small size of the test set.

There are only 32 intense events (class 1) in the entire dataset, and the test set comprises only 15% of that i.e., ∼4-5 heavy storm nights. This causes high variance in the performance of the NNs, which can give unrealistic performance estimates e.g., 100% accuracy for the extreme class when 4 out of 4 extreme events are classified correctly. Therefore, we train a high number of NNs and average their performance. Furthermore, we semi-randomly construct these NNs to diversify them and include them in ensemble models, which can boost their performance.

On the temporal subset, 20 ensembles are trained, where each ensemble consists out of 50 different dense neural networks. Each model in an ensemble is trained on the same training set, but the ensembles are trained on different training sets. Each NN is generated semi-randomly via the CNN class. The 2d parameters are left blank, so that only 1d layers are created. The sigmoid is selected as activation function, dropout is set to 0.4 and batch normalization is enabled. Then, the number of hidden layers and their size is determined by a random number generator for each model, with the number of layers between two and five and their size from 100 to 1000. Similarly, for the spatial subset 10 ensembles are trained, each consisting of 25 dense neural networks, their characteristics are identical, except for the number of layers, which is generated between four and six. Without preprocessing the spatial NNs start with 34219 input parameters (the number of highly-correlated pixels), this high number leads to a large amount of weights and subsequently longer training periods. Therefore, the 840 by 840 input maps are pooled by a 32 by 32 kernel before applying the correlation mask, which is pooled with the same settings. This yields a 1D tensor of size 124 that serves as input to the spatial NNs.

The temporal and spatial models are classified with *PyTorch*'s *BCEWithLogitsLoss* function, which calculates the binary cross-entropy between the predicted output and the label. Furthermore, *BCEWithLogitsLoss* allows to specify a weight for balancing the two classes, the weight should be the ratio of the number of negative (class 0) samples divided by the number of positive (class 1) samples, which equals 99 in our case. Lastly, the *TensorDataset* is used for both types of NNs as the temporal subset and spatial subset are both small enough to fit in main memory as tensors.

Spatially Explicit Predictions

Lastly, we want to evaluate the possibility of extending VIEWS with spatial predictions. We empirically analyse a number of CNNs for this purpose. First, it is important to consider the input which will be fed to the model and what label we want to predict. Until now, we made use of the *OT probability* variable from the OT dataset. However, this variable is constrained spatially i.e., the OTs usually only cover one pixel and thus share no information with their neighbours. This provides an unrealistic basis for a storm forecasting system i.e., if an OT forms over a pixel then in reality a radius of surrounding pixels (which have a 4 km spatial resolution) will experience severe weather phenomena. Thus, while the *OT probability* variable is an adequate proxy for analysing aggregated thunderstorm intensity, it does not provide an optimal basis for spatially forecasting severe weather. Therefore, we consider the *HIWC probability* variable available in the OT dataset (Figure 3.2). High Ice Water Content (HIWC) occurs in deep-convective cells and is an appropriate proxy for severe weather (Federer & Waldvogel, 1975; Kahn et al., 2018; Yost et al., 2018; Ratvasky et al., 2019).

Secondly, the size of the dataset, 3288 samples, is limited. Therefore, it is necessary to constrain the size of the samples fed to the CNNs. One effective way is to reduce the dimensionality of the data, which can be performed temporally and/or spatially by pooling operations. These pooling operations are also applied on the label to reduce the spatial resolution and provide a more

stable prediction target for the models. Moreover, the small dataset requires significant efforts to prevent overfitting, this is accomplished via dropout, batch normalization and regularization. Lastly, as opposed to the previous experiments, this task does not handle classification but regressions as we want to predict a continuous value for each pixel. Therefore, different loss functions are considered such as MSE.

Chapter 4

Results

4.1 Correlation-based Feature Selection

The goal of this section is to analyse where and when the statistical dependency between the daytime OTs and subsequent nighttime OTs is the most significant. Here, we make a temporal and spatial feature selection on the dataset based on Spearman's rank correlation for each time period and pixel in the domain.

The intensity of each type event decreases exponentially with its frequency of occurrence (Figure 4.1) e.g., the difference between a 1% (5379 OTs) and 10% (2774 OTs) is similar to the difference between the 10% and 100% events. In other words, a 1-in-100-day event is 10 times stronger than a 1-in-10-day event. Furthermore, the reported OT counts in Figure 4.1 for the afternoon greatly exceed the nighttime OT counts. However, it is important to realise that the nighttime OT counts are only aggregated over the ~15000 pixels that constitute Lake Victoria, while the afternoon OTs are summed up over the entire 840 by 840 domain.

The spatial and temporal sum of the daytime (04:00 to 18:00 EAT) OT probabilities in the 840 by 840 study domain has a Spearman rank correlation coefficient of 0.42 with the summed OT probabilities of pixels over Lake Victoria in the consecutive night (21:00 to 09:00 EAT). The p-value of $1.60e^{-143}$ indicates that both series are significantly correlated. This confirms the partial dependency of nighttime OT activity on the preceding afternoon OT intensity, described in Thiery et al. (2017) and Section 2.1.

Next, we perform a temporal and spatial selection on the daytime OT probability maps. First, all pixels in the study domain of the daytime OT probability maps are summed for each time period (04:00 to 18:00 EAT with quarter-hourly intervals), resulting in a tensor of shape 3288 by 56. For each of these 56 time intervals, the Spearman rank correlation gives the correlation with the total summed nighttime OT probability over Lake Victoria (Figure 4.2a). Similarly, for each pixel in the domain, its 56 values for the time intervals in the daytime period are summed, which results in a tensor of shape (3288, 840, 840). Then, Spearman's rank correlation between each pixel and the total nighttime OT count over Lake Victoria is calculated (Figure 4.2b). Temporal correlation, which is calculated for each 15 minute time interval, peaks between 09:00 and 18:00 EAT and has a p-value of 0 for each time interval. Spatial correlation is the highest over the pixels bordering the east side of the lake, its p-value is highly location-dependent (Figure 4.2c). Therefore, the temporal range is adjusted to 09:00-18:00 EAT and a spatial selection of pixels is

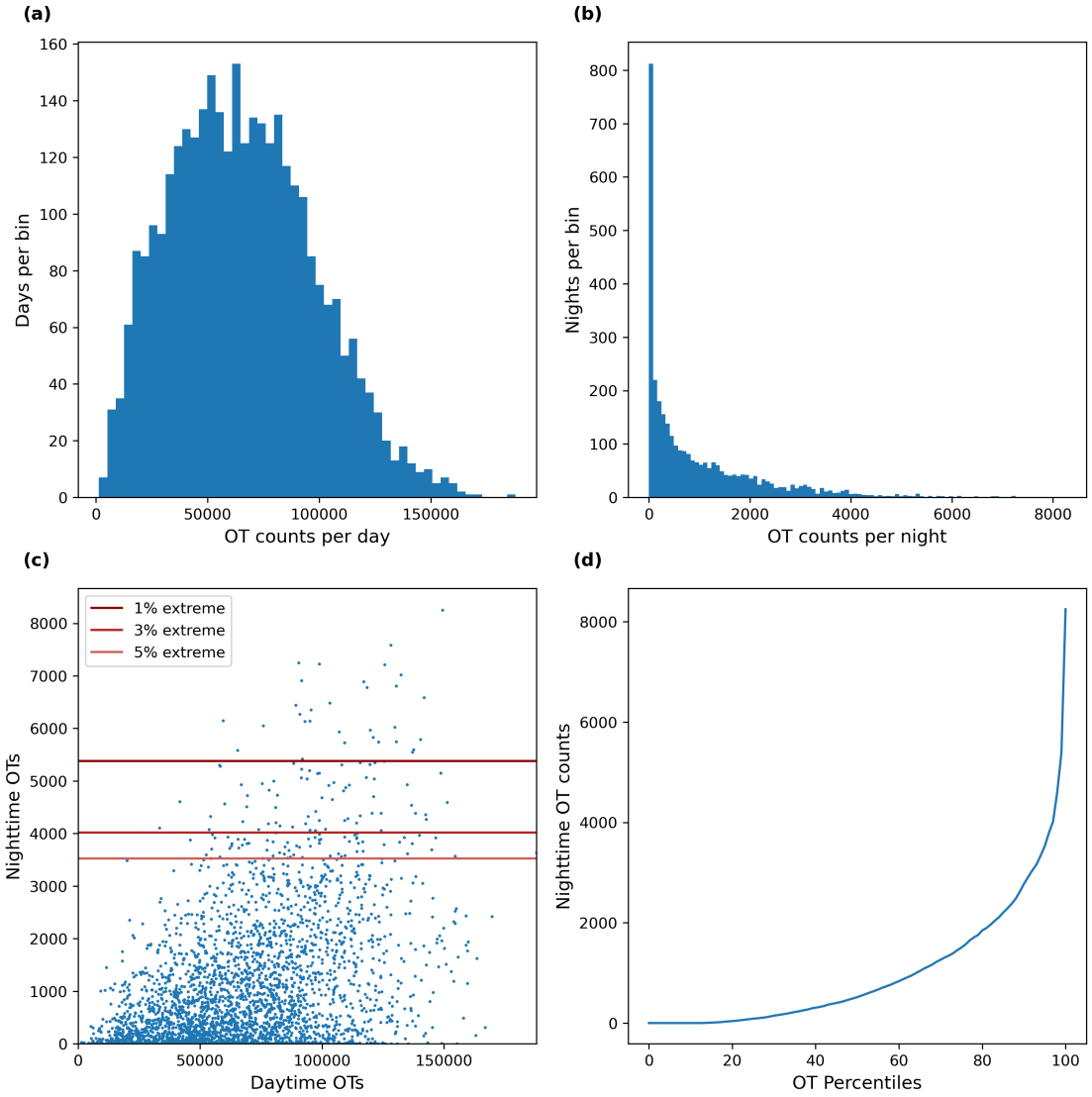


Figure 4.1: a) Distribution of daytime OT counts, summed over the entire daytime period (04:00 to 18:00 EAT) and the whole 840 by 840 domain. b) Distribution of the nighttime OT counts over Lake Victoria. c) the correlation of daytime and nighttime OT counts per sample, along with the thresholds for the 95^{th}, 97^{th} and 99^{th} OT percentiles. d) The relationship between nightly OT count and OT percentile.

performed. Pixels with a Spearman coefficient above 0.15 and p-value below 0.01 are selected for further analysis (Figure 4.2d). This spatial selection yields 34219 highly-correlated pixels and is used to construct the 'spatial subset', which is stored as a 3288 by 34219 tensor. Similarly, the 'temporal subset' has a shape of (3288, 36) and is also saved as .pt.

The first correlation analysis is then repeated but based on only the highly-correlated pixels in the highly-correlated time intervals. The 36 highly-correlated time intervals are selected, going from a $(3288, 56, 840, 840)$ tensor to a $(3288, 36, 840, 840)$ tensor. Then the spatial dimension is reduced to obtain a $(3288, 36, 34219)$ tensor, which is summed along the remaining time intervals and pixels to obtain a tensor of shape (3288). Using this spatial and temporal selection to calculate the Spearman rank correlation results in an improved correlation coefficient of 0.66 with a p-value of 0, compared to a correlation of 0.42 before the selection (Figure 4.3).

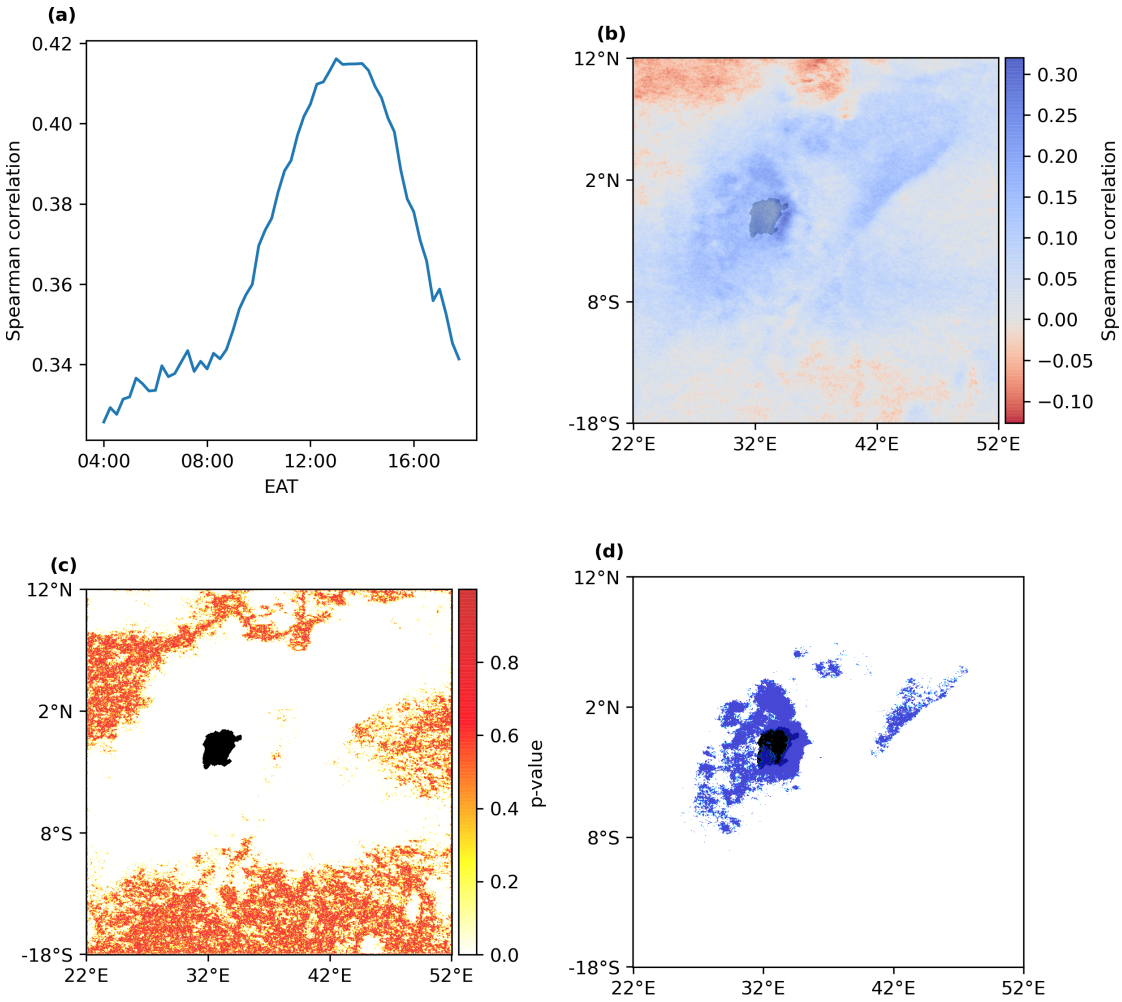


Figure 4.2: a) Temporal Spearman correlation coefficients, each time interval is aggregated over the spatial domain. b) Spatial Spearman correlation coefficients, each pixel is aggregated over the 04:00 to 18:00 EAT time domain. c) p-value of the pixels in the study area for the hypothesis that the daytime OTs and nighttime OTs are uncorrelated. d) The spatial mask for the daytime pixels, which have a high certainty ($p < 0.01$) of high correlation ($r_s > 0.15$) to the OTs of the succeeding night.

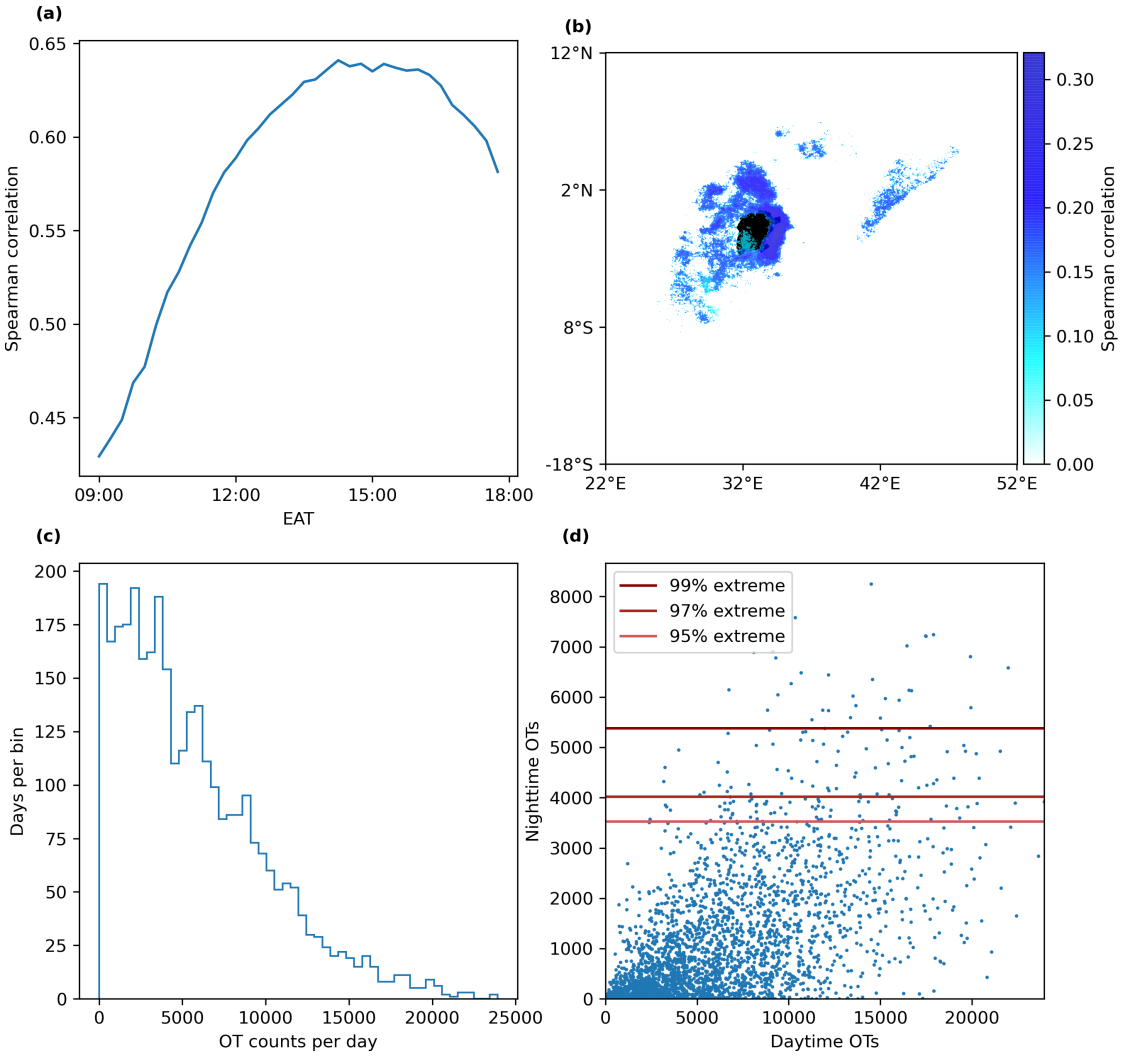


Figure 4.3: a) Temporal Spearman correlation coefficients, for each highly-correlated time interval (09:00–18:00 EAT), only the highly-correlated pixels (Figure 4.2d) were aggregated. b) Spatial Spearman correlation coefficients for the aggregated highly-correlated time intervals (09:00–18:00 EAT) in each of the highly-correlated pixels. c) The distribution of daily OT counts aggregated over the highly-correlated pixels in the highly-correlated time ranges. d) The relationship between (i) daily OT counts summed over the highly-correlated pixels in the highly-correlated time ranges and (ii) the subsequent nighttime OT counts over Lake Victoria.

4.2 Machine Learning Analysis

Here, the effect of the new OT dataset on the univariate logistic regression model of Thiery et al. (2017) is investigated. Furthermore, we evaluate the possibility of improving VIEWS by (i) pre-filtering the input features by the high correlation thresholds of Section 4.1, and (ii) introducing multivariate logistic regression. In the latter, the highly-correlated temporal and spatial subsets are each aggregated over n variables, which serve as the input features for different logistic regression models. First, we perform univariate logistic regression, similar to Thiery et al. (2017), which is followed by a univariate logistic regression with the aggregation of the highly-correlated pixels in the highly-correlated time intervals. In addition, the multivariate logistic regression is implemented with two, three, four and six temporal input variables, and two, three and four spatial input aggregates (see Section 3.2.4). For each of these unique sets of input variables, 75 configurations are considered, which only differ in the weight factor for the high storm-intensity class (1). For each configuration 1000 models are trained to minimize the variance of each configuration. Inter-model comparison results in high variances, due to the small amount of 1-class samples in the test set. On average, each test set only contains ~6 samples of the high storm-intensity (1) class ($32 \cdot 20\%$). All configurations show a standard deviation of ~1.5% for the events below the 99^{th} OT percentile (0) and ~15% for the events above the 99^{th} OT percentile (1).

For each different set of input variables, we draw a best-fit high-order polynomial curve connecting the average performance of its 75 configurations (Figure 4.4). This curve is subsequently used to derive the optimal performance (marked in red) of this set of input variables, where the optimal model is defined as having the smallest difference between H and $(1 - F)$. Thus, these models have equal performance in terms of hit rate H and false alarm rate F. The optimal models following the approach of Thiery et al. (2017) have a lower performance than reported on the 2010 OT dataset, the hit rate is 0.76 and the false alarm rate 0.24. The univariate models based on the temporal and spatial high-correlation subsets, have a H of 0.83 and a F of 0.18 (Figure 4.4).

Next, the same approach is applied to the models with two, three, four and six temporal input variables (Figure 4.5). The model with two input variables yields the best results, the models show varying degrees of overfitting when comparing the performances of the training and test set. The model with two input variables has an optimal H of 0.85 and F of 0.15. The optimal three, four and six temporal parameter models have a hit rate of 0.84, 0.83 and 0.83 and false alarm ratio of 0.16, 0.17 and 0.17, respectively. Similarly, the two, three and four spatial input variables were considered in 75 weight-configurations, which were all trained with 500 models each to minimize the variance. Here, the three-variable model outperforms the two-variable and four-variable models and has a H of 0.85 and a F of 0.15 (Figure 4.6). The two-variable spatial model has a H of 0.81 and F of 0.18, while this is 0.84 and 0.16 for the four-variable spatial model. All three test set performance curves show signs of some degree of overfitting when compared to the performance of the training set.

.

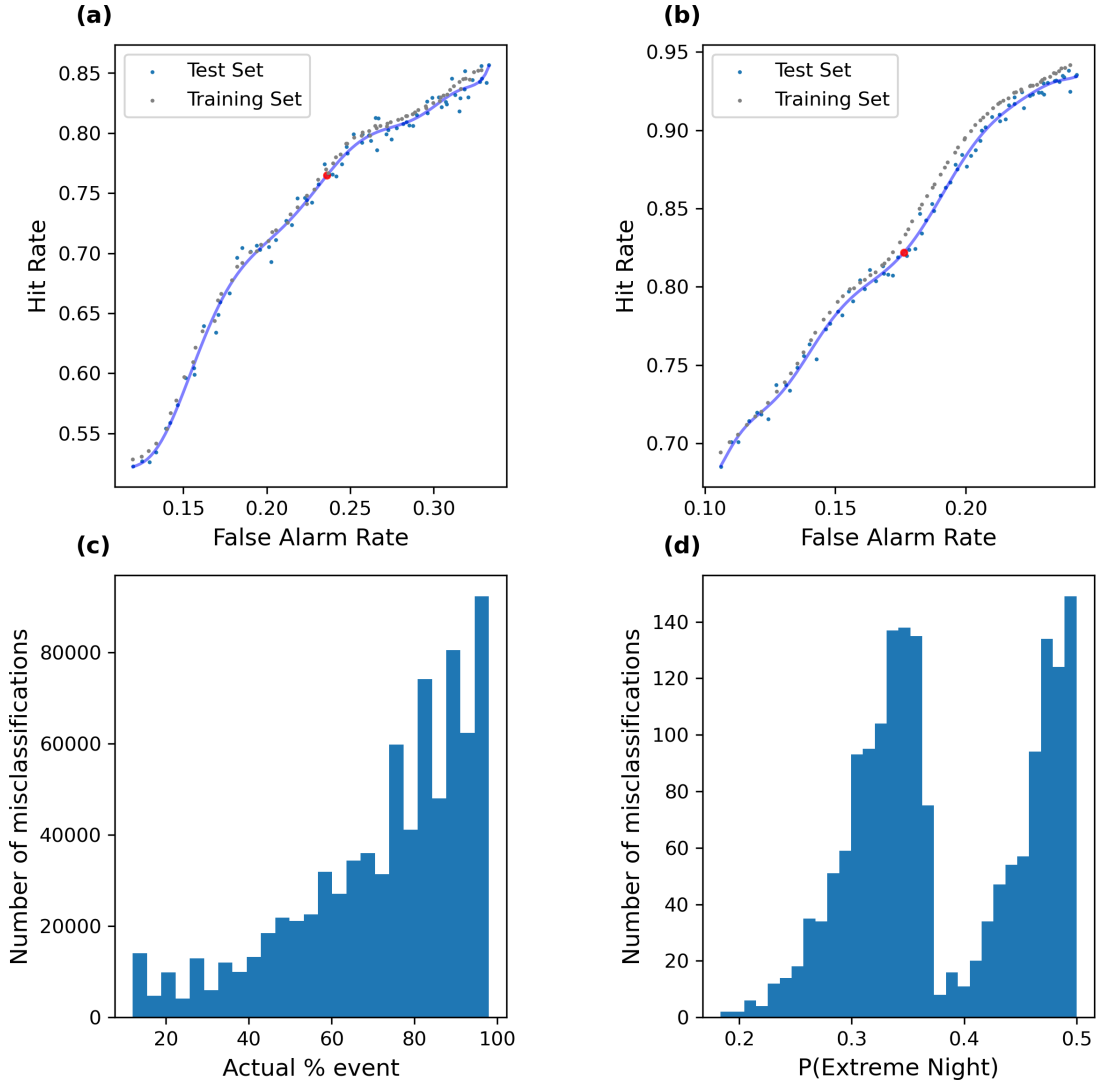


Figure 4.4: a) The performance of the univariate logistic regression models with the set-up of Thiery et al. (2017), the optimal model is marked in red. b) The performance of the univariate logistic regression models with the correlation-based pre-filtering of input data. c) The actual type of events that were misclassified as high-intensity events. d) the modelled high-intensity probability for missed high-intensity events.

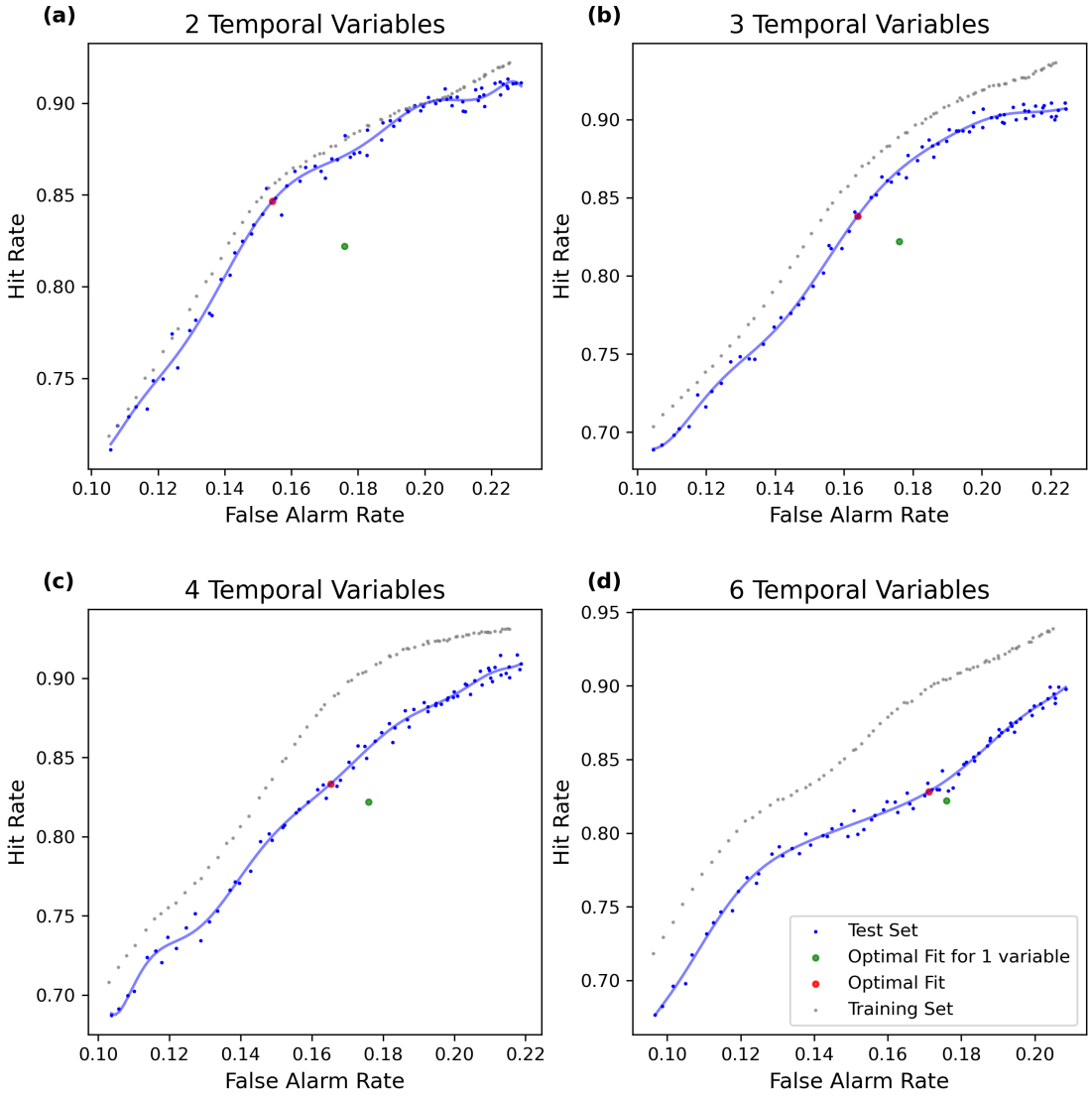


Figure 4.5: The performance of the four logistic regression models with two, three, four and six temporal input variables. The green dot is the performance of the optimal univariate model of Figure 4.4, while the red dot marks the optimal performance of each set-up. In blue is the average performance of each configuration on the test set, while the average performance for each configuration on the training set is marked in gray. The blue curve corresponds to the high-order polynomial that was fit on the test set performance.

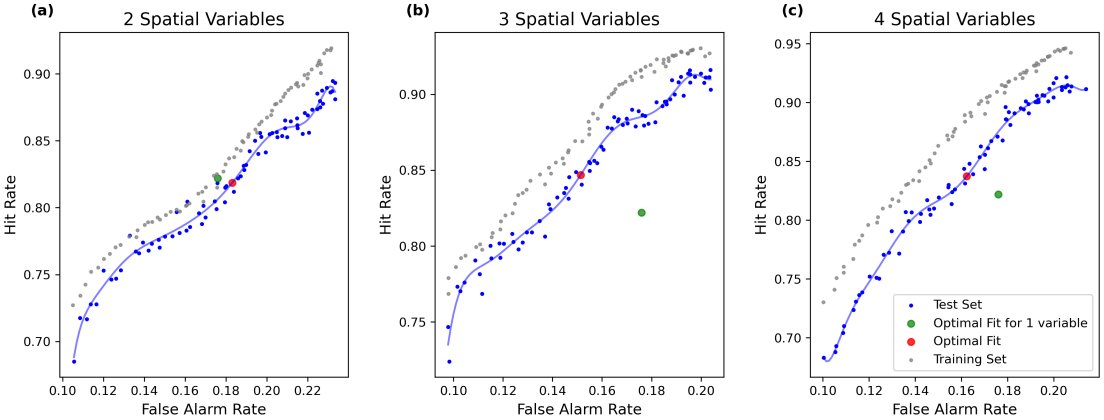


Figure 4.6: The performance of the three logistic regression models with two, three and four spatial input variables. The green dot is the performance of the optimal univariate model of Figure 4.4, while the red dot marks the optimal performance of each set-up. In blue is the average performance of each configuration on the test set, while the average performance for each configuration on the training set is marked in gray. The blue curve corresponds to the high-order polynomial that was fit on the test set performance.

4.3 Deep Learning Analysis

Here, we analyse the results from 20 ensembles, each containing 50 NNs, which are trained on the temporal subset. Furthermore, 12 ensembles, each containing 25 NNs trained on the spatial subset are regarded (Figure 4.7). The 20 ensemble models subsequently use hard (red) and soft (green) voting to predict the labels in the test set. Hard voting pulls each NN's prediction to either 0 (no thunderstorm-intense night) or 1 (thunderstorm-intense night) and returns the class that was predicted the most. Soft voting accounts the probability of each prediction, in soft voting the mean of the predictions is taken, which is then pulled to 0 or 1. Lastly, for each ensemble, the NNs are ranked based on their loss, and only the best half are retained. Then hard and soft voting is performed again for each ensemble, which now only considers these best-performing models. An identical approach is performed on the ensemble models trained on the spatial subset.

The average H and F of the NNs trained on the temporal data is 0.73 and 0.15 respectively (Figure 4.7). The hard voting on the temporal ensembles has a mean H of only 0.27 but a F of 0.03, while the mean soft voting ensembles is 0.74 for H and 0.15 for F. The temporal ensembles which only consider the best 50% of their NNs have a mean 0.32 H and 0.04 F when hard voting, and 0.68 and 0.14 when soft voting. Similarly, the average performance of the NNs trained on the spatial data is 0.51 and 0.11 for H and F respectively. The ensembles of the NNs trained on spatial information have a mean H of only 0.05 but an F of 0.01 when hard voting and an H and F of 0.45 and 0.09 when soft voting. The ensembles that only consider the votes of the 50% best-performing NNs have a mean H of 0.10 and F of 0.02 when hard voting, and 0.44 and 0.08 when soft voting. A number of spatially-explicit warning system are designed via the CNN class (see subsection 3.2.5). However, preliminary results currently show little improvement to random guessing.

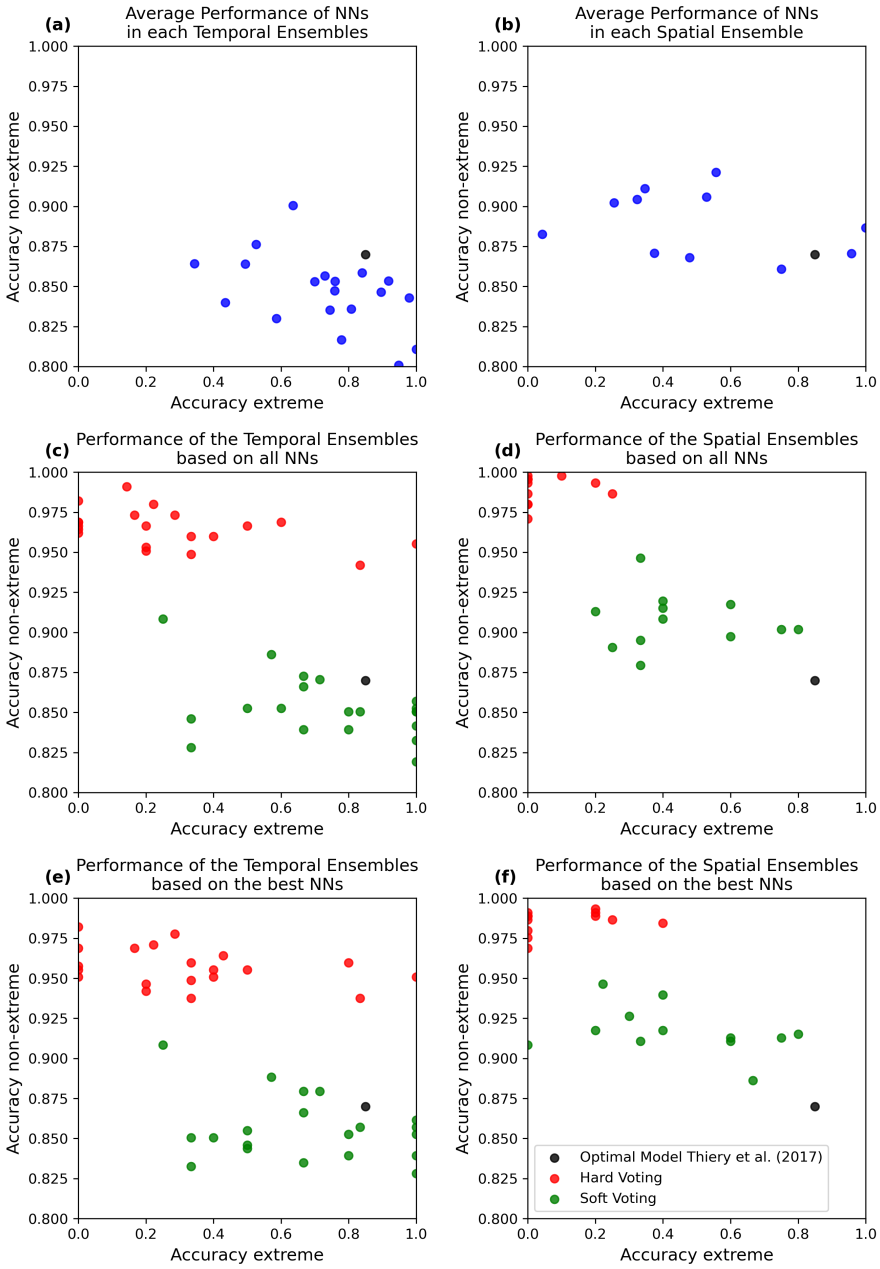


Figure 4.7: a) The average performance of the individual NNs in the temporal ensembles. The average performance of the individual NNs in the spatial ensembles. c) The performance of each temporal ensemble when using soft and hard voting. d) The performance of each spatial ensemble when using soft and hard voting. e) The performance of each temporal ensemble when using hard and soft voting only considering the 50% of NNs with the lowest loss. f) The performance of each spatial ensemble when using hard and soft voting only considering the 50% of NNs with the lowest loss.

Chapter 5

Discussion

5.1 Correlation-based Feature Selection

The spatial Spearman correlation coefficients (Figure 4.3) display a similar pattern as those in described in Thiery et al. (2017). Lake Victoria's eastern lake shore has the highest correlation, which is the result of intense convection forming through the combination of the lake breeze system with the steep topography east of the lake (Figure 1.1). The stronger the heating of the slopes in the afternoon, the more precipitation will occur during this period (Anyah et al., 2006). However, during the night this reversal will subsequently be stronger as well, leading to an increase in the katabatic winds (Anyah et al., 2006). A similar, but weaker phenomenon happens for the mountains west of the lake. Lastly, the daytime thunderstorms over the coastal region of South-Somalia also show a statistically significant relationship with the nighttime storms over Lake Victoria. The temporal correlation between the daytime thunderstorm activity around Lake Victoria and subsequent nighttime storm activity over the lake, which was not analysed directly in Thiery et al. (2017), peaks around 14:00 to 16:00 EAT. This corresponds to the peak in afternoon rainfall for the Lake Victoria region and thus agrees with real-world observations and simulations (Thiery et al., 2015).

The size of each sample can be reduced spatially by selecting only the pixels with a Spearman rank correlation coefficient > 0.15, and temporally by retaining only the 09:00 to 18:00 EAT time interval. This results in a selection of 36 time intervals with 34219 pixels each for all samples in the dataset. This differs from Thiery et al. (2017) in two ways. First, we consider pixels from the entire OT dataset domain (18°S–12°N, 22°E–52°E), as opposed to the 4°S–2°N, 30°E–36°E window of Thiery et al. (2017). Secondly, temporal filtering for the daytime OTs was not considered previously. The feature selection significantly increases the quality of the information in each sample (Figure 4.3). This improves the performance of ML models trained on this data (Franchina & Sergiani, 2019; Miller et al., 2019; Picard et al., 2020). Furthermore, it allows to reduce the size of each data sample for training extensive NNs. The required amount of training data for these models scales with the total information that is applied to the system (Kira & Rendell, 1992; Guyon & Elisseeff, 2003; Chandrashekar & Sahin, 2014). Here, training data is limited and this removal of low-correlation information can be of high importance for the successful training of NNs.

5.2 Machine Learning Analysis

Replicating the logistic regression model as described in Thiery et al. (2017) on the new OT dataset results in reduced performance. After performing identical pre-processing, we obtain a hit rate of 0.76 and a false alarm rate of 0.24, while Thiery et al. (2017) found a hit rate of 0.85 and false alarm rate of 0.13. The discrepancy between these observations could be explained by two differences in model set-ups. First, the version and length of NASA's LaRC OT dataset used to obtain the input and labels differs in both methods. Thiery et al. (2017) is based on the 2010 version, which uses a binary classification of OTs, while the employed 2021 version has, among other improvements, OT probability estimates instead. Furthermore, Thiery et al. (2017) considers the 01/01/2005–31/12/2013 period, while we have access to two additional years of data. Secondly, the methodological approaches differ slightly, as Thiery et al. (2017) employs a 'leave-one-year-out cross-validation' algorithm, while we iteratively trained a large number of models to obtain a well-fit average. However, both approaches are similar and should converge to more or less the same result. This implies that the largest part of the discrepancy between the two results is likely coming from differences in the dataset. Without additional improvements, the new version of the OT algorithm thus reduces the performance of VIEWS.

Next, we studied the effect of a more elaborate pre-processing of the OT data and a better informed pre-selection of the pixels used in the model. Here, the aggregation of the daytime OTs only considers the most correlated spatial pixels and time intervals in the domain. Performing the univariate logistic regression on this data yields a significantly improved H of 0.83 and F of 0.18 (Figure 4.4). Furthermore, multivariate logistic regression on the same temporal and spatial subsets outperform the univariate model (Figures 4.5 and 4.6). The two-variable temporal and three-variable spatial models have the highest performance (H=0.85, F=0.15), matching closely with the original prediction power of Thiery et al. (2017), which obtained a H of 0.85 and F of 0.13.

The authors of NASA's LaRC OT dataset state that the 2015 version of the algorithm provides a more realistic end-result than the 2010 version, and that the 2021 version further improves this 2015 update (K. Bedka & Khlopenkov, 2016; Khlopenkov et al., 2021). The large difference between the 2010 and 2021 datasets is the 99^{th} OT percentile, which corresponds to 2239 OTs in the 2010 version and 5379 OTs in the 2021 update, capturing more thunderstorms (Thiery et al., 2017). To offer state-of-the-art performance, the VIEWS project should be based on the most recent version of the OT dataset detection algorithm. Therefore, we propose to update the VIEWS project to the 2021 version of NASA's LaRC OT dataset and use the two-variable temporal model as benchmark for future references, which obtains a hit rate of 0.85 and a false alarm ratio of 0.15. Furthermore, the multi-variate models show (small) signs of overfitting when comparing their performance against the training set. If the size of the dataset can be increased, overfitting could be reduced. Currently, the acquired dataset ranges from 01/01/2005 until 31/12/2015. However, the SEVIRI product is still operational, which implies that the OT dataset can be extended with the time period of 01/01/2016 until present. This would increase the dataset size by ~50% and could thus lead to improved performances of the overfitting models. We have been in contact with the authors of the OT detection algorithm and hope to obtain this dataset in the near-future.

5.3 Deep Learning Analysis

Lastly, we evaluated whether DL can improve the performance of VIEWS and investigated whether we could provide spatially explicit predictions using DL. The average performance for NNs trained on the temporal subset ($H = 0.73$, $F = 0.15$) shows that there is potential for DL models to improve VIEWS. The performance measure of the NNs is currently heavily influenced by the small size of the test set. By definition, there are only 32 nights above the 99^{th} OT percentile (class 1) in the entire dataset, and the test set includes only 15%, which corresponds ~4-5 heavy storm nights. This causes a high variance in the performance of the NNs and is the main reason for the high number of NNs that are trained. The performance of the ensembles indicates that, grouping similar NNs together does not improve performance in terms of H. The lack of diversity in the considered NNs is likely to be the root cause of this issue (Hansen & Salamon, 1990; Krogh, Vedelsby, et al., 1995; G. Brown, 2004; Z.-H. Zhou, 2019). Moreover, the possibility of including spatial predictions by training CNNs was considered. In the current preliminary results, no model has been found that provides reliable estimates.

5.4 Future Research

In this thesis, only a small fraction of available ML and DL techniques are considered. Other opportunities exist to employ ML and DL to improve or extend the VIEWS warning system. Here, a number of these alternative approaches are proposed, mainly focusing on the spatially-explicit predictions. First, as mentioned in Section 5.2, the extension of NASA's LaRC OT dataset could boost the performance of the multivariate logistic regression, as well as the DL models by providing more training data in the near-future. Secondly, the CNNs for spatially-aware predictions could be trained to consider multiple fields as input e.g., *OT probability* and *HIWC probability*, or combine the OT dataset with other existing datasets with proxies for storm activity, like the Global Lightning dataset from Earths Network (Lapierre et al., 2019; Virts & Goodman, 2020). Moreover, ensemble models could be trained, where individual models are trained on different input variables. Potentially this could overcome the lack of diversity reported earlier.

Lastly, to build an operational early warning system, the optimal input data should be available at real-time. Therefore, it is optimal if a NN (or any other model) can be trained using the 'raw' daytime SEVIRI satellite time-series as input to predict the nighttime storms over Lake Victoria. Transfer learning, based on other models trained on SEVIRI images available in literature, could then provide an interesting opportunity. Ideally, the SEVIRI dataset was used in this study to build a spatially-explicit storm forecasting system. This was however not possible due to technical issues and large download times. The 2005-2015 SEVIRI dataset for the Lake Victoria region currently takes ~9 months to fully acquire via EUMETSAT and ~6 months via the Belgian Royal Meteorological Institute (RMI). We are acquiring this dataset and aim to perform these suggestions in future work.

Chapter 6

Conclusions

Every year, thousands of fishermen die in heavy nightly thunderstorms on Lake Victoria, located in East Africa. In this thesis, we investigated the use of ML and DL to improve extreme storm predictions over Lake Victoria. The research was subdivided in five objectives: first we evaluated the effect of the new 2021 LaRC OT dataset on the performance of the VIEWS early warning system. Second, we investigated whether a statistically-based feature selection can improve the performance of VIEWS, and third whether an improved ML approach, considering multivariate input, can provide better results than the current state of the art. In the fourth objective we analyse if VIEWS can be improved using NNs. Finally, we examined the possibility of providing spatially-explicit forecasts of intense thunderstorms over Lake Victoria using DL.

The original VIEWS experiment is replicated with the most recent version of NASA's LaRC OT dataset. Performance on this new dataset is significantly lower than the original by Thiery et al. (2017). However, the improved temporal and spatial selection of daytime OTs is able to improve these univariate classifiers. Moreover, multivariate logistic regression on these temporal and spatial subsets further increased the prediction power, close to the original performance of the VIEWS project. Then a large number of NNs were constructed to evaluate their efficiency for predicting intense storm nights over Lake Victoria. Individual NNs show promising results but their ensembles were unsatisfactory until now, most likely due to a lack in diversity between the NNs. Then, a number of CNNs were trained to provide spatially-explicit predictions of nighttime thunderstorms over the lake. However, no conclusive statements for these models can yet be made. Lastly, we propose a number of methods that can be applied to further extend VIEWS in future research such as training models directly on the SEVIRI dataset, possibly via transfer learning.

Overall, in this thesis we explored the use of ML and DL for predicting nightly thunderstorms. We were able to improve on the state of the art of the VIEWS project. Our results show that these techniques provide a promising path for future research to improve warning systems and potentially save lives of thousands of fishermen.

Bibliography

Adeli, H., & Wu, M. (1998). Regularization neural network for construction cost estimation. *Journal of construction engineering and management*, *124*(1), 18–24.

Aizman, A., Maltby, G., & Breuel, T. (2019). High performance i/o for large scale deep learning. *2019 IEEE International Conference on Big Data (Big Data)*, 5965–5967.

Aizman, A., Maltby, G., & Breuel, T. (2020). *Efficient pytorch i/o library for large datasets, many files, many gpus*. https://pytorch.org/blog/efficient-pytorch-io-library-for-large-datasets-many-files-many-gpus/

Akalin, A. (2020). *Computational genomics with r*. CRC Press.

Akgün, E., & Demir, M. (2018). Modeling course achievements of elementary education teacher candidates with artificial neural networks. *International Journal of Assessment Tools in Education*, *5*(3), 491–509.

Alapatt, D., Mascagni, P., Srivastav, V., & Padoy, N. (2020). Artificial intelligence in surgery: Neural networks and deep learning. *arXiv preprint arXiv:2009.13411*.

Aminou, D. (2002). Msg's seviri instrument. *ESA Bulletin(0376-4265)*, (111), 15–17.

Anthony, M., & Bartlett, P. L. (2009). *Neural network learning: Theoretical foundations*. cambridge university press.

Anyah, R. O., Semazzi, F. H., & Xie, L. (2006). Simulated physical mechanisms associated with climate variability over lake victoria basin in east africa. *Monthly weather review*, *134*(12), 3588–3609.

Ba, M. B., & Nicholson, S. E. (1998). Analysis of convective activity and its relationship to the rainfall over the rift valley lakes of east africa during 1983–90 using the meteosat infrared channel. *Journal of Applied Meteorology*, *37*(10), 1250–1264.

Bauer, P., Thorpe, A., & Brunet, G. (2015). The quiet revolution of numerical weather prediction. *Nature*, *525*(7567), 47–55.

Bedka, K., Brunner, J., Dworak, R., Feltz, W., Otkin, J., & Greenwald, T. (2010). Objective satellite-based detection of overshooting tops using infrared window channel brightness temperature gradients. *Journal of applied meteorology and climatology*, *49*(2), 181–202.

Bedka, K., & Khlopenkov, K. (2016). A probabilistic multispectral pattern recognition method for detection of overshooting cloud tops using passive satellite imager observations. *Journal of Applied Meteorology and Climatology*, *55*(9), 1983–2005.

Bjorck, J., Gomes, C., Selman, B., & Weinberger, K. Q. (2018). Understanding batch normalization. *arXiv preprint arXiv:1806.02375*.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. *Proceedings of compstat'2010* (pp. 177–186). Springer.

Bottou, L. (2012). Stochastic gradient descent tricks. *Neural networks: Tricks of the trade* (pp. 421–436). Springer.

Brown, G. (2004). *Diversity in neural network ensembles* (Doctoral dissertation). Citeseer.

Brown, R. A. (1991). *Fluid mechanics of the atmosphere*. Academic Press.

Bryan, G. H., Wyngaard, J. C., & Fritsch, J. M. (2003). Resolution requirements for the simulation of deep moist convection. *Monthly Weather Review*, *131*(10), 2394–2416.

Byers, H. R., & Rodebush, H. R. (1948). Causes of thunderstorms of the florida peninsula. *Journal of Atmospheric Sciences*, *5*(6), 275–280.

Cannon, T. (2014). World disasters report 2014–focus on culture and risk.

Caruana, R., Lawrence, S., & Giles, L. (2001). Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, 402–408.

Chamberlain, J., Bain, C., Boyd, D., McCourt, K., Butcher, T., & Palmer, S. (2014). Forecasting storms over lake victoria using a high resolution model. *Meteorological Applications*, *21*(2), 419–430.

Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*(1), 16–28.

Chauvin, Y., & Rumelhart, D. E. (2013). *Backpropagation: Theory, architectures, and applications*. Psychology press.

Coiffier, J. (2011). *Fundamentals of numerical weather prediction*. Cambridge University Press.

Dara, S., & Tumma, P. (2018). Feature extraction by using deep learning: A survey. *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 1795–1801.

Elliott, D. L. (1993). *A better activation function for artificial neural networks* (tech. rep.).

EUMETSAT. (n.d.). Netcdf format for eumetsat data centre products. https://www.eumetsat.int/data-centre-netcdf-format

Falcon, e. a., WA. (2019). Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, *3*.

Federer, B., & Waldvogel, A. (1975). Hail and raindrop size distributions from a swiss multicell storm. *Journal of Applied Meteorology and Climatology*, *14*(1), 91–97.

Folk, M., Heber, G., Koziol, Q., Pourmal, E., & Robinson, D. (2011). An overview of the hdf5 technology suite and its applications. *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, 36–47.

Franchina, L., & Sergiani, F. (2019). High quality dataset for machine learning in the business intelligence domain. *Proceedings of SAI Intelligent Systems Conference*, 391–401.

Gelaro, R., McCarty, W., Suárez, M. J., Todling, R., Molod, A., Takacs, L., Randles, C. A., Darmenov, A., Bosilovich, M. G., Reichle, R., et al. (2017). The modern-era retrospective analysis for research and applications, version 2 (merra-2). *Journal of climate*, *30*(14), 5419–5454.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Group, T. H. (2000). *Hierarchical data format version 5*. http://www.hdfgroup.org/HDF5

Gumisiriza, R., Mshandete, A. M., Rubindamayugi, M. S., Kansiime, F., & Kivaisi, A. K. (2009). Nile perch fish processing waste along lake victoria in east africa: Auditing and characterization. *African Journal of Environmental Science and Technology*, *3*(1), 013–020.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, *3*(Mar), 1157–1182.

Haltiner, G. J., & Williams, R. T. (1980). *Numerical prediction and dynamic meteorology* (tech. rep.).

Hanley, K. E., Pirret, J. S., Bain, C. L., Hartley, A. J., Lean, H. W., Webster, S., & Woodhams, B. J. (2021). Assessment of convection-permitting versions of the unified model over the lake victoria basin region. *Quarterly Journal of the Royal Meteorological Society*, *147*(736), 1642–1660.

Hanley, K. E., Plant, R. S., Stein, T. H., Hogan, R. J., Nicol, J. C., Lean, H. W., Halliwell, C., & Clark, P. A. (2015). Mixing-length controls on high-resolution simulations of convective storms. *Quarterly Journal of the Royal Meteorological Society*, *141*(686), 272–284.

Hansen, L. K., & Salamon, P. (1990). Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, *12*(10), 993–1001.

Harrington, P. (2012). *Machine learning in action*. Simon; Schuster.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). Overview of supervised learning. *The elements of statistical learning* (pp. 9–41). Springer.

Hatchett, B., Nauslar, N., Kaplan, M., Smith, C., & Nelson, K. (2019). Slope winds. *Encyclopedia of Wildfires and Wildland-Urban Interface Fires*.

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. *Neural networks for perception* (pp. 65–93). Elsevier.

Hinton, G., Srivastava, N., & Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, *14*(8), 2.

Holton, J. R. (1973). An introduction to dynamic meteorology. *American Journal of Physics*, *41*(5), 752–754.

Hutson, M. (2021). Who should stop unethical ai. *The Newyorker Annals of Technology*.

IOC, ICSU, UNESCO, UNEP, & WMO. (2019). Plan for improving observations around lake victoria that support numerical weather predictions, climate services and adaptation.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International conference on machine learning*, 448–456.

Jacobs, A. (2009). The pathologies of big data. *Communications of the ACM*, *52*(8), 36–44.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural networks*, *1*(4), 295–307.

Jiang, T., Gradus, J. L., & Rosellini, A. J. (2020). Supervised machine learning: A brief primer. *Behavior Therapy*, *51*(5), 675–687.

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260.

Kahn, B. H., Takahashi, H., Stephens, G. L., Yue, Q., Delanoë, J., Manipon, G., Manning, E. M., & Heymsfield, A. J. (2018). Ice cloud microphysical trends observed by the atmospheric infrared sounder. *Atmospheric Chemistry and Physics*, *18*(14), 10715–10739.

Kalnay, E. (2003). *Atmospheric modeling, data assimilation and predictability*. Cambridge university press.

Khlopenkov, K. V., Bedka, K. M., Cooney, J. W., & Itterly, K. (2021). Recent advances in detection of overshooting cloud topsfrom longwave infrared satellite imagery. *Journal of Geophysical Research: Atmospheres*.

Kira, K., & Rendell, L. A. (1992). A practical approach to feature selection. *Machine learning proceedings 1992* (pp. 249–256). Elsevier.

Kite, G. (1982). Analysis of lake victoria levels. *Hydrological Sciences Journal*, *27*(2), 99–110.

Kizza, M., Rodhe, A., Xu, C.-Y., Ntale, H. K., & Halldin, S. (2009). Temporal rainfall variability in the lake victoria basin in east africa during the twentieth century. *Theoretical and applied climatology*, *98*(1), 119–135.

Kizza, M., Westerberg, I., Rodhe, A., & Ntale, H. K. (2012). Estimating areal rainfall over lake victoria and its basin using ground-based and satellite data. *Journal of Hydrology*, *464*, 401–411.

Krogh, A., Vedelsby, J. et al. (1995). Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, *7*, 231–238.

Kudhongania, A., & Cordone, A. J. (1974). Batho-spatial distribution pattern and biomass estimate of the major demersal fishes in lake victoria. *African Journal of Tropical Hydrobiology and Fisheries*, *3*(1), 15–31.

Kwena, Z. A., Bukusi, E., Omondi, E., Ng'Ayo, M., & Holmes, K. K. (2012). Transactional sex in the fishing communities along lake victoria, kenya: A catalyst for the spread of hiv. *African Journal of AIDS Research*, *11*(1), 9–15.

Lai, S., Xu, L., Liu, K., & Zhao, J. (2015). Recurrent convolutional neural networks for text classification. *Twenty-ninth AAAI conference on artificial intelligence*.

Lapierre, J., Hoekzema, M., Stock, M., Merrill, C., & Thangaraj, S. C. (2019). Earth networks lightning network and dangerous thunderstorm alerts. *2019 11th Asia-Pacific International Conference on Lightning (APL)*, 1–5.

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

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541–551.

Lee, B., Amaresh, S., Green, C., & Engels, D. (2018). Comparative study of deep learning models for network intrusion detection. *SMU Data Science Review*, *1*(1), 8.

Li, M., Zhang, T., Chen, Y., & Smola, A. J. (2014). Efficient mini-batch training for stochastic optimization. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 661–670.

Liu, X., Duan, H., Huang, W., Guo, R., & Duan, B. (2021). Classified early warning and forecast of severe convective weather based on lightgbm algorithm. *Atmospheric and Climate Sciences*, *11*(2), 284–301.

LVFO. (2016). *Fisheries management plan iii (fmp iii) for lake victoria fisheries 2016-2020*. Lake Victoria Fisheries Organization Secretariat.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural networks*, *10*(9), 1659–1671.

Masi, F., Stefanou, I., Vannucci, P., & Maffi-Berthier, V. (2021). Thermodynamics-based artificial neural networks for constitutive modeling. *Journal of the Mechanics and Physics of Solids*, *147*, 104277.

Miller, P. E., Pawar, S., Vaccaro, B., McCullough, M., Rao, P., Ghosh, R., Warier, P., Desai, N. R., & Ahmad, T. (2019). Predictive abilities of machine learning techniques may be limited by dataset characteristics: Insights from the unos database. *Journal of cardiac failure*, *25*(6), 479–483.

Mills, E., Gengnagel, T., & Wollburg, P. (2014). Solar-led alternatives to fuel-based lighting for night fishing. *Energy for Sustainable Development*, *21*, 30–41.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of machine learning*. MIT press.

Mungai, D. N. (1984). *Analysis of some seasonal rainfall characteristics in the lake victoria region of kenya* (Doctoral dissertation).

Murugan, P., & Durairaj, S. (2017). Regularization and optimization strategies in deep convolutional neural network. *arXiv preprint arXiv:1712.04711*.

NASA. (2011). Netcdf-4/hdf5 file format. https://earthdata.nasa.gov/esdis/eso/standards-and-references/netcdf-4hdf5-file-format

Negri, A. J., & Adler, R. F. (1981). Relation of satellite-based thunderstorm intensity to radar-estimated rainfall. *Journal of Applied Meteorology and Climatology*, *20*(3), 288–300.

Njiru, J., van der Knaap, M., Kundu, R., & Nyamweya, C. (2018). Lake victoria fisheries: Outlook and management. *Lakes & Reservoirs: Research & Management*, *23*(2), 152–162.

Odi, U., & Nguyen, T. (2018). Geological facies prediction using computed tomography in a machine learning and deep learning environment. *SPE/AAPG/SEG Unconventional Resources Technology Conference*.

Parpart, P., Jones, M., & Love, B. C. (2018). Heuristics as bayesian inference under extreme priors. *Cognitive psychology*, *102*, 127–144.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pytorch.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32*, 8026–8037.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Phaisangittisagul, E. (2016). An analysis of the regularization between l2 and dropout in single hidden layer neural network. *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, 174–179.

Picard, S., Chapdelaine, C., Cappi, C., Gardes, L., Jenn, E., Lefèvre, B., & Soumarmon, T. (2020). Ensuring dataset quality for machine learning certification. *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 275–282.

Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Physical review letters*, *59*(19), 2229.

Prechelt, L. (1998). Early stopping-but when? *Neural networks: Tricks of the trade* (pp. 55–69). Springer.

Proud, S. R. (2015). Analysis of overshooting top detections by meteosat second generation: A 5-year dataset. *Quarterly Journal of the Royal Meteorological Society*, *141*(688), 909–915.

Ratvasky, T. P., Harrah, S. D., Strapp, J. W., Lilie, L. E., Proctor, F. H., Strickland, J. K., Hunt, P. J., Bedka, K., Diskin, G., Nowak, J. B., et al. (2019). Summary of the high ice water content (hiwc) radar flight campaigns.

Rebekić, A., Lončarić, Z., Petrović, S., & Marić, S. (2015). Pearson's or spearman's correlation coefficient-which one to use? *Poljoprivreda*, *21*(2), 47–54.

Rew, R., Hartnett, E., Caron, J., et al. (2006). Netcdf-4: Software implementing an enhanced data model for the geosciences. *22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanograph, and Hydrology*, *6*.

Reynolds, D. W. (1980). Observations of damaging hailstorms from geosynchronous satellite digital data. *Monthly Weather Review*, *108*(3), 337–348.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rumelhart, D. E., Widrow, B., & Lehr, M. A. (1994). The basic ideas in neural networks. *Communications of the ACM*, *37*(3), 87–93.

Sak, H., Senior, A. W., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling.

Sandvig, C., Hamilton, K., Karahalios, K., & Langbort, C. (2015). Can an algorithm be unethical. *65th annual meeting of the International Communication Association*.

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization? *Proceedings of the 32nd international conference on neural information processing systems*, 2488–2498.

Sarle, W. S. et al. (1996). Stopped training and other remedies for overfitting. *Computing science and statistics*, 352–360.

Savijärvi, H. (1997). Diurnal winds around lake tanganyika. *Quarterly Journal of the Royal Meteorological Society*, *123*(540), 901–918.

Schmid, J. (2000). The seviri instrument. *Proceedings of the 2000 EUMETSAT meteorological satellite data user's conference, Bologna, Italy*, *29*.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85–117.

Schön, C., & Dittrich, J. (2019). Make thunderbolts less frightening–predicting extreme weather using deep learning. *arXiv preprint arXiv:1912.01277*.

Sedgwick, P. (2014). Spearman's rank correlation coefficient. *Bmj*, *349*.

Semazzi, F. (2011). Enhancing safety of navigation and efficient exploitation of natural resources over lake victoria and its basin by strengthening meteorological services on the lake. *North Carolina State University Climate Modeling Laboratory Tech. Rep*, *104*.

Semwal, V. B., Mondal, K., & Nandi, G. C. (2017). Robust and accurate feature selection for humanoid push recovery and classification: Deep learning approach. *Neural Computing and Applications*, *28*(3), 565–574.

Seto, K. C., Güneralp, B., & Hutyra, L. R. (2012). Global forecasts of urban expansion to 2030 and direct impacts on biodiversity and carbon pools. *Proceedings of the National Academy of Sciences*, *109*(40), 16083–16088.

Setvák, M., Bedka, K., Lindsey, D. T., Sokol, A., Charvát, Z., Št'ástka, J., & Wang, P. K. (2013). A-train observations of deep convective storm tops. *Atmospheric research*, *123*, 229–248.

Sezer, O. B., Gudelek, M. U., & Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing*, *90*, 106181.

Shaheen, F., Verma, B., & Asafuddoula, M. (2016). Impact of automatic feature extraction in deep learning architecture. *2016 International conference on digital image computing: techniques and applications (DICTA)*, 1–8.

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Sharma, S., & Sharma, S. (2017). Activation functions in neural networks. *Towards Data Science*, *6*(12), 310–316.

Sichangi, A. W., & Makokha, G. O. (2017). Monitoring water depth, surface area and volume changes in lake victoria: Integrating the bathymetry map and remote sensing data during 1993–2016. *Modeling Earth Systems and Environment*, *3*(2), 533–538.

Song, Y., Semazzi, F. H., Xie, L., & Ogallo, L. J. (2004). A coupled regional climate model for the lake victoria basin of east africa. *International Journal of Climatology: A Journal of the Royal Meteorological Society*, *24*(1), 57–75.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Stensrud, D. J. (2009). *Parameterization schemes: Keys to understanding numerical weather prediction models*. Cambridge University Press.

Stephens, G. L., L'Ecuyer, T., Forbes, R., Gettelmen, A., Golaz, J.-C., Bodas-Salcedo, A., Suzuki, K., Gabriel, P., & Haynes, J. (2010). Dreary state of precipitation in global models. *Journal of Geophysical Research: Atmospheres*, *115*(D24).

Sun, X., Xu, W., Jiang, H., & Wang, Q. (2020). A deep multitask learning approach for air quality prediction. *Annals of Operations Research*, 1–29.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, *4*(1), 1–103.

Tetko, I. V., Livingstone, D. J., & Luik, A. I. (1995). Neural network studies. 1. comparison of overfitting and overtraining. *Journal of chemical information and computer sciences*, *35*(5), 826–833.

Thiery, W., Davin, E. L., Panitz, H.-J., Demuzere, M., Lhermitte, S., & Van Lipzig, N. (2015). The impact of the african great lakes on the regional climate. *Journal of Climate*, *28*(10), 4061–4085.

Thiery, W., Davin, E. L., Seneviratne, S. I., Bedka, K., Lhermitte, S., & Van Lipzig, N. P. (2016). Hazardous thunderstorm intensification over lake victoria. *Nature communications*, *7*(1), 1–7.

Thiery, W., Gudmundsson, L., Bedka, K., Semazzi, F. H., Lhermitte, S., Willems, P., Van Lipzig, N. P., & Seneviratne, S. I. (2017). Early warnings of hazardous thunderstorms over lake victoria. *Environmental Research Letters*, *12*(7), 074012.

Tribbia, J., & Baumhefner, D. (2004). Scale interactions and atmospheric predictability: An updated perspective. *Monthly weather review*, *132*(3), 703–713.

Van de Walle, J., Thiery, W., Brousse, O., Souverijns, N., Demuzere, M., & van Lipzig, N. P. (2020). A convection-permitting model for the lake victoria basin: Evaluation and insight into the mesoscale versus synoptic atmospheric dynamics. *Climate Dynamics*, *54*(3), 1779–1799.

Vanderkelen, I., Van Lipzig, N. P., & Thiery, W. (2018). Modelling the water balance of lake victoria (east africa)–part 1: Observational analysis. *Hydrology and Earth System Sciences*, *22*(10), 5509–5525.

Verschuren, D., Johnson, T. C., Kling, H. J., Edgington, D. N., Leavitt, P. R., Brown, E. T., Talbot, M. R., & Hecky, R. E. (2002). History and timing of human impact on lake victoria, east africa. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, *269*(1488), 289–294.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Virts, K. S., & Goodman, S. J. (2020). Prolific lightning and thunderstorm initiation over the lake victoria basin in east africa. *Monthly Weather Review*, *148*(5), 1971–1985.

Wang, C.-C., Kirshbaum, D. J., & Sills, D. M. (2019). Convection initiation aided by lake-breeze convergence over the niagara peninsula. *Monthly Weather Review*, *147*(11), 3955–3979.

Wang, L., Zhang, Z., Zhang, X., Zhou, X., Wang, P., & Zheng, Y. (2020). A deep-forest based approach for detecting fraudulent online transaction. *Advances in Computers*.

Wang, Y. E., Wei, G.-Y., & Brooks, D. (2019). Benchmarking tpu, gpu, and cpu platforms for deep learning. *arXiv preprint arXiv:1907.10701*.

Watkiss, P., Powell, R., Hunt, A., & Cimato, F. (2020). *The socio-economic benefits of the highway project*.

Wiering, M. A., & Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, *12*(3).

Wilks, D. (2011). Chapter 8 - forecast verification. In D. S. Wilks (Ed.), *Statistical methods in the atmospheric sciences* (pp. 301–394). Academic Press. https://doi.org/https://doi.org/10.1016/B978-0-12-385022-5.00008-7

Williams, K., Chamberlain, J., Buontempo, C., & Bain, C. (2015). Regional climate model performance in the lake victoria basin. *Climate Dynamics*, *44*(5-6), 1699–1713.

Wilson, D. R., & Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural networks*, *16*(10), 1429–1451.

Xu, Y., Mo, T., Feng, Q., Zhong, P., Lai, M., Eric, I., & Chang, C. (2014). Deep learning of feature representation with multiple instance learning for medical image analysis. *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 1626–1630.

Yost, C. R., Bedka, K. M., Minnis, P., Nguyen, L., Strapp, J. W., Palikonda, R., Khlopenkov, K., Spangenberg, D., Smith Jr, W. L., Protat, A., et al. (2018). A prototype method for diagnosing high ice water content probability using satellite imager data. *Atmospheric measurement techniques*, *11*(3), 1615–1637.

Yu, X.-H., & Chen, G.-A. (1997). Efficient backpropagation learning using optimal learning rate and momentum. *Neural networks*, *10*(3), 517–527.

Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhou, C., Sun, C., Liu, Z., & Lau, F. (2015). A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*.

Zhou, K., Zheng, Y., Dong, W., & Wang, T. (2020). A deep learning network for cloud-to-ground lightning nowcasting with multisource data. *Journal of Atmospheric and Oceanic Technology*, *37*(5), 927–942.

Zhou, K., Zheng, Y., Li, B., Dong, W., & Zhang, X. (2019). Forecasting different types of convective weather: A deep learning approach. *Journal of Meteorological Research*, *33*(5), 797–809.

Zhou, Y., Li, G., & Tan, Y. (2015). Computational aesthetics of photos quality assessment and classification based on artificial neural network with deep learning methods. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, *8*(7), 273–282.

Zhou, Z.-H. (2019). *Ensemble methods: Foundations and algorithms*. Chapman; Hall/CRC.

Zou, Q., Ni, L., Zhang, T., & Wang, Q. (2015). Deep learning based feature selection for remote sensing scene classification. *IEEE Geoscience and Remote Sensing Letters*, *12*(11), 2321–2325.

Appendix

6.1 Tuple Generator

```
import os
import subprocess
from multiprocessing import Pool
import netCDF4 as nc

timestamps = ['00', '15', '30', '45']

afternoon_timestamps = ['0' + str(x) + y for x in range(4, 10) for y in timestamps] \
                       + [str(x) + y for x in range(10, 18) for y in timestamps]

evening_timestamps = [str(x) + y for x in range(21, 24) for y in timestamps]
night_timestamps = ['0' + str(x) + y for x in range(9) for y in timestamps]
evening_night_timestamps = evening_timestamps + night_timestamps

DATAFOLDER = r"/theia/data/brussel/vo/000/bvo00012/data/dataset/nasalarc_otdetection_lakevictoria"
RESULTFOLDER = r"/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Dataset/tuples"
directory_list = sorted(os.listdir(f"{DATAFOLDER}/"))

def get_data(directory):
    year, day = int(directory[-8:-3]), int(directory[-3:])
    index = directory_list.index(directory)
    next_directory = directory_list[index + 1]
    next_year, next_day = int(next_directory[-8:-3]), int(next_directory[-3:])
    # If the next folder does not contain the data of the next day of the same year
    if day + 1 != next_day:
        # If (it is a non-leap year AND the day is not #365) OR (it is a leap year AND the day is not #366)
        # OR the next year is not the current + 1 OR the next day is not #1, then we'll skip the day
        if ((year % 4) and (day != 365)) or (not (year % 4) and (day != 366)) \
                or (year + 1 != next_year) or (next_day != 1):
            print(f'{year}{day} has no adjacent folder')
            return

    # Generate the names of the afternoon files
    afternoon_files = [f"{DATAFOLDER}/{directory}/{filename}" for filename in
                       sorted(os.listdir(f"{DATAFOLDER}/{directory}/")) if filename[-7:-3] in afternoon_timestamps]

    # Generate the names of the evening files
    night_files = [f"{DATAFOLDER}/{directory}/{filename}" for filename in
                   sorted(os.listdir(f"{DATAFOLDER}/{directory}/"))
                   if filename[-7:-3] in evening_timestamps]

    # Add the names of the night files of the next day to the evening files
    night_files.extend([f"{DATAFOLDER}/{next_directory}/{filename}" for filename in
                        sorted(os.listdir(f"{DATAFOLDER}/{next_directory}/")) if filename[-7:-3] in night_timestamps])

    for file_list in (afternoon_files, night_files):
        for idx, filename in enumerate(file_list):
            duplicate_list = [other_filename for other_filename in file_list[idx + 1::] if
                              other_filename[-7:-3] == filename[-7:-3]]
            for duplicate in duplicate_list:
                file_list.remove(duplicate)
```

```
# Remove afternoon slices with an invalid shape (rare, ~180 slices in total)
for f in afternoon_files:
    dataset = nc.Dataset(f)
    lon = dataset['longitude'].size
    lat = dataset['latitude'].size
    if (lon, lat) != (840, 840):
        afternoon_files.remove(f)

# Find which time slices are missing:
afternoon_slices_present = [filename[-7:-3] for filename in afternoon_files]
afternoon_slices_bool = [int(time_slice in afternoon_slices_present) for time_slice in afternoon_timestamp

if not any(afternoon_slices_bool[0:2]) or not any(afternoon_slices_bool[-2::]) or '000' in ''.join(
        str(x) for x in afternoon_slices_bool):
    print(f'{year}{day} has too many consecutively missing files in the afternoon')
    return

# Remove night slices with an invalid shape (rare, ~180 slices in total)
for f in night_files:
    dataset = nc.Dataset(f)
    lon = dataset['longitude'].size
    lat = dataset['latitude'].size
    if (lon, lat) != (840, 840):
        night_files.remove(f)

# Find which time slices are missing:
night_slices_present = [filename[-7:-3] for filename in night_files]
night_slices_bool = [int(time_slice in night_slices_present) for time_slice in evening_night_timestamps]

if not any(night_slices_bool[0:2]) or not any(night_slices_bool[-2::]) or '000' in ''.join(
        str(x) for x in night_slices_bool):
    # If the first or last two are missing or there are three consecutive files missing then we won't use
    print(f'{year}{day} has too many consecutively missing files in the night')
    return

# For the missing files: copy the slice before it if that file wasn't missing itself otherwise the slice a
previous_val = False
for idx, bool_val in enumerate(afternoon_slices_bool):
    if bool_val:
        previous_val = True
    else:
        # -int(previous_val) equals 0 if previous_val is False and equals -1 if previous_val is True
        afternoon_files.insert(idx, afternoon_files[idx - int(previous_val)])
        previous_val = False

previous_val = False
for idx, bool_val in enumerate(night_slices_bool):
    if bool_val:
        previous_val = True
    else:
        # Equals -1 (previous) if previous_val is True and equals +1 if previous_val is False
        night_files.insert(idx, night_files[idx - int(previous_val)])
        previous_val = False

# Names for the two merged files
afternoon_result = f"{RESULTFOLDER}/{directory[-8:]}.afternoon.nc"
night_result = f"{RESULTFOLDER}/{directory[-8:]}.night.nc"

# Concatenate the afternoon_files
afternoon_code = subprocess.call(f"ncecat -O --no_tmp_fl {' '.join(afternoon_files)} {afternoon_result}",
                                 shell=True)

# afternoon_code will be 0 (False) if everything went well, otherwise nco returned an error code
if afternoon_code:
    print(f"{year}{day} returned an error for the afternoon")
    # Remove the erroneous afternoon
    subprocess.call(f"rm {afternoon_result}", shell=True)
    # No need to continue to the night files if the afternoon files failed
    return

# Concatenate the night files
night_code = subprocess.call(f"ncecat -O --no_tmp_fl {' '.join(night_files)} {night_result}", shell=True)
```

```
    # night_code will be 0 (False) if everything went well, otherwise nco returned an error code
    if night_code:
        print(f"{year}{day} returned an error for the night")
        # Remove the erroneous night result and its afternoon as well
        subprocess.call(f"rm {afternoon_result} {night_result}", shell=True)


if __name__ == '__main__':
    pool = Pool()
    pool.map(get_data, directory_list[:-1])
    pool.close()
```

6.2 Tar Generator

```
import os
import webdataset as wds

dataset_folder = "/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Dataset/"
MAXCOUNT = 10  # Maximum number of samples per shard

def yield_samples():
    for root, dirs, files in os.walk(dataset_folder + "tuples/", topdown=False):
        for fname in files:
            if 'afternoon' in fname:  # For each afternoon file
                afternoon_fpath = os.path.join(root, fname)  # Open the afternoon
                night_fpath = os.path.join(root, fname.replace('afternoon', 'night'))
                with open(afternoon_fpath, 'rb') as afternoon:
                    afternoon_binary = afternoon.read()  # Store the binary afternoon data
                with open(night_fpath, 'rb') as night:
                    night_binary = night.read()  # Store the binary night data
                sample = {
                    "__key__": os.path.splitext(fname)[0],
                    "afternoon": afternoon_binary,
                    "night": night_binary
                }
                yield sample


with wds.ShardWriter(dataset_folder + "shards/shard-%03d.tar", maxcount=MAXCOUNT) as sink:
    for sample in yield_samples():
        sink.write(sample)
```

6.3 OT Probability Extractor

```
import os
from multiprocessing import Pool
import nctoolkit as nc

nc.options(parallel=True)
nc.options(cores=10)

data_folder = "/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Dataset/tuples/"
result_folder = "/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Dataset/Products/OTsFull/"
# Sorting is not required but allows to estimate the progress during execution by viewing the result_folder
filenames = sorted(os.listdir(data_folder))

def getField(filename):
    ds = nc.open_data(data_folder + filename)  # Open the netCDF4 dataset
    ds.select(variables='ot_probability')  # Select the ot_probability variable
    ds.reduce_dims()  # Reduce redundant dimensions, equals .squeeze() in numpy/torch
    ds.to_nc(result_folder + filename)  # Save the result


if __name__ == '__main__':
    pool = Pool()
    pool.map(getField, filenames)
    pool.close()
```

6.4 DataLoader

```
import netCDF4 as nc
import numpy as np
import pytorch_lightning as pl
import torch
import webdataset as wds
from skimage.measure import block_reduce
from torch.utils.data import DataLoader, Dataset

if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')


def decode_netcdf(sample):
    '''
    Unpacks two netCDF4 files from a byte stream
    '''
    return nc.Dataset('in-mem-file', mode='r', memory=sample['afternoon.afternoon']), nc.Dataset('in-mem-file'
                                                                                                 mode='r',
                                                                                                 memory=sample
                                                                                                     'afternoo


class NetcdfDataset(pl.LightningDataModule):
    def __init__(self, batch_size, bucket, num_workers=0, input_transforms=None, label_transforms=None):
        super().__init__(self)
        self.batch_size, self.num_workers, self.bucket = batch_size, num_workers, bucket
        self.input_transforms, self.label_transforms = input_transforms, label_transforms

        num_shards = len(bucket)
        shard_idx_list = np.arange(num_shards)
        np.random.shuffle(shard_idx_list)
        train_val_test_idcs = np.split(shard_idx_list, [int(num_shards * 0.75), int(
            num_shards * 0.85)])  # Split train (75%), val (10%) and test (15%) set
        self.train_urls, self.val_urls, self.test_urls = [[bucket[idx] for idx in idx_list] for idx_list in
                                                          (train_val_test_idcs)]

    def make_loader(self, urls):
        dataset = (wds.WebDataset(urls)
                   .map(decode_netcdf)
                   .map_tuple(self.input_transforms, self.label_transforms)
                   .batched(self.batch_size, partial=False)
                   )
        loader = wds.WebLoader(dataset, batch_size=None, shuffle=True, num_workers=self.num_workers)
        return loader

    def train_dataloader(self):
        return self.make_loader(urls=self.train_urls)

    def val_dataloader(self):
        return self.make_loader(urls=self.val_urls)

    def test_dataloader(self):
        return self.make_loader(urls=self.test_urls)

    def all_loader(self):
        return self.make_loader(urls=self.bucket)


class TensorDataset(pl.LightningDataModule):
    def __init__(self, batch_size, input_tensor, label_tensor, num_workers=0, input_transforms=None,
                 label_transforms=None, collate_fn=None):
        super().__init__(self)
        self.batch_size, self.collate_fn, self.num_workers = batch_size, collate_fn, num_workers
        self.input_transforms, self.label_transforms = input_transforms, label_transforms
        self.input_tensor, self.label_tensor = input_tensor, label_tensor

        num_idcs = len(label_tensor)
        idx_list = np.arange(num_idcs)
```

```
        np.random.shuffle(idx_list)
        train_val_test_idcs = np.split(idx_list, [int(num_idcs * 0.75), int(
            num_idcs * 0.85)])  # Split train (75%), val (10%) and test (15%) set
        self.train_idcs, self.val_idcs, self.test_idcs = [[idx_list[idx] for idx in idcs] for idcs in
                                                          (train_val_test_idcs)]
        self.all_idcs = idx_list

    def make_loader(self, idcs):
        data_set = My_Dataset(self.input_tensor[idcs], self.label_tensor[idcs], self.input_transforms,
                              self.label_transforms)
        return DataLoader(data_set, batch_size=self.batch_size, shuffle=True, drop_last=True,
                          collate_fn=self.collate_fn, num_workers=self.num_workers)

    def train_dataloader(self):
        return self.make_loader(idcs=self.train_idcs)

    def val_dataloader(self):
        return self.make_loader(idcs=self.val_idcs)

    def test_dataloader(self):
        return self.make_loader(idcs=self.test_idcs)

    def all_loader(self):
        return self.make_loader(idcs=self.all_idcs)


class My_Dataset(Dataset):
    def __init__(self, input_tensor, label_tensor, input_transforms=None, label_transforms=None):
        super().__init__()
        self.input_tensor, self.label_tensor = input_tensor, label_tensor
        self.input_transforms, self.label_transforms = input_transforms, label_transforms

    def __len__(self):
        return len(self.label_tensor)

    def __getitem__(self, idx):
        x, y = self.input_tensor[idx], self.label_tensor[idx]
        if self.input_transforms:
            x = self.input_transforms(x)
        if self.label_transforms:
            y = self.label_transforms(y)
        return x, y


class CropFieldsAndSize:
    """
    Need to perform both operations in one action as cropping in size requires the 'longitude' and 'latitude' attribut
    of the netcdf4 dataset, cropping by fields also requires the object to remain netcdf4 dataset.
    Alternatively, a new dataset could be created with the desired fields + longitude and latitude and then pass this
    to a separate crop function for size
    """

    def __init__(self, fields, max_lon=False, min_lat=False, height=840, width=840):
        if type(fields) == str:
            fields = [fields]
        self.fields, self.max_lon, self.min_lat, self.height, self.width = fields, max_lon, min_lat, height, width

    def __call__(self, netcdf4_dataset):
        check = all([type(x) in (int, float) for x in (self.min_lat, self.max_lon)])
        if check:
            lon_idx = np.argmin(np.abs(netcdf4_dataset.variables['longitude'][:] - self.max_lon))
            lat_idx = np.argmin(np.abs(netcdf4_dataset.variables['latitude'][:] - self.min_lat))
        result = torch.zeros((len(self.fields), netcdf4_dataset.dimensions['record'].size, self.height, self.width))
        for idx, field in enumerate(self.fields):
            masked_array = netcdf4_dataset[field][:].squeeze()
            if check:
                masked_array = masked_array[:, lat_idx:lat_idx + self.height, lon_idx:lon_idx + self.width]
            result[idx] = torch.FloatTensor(masked_array.filled())
        return result.squeeze()


class SumDim:
    def __init__(self, dim=0):
```

```
        self.dim = dim

    def __call__(self, sample):
        return torch.sum(sample, dim=self.dim)


class MaxMapsIntoSingleMap:
    def __call__(self, sample):
        return torch.max(sample, dim=0)


class CombineProbabilityMapsIntoSingleMap:
    def __call__(self, sample):
        no_storm_prob = torch.ones_like(sample) - sample
        no_storm_prob = torch.prod(no_storm_prob, dim=0)
        return torch.ones_like(no_storm_prob) - no_storm_prob


class Pool:
    def __init__(self, pooled=0):
        self.pooled = pooled

    def __call__(self, sample):
        if self.pooled:
            if len(sample.shape) == 2:
                sample = torch.Tensor(block_reduce(sample, (2 ** self.pooled, 2 ** self.pooled), np.average))
            elif len(sample.shape) == 3:
                sample = torch.Tensor(block_reduce(sample, (1, 2 ** self.pooled, 2 ** self.pooled), np.average
        return sample


class Mask:
    def __init__(self, mask, pooled=0):
        self.pooled = pooled
        self.mask = mask
        if self.pooled:
            self.mask = torch.Tensor(block_reduce(self.mask, (2 ** self.pooled, 2 ** self.pooled), np.max)).bo

    def __call__(self, sample):
        if len(sample.shape) == 3:
            channels = sample.shape[0]
            return torch.masked_select(sample, self.mask).view(channels, -1)
        return sample[self.mask]


class Reconstruct2D:
    def __init__(self, mask):
        self.mask = mask

    def __call__(self, sample):
        return torch.zeros_like(self.mask, dtype=sample.dtype).masked_scatter_(self.mask, sample)


class SumAll:
    def __call__(self, sample):
        return torch.sum(sample)


class BinarizeValues:
    def __call__(self, sample):
        return torch.where(sample > 1, torch.ones_like(sample), torch.zeros_like(sample))


class ToTensor:
    def __init__(self, field):
        self.field = field

    def __call__(self, sample):
        return torch.FloatTensor(sample[self.field][:].filled().squeeze())


class SelectTimes:
    def __init__(self, start, end=False):
```

```
        self.start, self.end = start, end

    def __call__(self, sample):
        if self.end:
            return sample[self.start:self.end]
        return sample[self.start::]


class ToExtreme:
    def __call__(self, sample):
        return torch.FloatTensor([int(float(sample) > 5379)])


class ToLong:
    def __call__(self, sample):
        return sample.long()


class ToFloat:
    def __call__(self, sample):
        return sample.float()


class ApplyFunction:
    def __init__(self, function):
        self.function = function

    def __call__(self, sample):
        return torch.Tensor(self.function(sample))
```

6.5 Classifier

```
import pytorch_lightning as pl
import torch
from torch import nn

class Classifier(pl.LightningModule):
    def __init__(self, model, lr=1e-3, optimizer=torch.optim.Adam, optimizer_params={}, scheduler=None,
                 scheduler_params={}, monitor_value=None, loss_function=nn.CrossEntropyLoss):
        super().__init__()
        self.model = model
        self.learning_rate = lr
        self.optimizer = optimizer
        self.optimizer_params = optimizer_params
        self.scheduler = scheduler
        self.scheduler_params = scheduler_params
        self.monitor_value = monitor_value
        self.loss_function = loss_function

    @staticmethod
    def binary_accuracy(output, batch_y):
        prediction = output // 0.5
        idcs_0 = torch.nonzero(batch_y == 0, as_tuple=True)
        total_0 = prediction[idcs_0].numel()
        # count_nonzero not available in pytorch 1.6.0
        correct_0 = int(torch.where(prediction[idcs_0] == 0, torch.ones_like(prediction[idcs_0]),
                                    torch.zeros_like(prediction[idcs_0])).sum())
        idcs_1 = torch.nonzero(batch_y == 1, as_tuple=True)
        total_1 = prediction[idcs_1].numel()
        # count_nonzero not available in pytorch 1.6.0
        correct_1 = int(torch.where(prediction[idcs_1] == 1, torch.ones_like(prediction[idcs_1]),
                                    torch.zeros_like(prediction[idcs_1])).sum())

        return {'correct_0': correct_0, 'total_0': total_0, 'correct_1': correct_1, 'total_1': total_1}

    def training_step(self, batch, batch_index):
        '''
        This function takes a batch of samples and batch_index as input.
        The batch is split into its X (input) and y (output) values,
        the X will be fed to the network and its output is stored as 'output'.
        Then the loss function will be applied on the output and y, and the loss and accuracy will be logged.
        This function may not call 'validation_step' even though both functions perform similar operations.
        'validation_step' is wrapped with a 'torch.set_grad_enabled(False)', so gradients will not be calculat
        '''
        batch_x, batch_y = batch
        output = self.model.forward(batch_x)  # Forward pass
        loss = self.loss_function(output, batch_y)  # Calculate loss
        bin_acc = self.binary_accuracy(output, batch_y)
        self.log_dict({'Training␣loss': loss, 'correct_0': bin_acc['correct_0'], 'total_0': bin_acc['total_0']
                       'correct_1': bin_acc['correct_1'], 'total_1': bin_acc['total_1']}, logger=True, on_step
        return {"loss": loss}

    def training_epoch_end(self, training_step_outputs):
        '''
        Training loss and accuracy is already monitored at each step, so we won't do anything specifically aft
        '''
        pass

    def validation_step(self, batch, batch_index):
        '''
        This function operates in a similar way as 'training_step', it is wrapped in a 'torch.set_grad_enabled
        The only difference is that we won't log the validation loss and accuracy every step, as we only want
        '''
        batch_x, batch_y = batch
        output = self.model.forward(batch_x)
        loss = self.loss_function(output, batch_y)  # Calculate loss
        bin_acc = self.binary_accuracy(output, batch_y)
        return {"loss": loss, 'correct_0': bin_acc['correct_0'], 'total_0': bin_acc['total_0'],
                'correct_1': bin_acc['correct_1'], 'total_1': bin_acc['total_1']}

    def validation_epoch_end(self, validation_step_outputs):
        # log the accuracy and loss after one epoch
```

```
        self.log_dict(self.end_epoch(validation_step_outputs, desc='Validation'), logger=True)

    def test_step(self, batch, batch_index):
        # At each batch testing step we'll calculate the loss
        return self.validation_step(batch, batch_index)

    def test_epoch_end(self, test_step_outputs):
        # log the accuracy and loss after one test epoch (so usually only once)
        self.log_dict(self.end_epoch(test_step_outputs, desc='Test'), logger=True)

    def end_epoch(self, step_outputs, desc=''):
        # Calculate the accuracy and loss of an epoch
        loss = round(torch.stack([step['loss'] for step in step_outputs]).mean().item(), 3)
        correct_0 = sum([step["correct_0"] for step in step_outputs])
        total_0 = sum([step["total_0"] for step in step_outputs])
        correct_1 = sum([step["correct_1"] for step in step_outputs])
        total_1 = sum([step["total_1"] for step in step_outputs])
        return {f'{desc} loss': loss, f'{desc} accuracy 0': round(correct_0 / total_0, 3),
                f'{desc} correct 1': correct_1, f'{desc} total 1': total_1}

    def configure_optimizers(self):
        optimizer = self.optimizer(self.parameters(), lr=self.learning_rate, **self.optimizer_params)
        if self.scheduler:
            scheduler = self.scheduler(optimizer, **self.scheduler_params)
            return {"optimizer": optimizer, "lr_scheduler": scheduler, "monitor": self.monitor_value}
        return {"optimizer": optimizer}
```

6.6 CNN

```
import pytorch_lightning as pl
import torch.nn as nn


class CNN(pl.LightningModule):
    '''
    This class will create the structure of the CNNs, it can take a wide range of optional parameters.
    The only mandatory parameter is the input_size and output_size.
    Every parameter can be defined as a singular value, this will then be used at every step.
    It can also be defined as a list/tuple and then it can have separate values for each layer/step.
    Note that the length should match the amount of convolutional and linear layers if it is passed as list or
    '''

    def __init__(self, input_size, output_size, activation=nn.ReLU,
                 size_2d=[], kernel_2d=4, stride_2d=1, pad_2d=0, drop_2d=0, groups_2d=1, dilation_2d=1,
                 pool_2d=False, pool_2d_size=0, pool_2d_stride=2, pool_2d_pad=0, batchnorm_2d=False,
                 size_1d=[], drop_1d=0, batchnorm_1d=False, final_activation=None):
        super().__init__()

        self.output_size, self.activation, \
        self.size_2d, self.kernel_2d, self.stride_2d, self.pad_2d, self.drop_2d, self.groups_2d, self.dilation
        self.pool_2d, self.pool_2d_size, self.pool_2d_stride, self.pool_2d_pad, self.batchnorm_2d, \
        self.size_1d, self.drop_1d, self.batchnorm_1d, self.final_activation \
            = output_size, activation, \
              size_2d, kernel_2d, stride_2d, pad_2d, drop_2d, groups_2d, dilation_2d, \
              pool_2d, pool_2d_size, pool_2d_stride, pool_2d_pad, batchnorm_2d, \
              size_1d, drop_1d, batchnorm_1d, final_activation

        if len(input_size) == 1:
            self.height, self.width, self.channels = 1, input_size[0], 1
        elif len(input_size) == 2:
            self.height, self.width, self.channels = input_size[0], input_size[1], 1
        elif len(input_size) == 3:
            self.height, self.width, self.channels = input_size[1], input_size[2], input_size[0]

        # This will check whether the length of the input parameters corresponds to the amount of convolutiona
        self.validate_input_parameters()
        self.build_2d_layers()  # Build the 2d convolutional layers
        self.build_1d_layers()  # Construct the linear layers

    def forward(self, x):
        """
        Defines a forward pass of the CNN
        """
        x = self.layers_2d(x)
        x = x.view(-1, self.conv_output_size)  # Flatten the output of the convolutional layers
        x = self.layers_1d(x)
        return x.view(-1, self.output_size).float()

    def validate_input_parameters(self):
        """
        Validates the length and type of the input parameters
        """

        if type(self.size_2d) == int:
            self.size_2d = [self.size_2d]
        for size in self.size_2d:
            assert size > 0, 'Invalid␣size_2d,␣must␣be␣strictly␣positive'
        length_2d = len(self.size_2d)

        if type(self.kernel_2d) == int:
            self.kernel_2d = [self.kernel_2d] * length_2d
        for kernel_2d in self.kernel_2d:
            assert kernel_2d > 0, 'Invalid␣kernel_2d,␣must␣be␣strictly␣positive'

        if type(self.stride_2d) == int:
            self.stride_2d = [self.stride_2d] * length_2d
        for stride_2d in self.stride_2d:
            assert stride_2d > 0, 'Invalid␣stride_2d,␣must␣be␣strictly␣positive'
```

```
if type(self.pad_2d) == int:
    self.pad_2d = [self.pad_2d] * length_2d
for pad_2d in self.pad_2d:
    assert pad_2d >= 0, 'Invalid␣pad_2d,␣may␣not␣be␣negative'

if type(self.dilation_2d) == int:
    self.dilation_2d = [self.dilation_2d] * length_2d
for dilation_2d in self.dilation_2d:
    assert dilation_2d > 0, 'Invalid␣dilation_2d,␣must␣be␣strictly␣positive'

if type(self.groups_2d) == int:
    self.groups_2d = [self.groups_2d] * length_2d
for groups_2d in self.groups_2d:
    assert groups_2d > 0, 'Invalid␣groups_2d,␣must␣be␣strictly␣positive'

if callable(self.pool_2d):
    self.pool_2d = [self.pool_2d] * length_2d
for pool_2d in self.pool_2d:
    assert callable(
        pool_2d) or pool_2d == None, 'Invalid␣pool_2d,␣should␣be␣a␣2D␣non-adaptive␣torch.nn␣pool␣callable␣or␣N

if type(self.pool_2d_size) == int:
    self.pool_2d_size = [self.pool_2d_size] * length_2d
for pool_2d_size in self.pool_2d_size:
    assert pool_2d_size > 0, 'Invalid␣pool_2d_size,␣must␣be␣strictly␣positive'

if type(self.pool_2d_stride) == int:
    self.pool_2d_stride = [self.pool_2d_stride] * length_2d
for pool_2d_stride in self.pool_2d_stride:
    assert pool_2d_stride > 0, 'Invalid␣pool_2d_stride,␣must␣be␣strictly␣positive'

if type(self.pool_2d_pad) == int:
    self.pool_2d_pad = [self.pool_2d_pad] * length_2d
for pool_2d_pad in self.pool_2d_pad:
    assert pool_2d_pad >= 0, 'Invalid␣pool_2d_pad,␣may␣not␣be␣negative'

if type(self.batchnorm_2d) == bool:
    self.batchnorm_2d = [self.batchnorm_2d] * length_2d

if type(self.drop_2d) in (float, int):
    self.drop_2d = [self.drop_2d] * length_2d
for drop_2d in self.drop_2d:
    assert drop_2d >= 0 and drop_2d < 1, 'Invalid␣drop_2d,␣must␣be␣between␣0␣and␣1'

for param in (
        self.kernel_2d, self.stride_2d, self.pad_2d, self.pool_2d, self.pool_2d_size, self.pool_2d_stride,
        self.pool_2d_pad,
        self.batchnorm_2d, self.drop_2d):
    assert len(
        param) == length_2d, f'Wrong␣size␣of␣2d␣convolutional␣parameter,␣expected␣{length_2d},␣got␣{len(param)

################################################################################################

if type(self.size_1d) == int:
    self.size_1d = [self.size_1d]
for size in self.size_1d:
    assert size > 0, 'Invalid␣size_1d,␣must␣be␣strictly␣positive'
length_1d = len(self.size_1d)

if type(self.batchnorm_1d) == bool:
    self.batchnorm_1d = [self.batchnorm_1d] * length_1d

if type(self.drop_1d) in (float, int):
    self.drop_1d = [self.drop_1d] * length_1d
for drop_1d in self.drop_1d:
    assert drop_1d >= 0 and drop_1d < 1, 'Invalid␣drop_1d,␣must␣be␣between␣0␣and␣1'
assert len(self.drop_1d) == length_1d, 'Wrong␣length␣of␣drop_1d␣parameter'

################################################################################################

assert callable(self.activation), 'activation␣function␣not␣callable'
if self.final_activation:
    assert callable(self.final_activation), 'final_activation␣function␣not␣callable'
```

```
    def build_2d_layers(self):
        """
        Builds the 2D convolutional layers with the specified parameters
        """
        self.layers_2d = []

        for index, out_size in enumerate(self.size_2d):  # Here the 2D convolutional layers are generated
            self.layers_2d.append(
                nn.Conv2d(in_channels=self.channels, out_channels=out_size, kernel_size=self.kernel_2d[index],
                          stride=self.stride_2d[index], padding=self.pad_2d[index], groups=self.groups_2d[inde
                          dilation=self.dilation_2d[index]))
            assert self.kernel_2d[index] <= self.height, f'Kernel␣size␣too␣big␣in␣2D␣convolutional␣layer␣{inde
            assert self.kernel_2d[index] <= self.width, f'Kernel␣size␣too␣big␣in␣2D␣convolutional␣layer␣{index
            self.height = (self.height + 2 * self.pad_2d[index] - (self.kernel_2d[index] - 1) - 1) // self.str
                index] + 1  # Update self.height

            self.width = (self.width + 2 * self.pad_2d[index] - (self.kernel_2d[index] - 1) - 1) // self.strid
                index] + 1  # Update self.height

            if self.batchnorm_2d[index]:
                self.layers_2d.append(nn.BatchNorm2d(self.size_2d[index]))

            self.layers_2d.append(self.activation())  # Add activation function

            if self.pool_2d[index]:
                pool = self.pool_2d[index]
                self.layers_2d.append(
                    pool(kernel_size=self.pool_2d_size[index], stride=self.pool_2d_stride[index],
                         padding=self.pool_2d_pad[index]))
                self.height = (self.height + 2 * self.pool_2d_pad[index] - (self.pool_2d_size[index] - 1) - 1)
                              self.pool_2d_stride[index] + 1  # Update self.height

                self.width = (self.width + 2 * self.pool_2d_pad[index] - (self.pool_2d_size[index] - 1) - 1) /
                             self.pool_2d_stride[index] + 1  # Update self.width

            if self.drop_2d[index]:
                self.layers_2d.append(nn.Dropout2d(self.drop_2d[index]))

            self.channels = out_size  # We need this in order to construct the next convolutional layer

        # This will be needed to construct the first linear layer and to flatten the data between the convolut
        self.conv_output_size = self.channels * self.height * self.width
        self.layers_2d = nn.Sequential(*self.layers_2d)

    def build_1d_layers(self):
        """
        Builds the linear layers with the specified parameters
        """
        previous_lin_size = self.conv_output_size
        self.layers_1d = []

        for index, out_size in enumerate(self.size_1d):
            self.layers_1d.append(nn.Linear(previous_lin_size, out_size))

            if self.batchnorm_1d[index]:
                self.layers_1d.append(nn.BatchNorm1d(out_size))

            self.layers_1d.append(self.activation())

            if self.drop_1d[index]:
                self.layers_1d.append(nn.Dropout(self.drop_1d[index]))

            previous_lin_size = out_size

        self.layers_1d.append(
            nn.Linear(previous_lin_size, self.output_size))  # We always have at least one linear layer
        if callable(self.final_activation):
            self.layers_1d.append(self.final_activation())
        self.layers_1d = nn.Sequential(*self.layers_1d)
```

6.7 Trainer

```
import datetime
import os

import numpy as np
import pytorch_lightning as pl
import torch
from CNN import CNN
from Classifier import *
from DataLoader import *
from pytorch_lightning.loggers import TensorBoardLogger
from skimage.measure import block_reduce
from torch import nn
from torchvision import transforms

LOG_LOCATION = r"/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Logs"
SAVE_LOCATION = r"/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Models"

shards = "/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Dataset/OT_Probability_shards/"
shard_bucket = [shards + filename for filename in os.listdir(shards)]

INPUT_TENSOR = torch.load(
    f"/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Dataset/Products/OT_Probability_{TEXT}.pt")
LABEL_TENSOR = torch.load(
    "/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Dataset/Products/OT_Probability_night.pt")

BATCH_SIZE = 64
NUM_WORKERS = 3
GPUS = 1
EPOCHS = 20
POOL = 5

INPUT_FIELD = 'ot_probability'
TARGET_FIELDS = 'ot_probability'

DAY_MASK = torch.load(
    "/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Dataset/Products/Day_Mask.pt").bool()
DAY_MASK_SIZE = int(torch.sum(torch.Tensor(block_reduce(DAY_MASK, (2 ** POOL, 2 ** POOL), np.max))))

LAKE_MASK = torch.load(r'/theia/data/brussel/vo/000/bvo00012/vsc10262/Thesis_Seppe_Lampe/Lake_Victoria_Mask.pt').bool(
LAKE_MASK_SIZE = int(torch.sum(LAKE_MASK))

CNN_PARAMS = {'input_size': [DAY_MASK_SIZE], 'output_size': 1, 'activation': nn.Sigmoid,
              'size_2d': [], 'kernel_2d': [], 'stride_2d': [], 'pad_2d': [], 'drop_2d': [], 'groups_2d': [],
              'dilation_2d': [],
              'pool_2d': [], 'pool_2d_size': [], 'pool_2d_stride': [], 'pool_2d_pad': [], 'batchnorm_2d': [],
              'size_1d': [36, 36, 18], 'batchnorm_1d': True, 'drop_1d': 0.2, 'final_activation': nn.Sigmoid}

input_transforms = transforms.Compose([
    Reconstruct2D(mask=DAY_MASK),
    Pool(pooled=POOL),
    Mask(mask=DAY_MASK, pooled=POOL)
])

label_transforms = transforms.Compose([
    ToExtreme()
])

if __name__ == '__main__':
    dataset = TensorDataset(batch_size=BATCH_SIZE, num_workers=NUM_WORKERS,
                            input_tensor=INPUT_TENSOR, label_tensor=LABEL_TENSOR,
                            input_transforms=input_transforms, label_transforms=label_transforms, collate_fn=None)
    net = CNN(**CNN_PARAMS)
    classifier = Classifier(model=net, optimizer=torch.optim.Adam, loss_function=my_loss)
    logger = TensorBoardLogger(save_dir=LOG_LOCATION, default_hp_metric=False)
    # Create a trainer which will use 1 GPU, which will run for X epochs and we'll allow it to find the best learning
    trainer = pl.Trainer(gpus=GPUS, logger=logger, max_epochs=EPOCHS, progress_bar_refresh_rate=0, auto_lr_find=True,
                         checkpoint_callback=False)  # checkpoint_callback enables/disables checpoint saving every epo

    # Train the model with the dataset
    trainer.fit(classifier, dataset)
```

```
# Test the model
test_params = trainer.test(classifier)
date_time = datetime.datetime.now().strftime('%d-%m-%Y_%H%M')
name = SAVE_LOCATION + "/" + date_time + '.pt'
torch.save(classifier, name)
```