Physics-Constrained Deep Learning for Accelerating Climate Modeling

Thesis approved by
the Department of Computer Science
University of Kaiserslautern-Landau
for the award of the Doctoral Degree

Doctor of Engineering (Dr.-Ing.)
to

Paula Reglindis Harder

Date of Defense: 18.11.2024
Dean: Prof. Dr. Christoph Garth
Reviewer: Prof. Dr. Nicolas R. Gauger
Reviewer: Prof. Dr. Ing. Janis Keuper
Reviewer: Prof. Dr. Philip Stier

DE-386

Abstract

Accurate modeling of weather and climate is critical for taking effective action to combat climate change. Predicted and observed quantities such as precipitation, clouds, aerosols, wind speed, and temperature impact decisions in sectors such as agriculture, energy, health, and transportation. While these quantities are often required at a fine geographical and temporal scale to ensure informed decision-making, most climate and weather models are extremely computationally expensive to run, resulting in coarse-resolution predictions. Recent advances in deep learning (DL) make it an attractive tool for speeding up simulations. The two main ways to decrease computational efforts with DL are downscaling, the increase of the resolution directly on the predicted climate variables, and emulation, the replacement of model parts to achieve faster runs initially.

This thesis leverages DL models for accelerated climate forecasting while making sure the methods are feasible for physical modeling. Standard DL approaches often violate simple physical constraints such as positivity or conservation properties. We develop novel methodologies to incorporate those constraints into the training and into architectures of DL. First, we look into so-called soft-constraining methods that introduce an additional regularisation term. Then several hard-constraining methods that change the neural networks (NNs) architectures, by adding final constraining layers, are discussed.

We consider two application cases to test our constraining methodology and evaluate the potential of DL for speeding up climate modeling. The first test case is downscaling. We not only show how our hard-constraining layers guarantee the constraints to be satisfied, but also increase the overall predictive performance. In the second employment of our constrained DL models, the aerosol microphysics module in the global ICON climate model is replaced by a NN. We both investigate offline performance, as well as implement the NN in Fortran to run it online within ICON, achieving a stable an accurate coupled simulation. We discuss challenges and choices for a successful deployment of DL in climate and weather simulations.

Acknowledgments

This PhD journey has been an incredible time filled with learning, discovering a new field of research, and exploring the world. I am deeply grateful to all the people who made this possible and supported me along the way.

First, I would like to thank my PhD supervisor, Nicolas Gauger, for his constant support, timely advice, and the freedom he gave me throughout this journey. As a newcomer to the field of deep learning, I especially appreciated the mentorship of my supervisor, Janis Keuper. I am thankful for the invaluable meetings, ideas, advice, and help I received from him, particularly during the early stages of my PhD. Later on, I greatly valued the freedom to visit other places and collaborate with different groups. It has been a privilege to have Philip Stier as an advisor and committee member. I sincerely thank him for being part of my PhD, helping me find an impactful climate modeling research project, and offering scientific guidance.

I would like to express my gratitude to Fraunhofer ITWM for generously funding me for 3.5 years. A special thanks to Franz-Joseph Pfreundt for initiating this unique PhD position and welcoming me to the High-Performance Computing Group. I also want to thank Frauke for her constant support with everything, Dominik for his initial help with clusters and Fortran, Peter for taking over the adversarial attack work and exchanging ideas on research and career, and Dominik, Max, and Firozeh for being great colleagues during Covid times. For my first research project on adversarial attacks, I am especially grateful to Margret Keuper for her guidance.

I was fortunate to spend six months of my PhD at the University of Oxford. My heartfelt thanks to Philip Stier for making this possible and for including me in seminars, group meetings, reading clubs, and lectures. This environment provided a unique opportunity to learn more about climate science, especially cloud and aerosol processes. I am immensely grateful to Duncan Watson-Parris for his mentorship as a postdoctoral advisor. His ideas, feedback, and expertise, not only on the M7 emulator project, significantly elevated my PhD experience. A huge thank you to Philipp Weiss, whose Fortran expertise was key in coupling NeuralM7 to ICON—I couldn't have

done it without him. Thanks also to my former colleagues in the Climate Processes Group—Alyson, Andrew, Beth, Ross, Peter, Shipeng, Shahine, and Philipp—for their valuable feedback, assistance, and the welcoming atmosphere. Special thanks to Matthew Chantry for his insightful feedback during a reading group session at the start of the NeuralM7 project.

Midway through my PhD, I was excited to intern at Mila Quebec AI Institute, where the project on physics-constrained downscaling became a core part of this thesis. A huge thank you to the entire team, including IBM researchers Campbell Watson, Daniela Szwarcman, and Prassana Sattegeri. I am grateful to Alex Hernandez for being a fantastic postdoctoral advisor, and to Qidong and Venkatesh for being wonderful fellow interns. Later, I enjoyed collaborating with new team members Christina, Ayush, and Jose—it was a pleasure working with them. I deeply appreciate the immense support from David Rolnick, which extended far beyond his role as a stellar internship advisor. Being part of David's lab was truly enriching; I was fortunate to work alongside inspiring and kind group members, including Melisandre, Alex, Julia, Gabi, Michelle, Charlie, and Devin.

After my first year of PhD research, participating in the Frontier Development Lab summer research sprint was the perfect opportunity to refine my ML skills and expand my network. I am grateful to the Lunar and whole the FDL team for this opportunity, as well as for the chance to return the following year. During my final summer, I interned at the Allen Institute for AI in Seattle, collaborating with the climate modeling team on ocean emulators. I am incredibly thankful to the fantastic team for an enriching learning experience, and to fellow interns James, Salva, Henry, and Prakhar, who made that time truly special.

The field of ML for Earth System Modeling attracts amazing people, and I am lucky to have met so many inspiring colleagues who became friends. My heartfelt thanks go to Alyson, Philipp, Julia, Alex, Gabi, Philine, Salva, James, Valentina, Seb, Ellen, Marlena, and many more. Special thanks to Rosa, the most reliable and supportive friend I could have ever wished for. I also want to thank Michael and his family, the Retter family, Kisa, Jess, Henry, and Paula for their encouragement, hospitality, and advice.

I am deeply grateful for the immense support from my partner, Karl. Thanks also to my brother, who has been an advisor throughout my educational journey. I cannot thank my parents enough for always being there for me. During my PhD years, they reminded me of what really matters, which is not the next talk, publication, or degree.

Contents

Chapter 1

Introduction

Deep learning shows great potential in many fields, including climate modeling. In this thesis, we research approaches to employ deep learning to accelerate simulations, while ensuring physical consistency within the methods used.

The introduction to this thesis consists of three parts, starting with a motivational section that answers why the problem we are tackling is relevant and how the different constituents matter to achieve the goal of accelerating climate models with deep learning.

The second section, the contribution section, describes existing work that addresses similar or related issues concurrently with this thesis. We make clear how our work distinguishes itself from others and pinpoint the ways in which it contributes to scientific advances in the field of deep learning in climate science.

The introduction is then concluded with the third section, an overview of the thesis structure.

1.1 Motivation

This thesis tackles the question of how deep learning can help accelerate climate modeling and which ways of adaptations of existing methods are required to be successful in this novel field of application. In this first section, we answer some motivational and introductory questions:

- How do weather and climate impact society?
- Why do we need faster and more accurate weather and climate models?
- How can deep learning aid climate and weather modeling?

- Why do we need physical constraints incorporated in deep learning models applied to climate modeling?

We discuss these questions while briefly introducing important methods and concepts used throughout the thesis, such as deep learning, physical constraining, downscaling, and emulation. A more detailed description of these methods can be found in Chapter 2.

How do weather and climate impact society? Everyday weather and local climate affect our lives in various ways. Based on the forecasts, we decide whether to put on a raincoat, how to plan our weekend or we receive a warning about health risks from a heat wave. Our food supply through agriculture is highly dependent on temperature and precipitation and transport, tourism, and infrastructure can be influenced by conditions such as snowfall or thunderstorms. Wind speed and solar radiation prediction support the efficient use of renewable energy production. We prepare for disasters such as floods or try to adapt to a changing climate.

What are earth system models? Both climate and weather models describe atmospheric processes governed by equations of fluid dynamics. The same *earth system models* (ESMs) can be used for climate and weather, but they span different temporal and spatial scales. *Weather modeling* is an initial value problem, whereas *climate modeling* is a boundary condition problem. ESMs not only model atmospheric circulation, but different systems are included such as the ocean, the cryosphere[1], and land-surface processes. Modeling the atmosphere alone already involves various subprocesses: radiation, cloud formation, or aerosols are some processes that affect our climate in complex ways while interacting with each other.

Why do we need faster and more accurate weather and climate models? The computational costs of ESMs are immense; it can take months to obtain multidecade, global climate predictions. The computing power needed depends heavily on forecasts' lead time and resolution. Here, we have a trade-off of speed and accuracy. Any effort to include more complex processes in a climate model results in slower runs. For example, aerosol processes are often not represented in sufficient detail in global climate models that contributes to the problem that aerosol forcing remains the largest source of uncertainty in the anthropogenic effect on the current climate [Bellouin et al., 2020].

[1]Cryosphere refers to all parts of the earth system that includes water in solid form, such as sea ice or snow.

Faster modeling can be beneficial in multiple ways. On the one hand, it enables us to run at finer resolution for longer lead times, providing more detailed information on weather and climate. On the other hand, forecasts can be made more easily accessible by being able to run on smaller computing infrastructures, up to the extreme of being able to run on a personal computer. This decrease in computational demand also saves energy and resources and, with that, has economic and environmental advantages.

What is deep learning? In recent years, we all have been able to witness impressive advancement in deep learning, especially with extremely popular tools like ChatGPT [Brown et al., 2020] or DALL·E [Ramesh et al., 2022]. *Deep learning* (DL), is a subfield of *machine learning* (ML), where algorithms learn from data, by searching for an approximation function while optimizing some cost function. *Neural networks* (NNs) are a specific model type within machine learning thatconnect input and output nodes through layers of nodes with linear operations and nonlinear activation function, resulting in so-called universal function approximators [Hornik et al., 1989]. If NNs are involved and consist of two or more hidden layers, it is considered as deep learning. A vast amount of resources in both for-profit and non-profit entities have been put into the research and development of deep learning methodology and tools. Can we take advantage of some of the advancements to benefit a field like climate modeling? For more details on deep learning and various architectures, we refer to Section 2.2.

How can deep learning aid climate modeling? The idea of using ML or, in particular, DL to help modeling earth systems is fairly young, it became popular starting with approaches such as Rasp et al. [2018a] which models clouds with an NN. Two of the most common ways to employ deep learning within climate modeling are so-called downscaling and emulation, both aiming at speeding up simulations but with different approaches. Downscaling allows us to run a climate model at a lower resolution and then increases resolution afterward (see Figure 1.1), while emulation replaces expensive model parts within the simulation with faster ML surrogates (see Figure 1.2).

What is downscaling? *Downscaling* (DS) refers to obtaining some high-resolution (HR) climate data given low-resolution (LR) data, using statistical methods as opposed to physical models. Downscaling via established statistical methods—*statistical downscaling*—has been long used by the climate science community to increase the resolution of climate data [Maraun and Widmann, 2018]. If the LR data is of the same quantity as its HR coun-

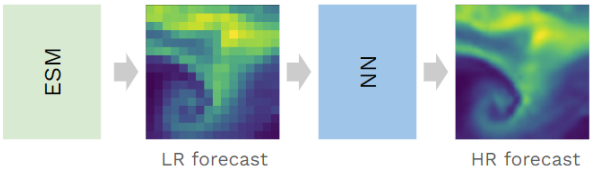

Figure 1.1: A schematic view of downscaling as a super-resolution task tackled by DL. An earth system model creates a low-resolution forecast that is then super-resolved by a neural network. This can save computational costs or time compared to directly creating the high-resolution forecast with the ESM.

terpart downscaling poses a very similar problem to that of *super-resolution* (SR) in computer vision, where the resolution on an image is increased. Following the fast-paced advancements in SR, traditional statistical methods to tackle downscaling are being more and more replaced with deep learning approaches. SR has evolved rapidly using various deep learning architectures, with such methods now including super-resolution convolutional neural networks (CNNs) [Dong et al., 2016], generative adversarial models (GANs) [Wang et al., 2018], vision transformers [Yang et al., 2020], and diffusion-based models [Mardani et al., 2023]. For a detailed introduction to downscaling see Section 2.3.

What is emulation? *Emulation* refers to replacing an entire climate or weather model or parts of it, it is also referred to as surrogate modeling. The goal is to find a much faster replacement for a physical model or describe a process more accurately. This can be achieved by using machine learning. *Full model emulation*, replacing the entire ESM [Watson-Parris et al., 2022] with ML has different challenges than the more common task of replacing specific parts that contribute significantly to the computational costs of simulations [Cachay et al., 2021], *submodel emulation*. An emulator can help with the trade-off of representing certain processes and the computational burden connected to incorporating them [Rasp et al., 2018a]. In submodel emulation, we distinguish between *offline* emulation and *online* emulation. Offline emulation describes running an emulator as a standalone model, without feeding its output back into a bigger model. Online emulation includes coupling the emulator to an ESM, that uses the emulators predictions for different submodels within the ESM.

In this thesis, we only consider submodel emulation, visualized in Figure

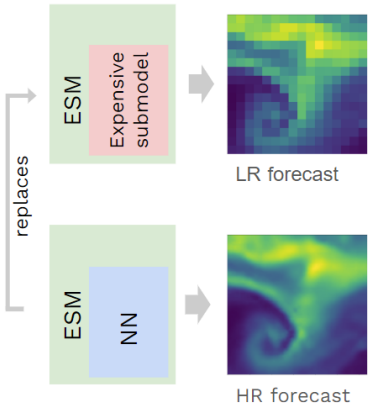

Figure 1.2: A schematic view of emulation tackled by DL. Here, an ESM can be slow including an expensive submodel. Due to computational constraints it produces only a low-resolution forecast. The setup can be speed-up using a NN as a faster surrogate for the expensive submodel. With this faster setup, we are then able to produce a higher resolution forecast.

1.2, but consider both offline and online tasks. The example looked at here is aerosol microphysics emulation. Aerosol microphysics are crucial for accurate climate prediction but often are not modeled in sufficient detail due to computational constraints. For details on emulation see Section 2.4, for more details on aerosols see Section 2.5.

Why do we need physical constraints incorporated in deep learning models? In most cases, deep learning architectures from other fields like computer vision are applied to climate tasks with only minimal changes or slightly adapted to increase predictive accuracy. It is often neglected here that predicted physical quantities require not only accurate models but also physically plausible ones. Some employed DL models predict negative mass, violate conservation laws, or predict a higher minimum temperature than maximum temperature [Wang and Tian, 2022]. First, this decreases the trust and willingness of climate scientists and end users to use these new methods, and second, included within physical models, it can easily cause instabilities, when coupled to classical physical models.

Downscaling methods are mostly used as a post-processing tool. However, downscaling tools can also be applied online within a global climate model [Quiquet et al., 2018], where a lower-resolution output of a climate model part is downscaled and its high-resolution version is fed back into the climate

model.

For emulation tasks that are integrated into earth system models it is inevitable to ensure physically consistent predictions, e.g. a negative mass value causes a crash of the run, and violating a conservation law produces instability issues. A slight bias in mass conservation can get amplified with every time step and lead to a blow-up after multiple timesteps.

Deep learning searches for a close-to-optimal solution in a vast function space; constraining the model decreases this space and thus can help converge to a better local optimum faster, making the learning task easier.

How to do physics-constraining? First works on ML for climate science have attempted to enforce certain physical constraints via soft penalties in the loss [Beucler et al., 2019] inspired by the so-called physical-informed neural networks (PINNs) that solve differential equations by adding them to the loss function [Raissi et al., 2019]. The term physics-informing is used with different meanings; apart from solving differential equations, it is also widely used to describe any physics-related information included in the development of the deep learning methods [Kashinath et al., 2021]. In contrast to that, we use the term *physics-constrained* deep learning here to refer to the case where we have some constraints given by equations or inequalities depending on input and output variables of the network (see Chapter 3). The term *soft constraints* refers to a way of incorporating constraints without any guarantee of them being satisfied at inference time (i.e., when we apply the NN after training), whereas *hard constraints* refers to the case where we do have a guarantee of constraint satisfaction by construction. For details on constraining we refer to Chapter 3.

1.2 Contribution

In this section, we discuss relevant existing work in the field of DL for climate and weather modeling, focusing on downscaling, emulation, and physically constrained architectures. We then specify what our work contributes within the field of DL for climate modeling, both methodologically and by tackling new application cases.

1.2.1 Existing Work

We review works on deep learning for climate modeling in general by going through the most relevant publications in the field. Then we discuss literature on downscaling and emulation, specifically aerosol emulation, giving an

overview of work we build on or that tackles problems similar to the ones addressed by this thesis. We conclude with constrained DL for climate where we discuss the related work in detail.

Deep Learning for Climate Modeling The field of DL for climate and weather modeling has been growing rapidly within the past four years [de Burgh-Day and Leeuwenburg, 2023]. Besides downscaling and submodel emulation which are discussed in more detail below, the most influential works in the field started with learning cloud modeling [Rasp et al., 2018a] and nowadays focus on full weather models [Lam et al., 2022] which are being included for operational forecasting by major institutions.

The first neural network applied in the climate domain was developed by Chevallier et al. [1998] to learn longwave radiative budget from observations with a fully-connected neural network with only one hidden layer. The beginning of the time when ML, especially DL got increasingly influential in the field of earth system modeling was marked by famous papers such as Gentine et al. [2018b] that model moist convection with a fully-connected neural network as a column model, learning from a physical higher resolution model. Other efforts that laid the foundations of the now vibrant field of DL for atmospheric sciences are a neural network approach to approximate the global weather prediction toy model, the Lorenz 95 model [Lorenz, 2006], comparing different architectural choices [Dueben and Bauer, 2018] or using a neural network to predict sources of heat and moisture by learning from a higher-resolution aqua-planet simulation [Brenowitz and Bretherton, 2018]. Pioneering but less well-known work, includes an approach that introduces a novel architecture – the convolutional LSTM (long-short-term memory model) – to accommodate the spatial-temporal structure of climate and weather data and perform short-term rainfall predictions [Shi et al., 2015].

In parallel to the first modeling approaches, DL was frequently employed for tasks that are closer to computer vision tasks, such as the detection and classification of weather events. First works, such as Liu et al. [2016] and Racah et al. [2016] applied CNNs for detecting extreme weather events from climate model and reanalysis data such as tropical cyclones, atmospheric rivers, and weather fronts. Lguensat et al. [2017] introduced a convolutional encoder-decoder model followed by a pixel-wise classification layer to automatically detect oceanic eddies, i.e. moving rotating water masses that give insights into larger-scale oceanic circulations.

After previous work [Schultz et al., 2021] demonstrated the limitations of neural networks in weather prediction recently a new wave of deep learning weather prediction models started to compete with classical deterministic

numerical forecasts. Prominent examples are GraphCast [Lam et al., 2022], which employs a graph neural network architecture [Bronstein et al., 2016], FourCastNet [Pathak et al., 2022], which uses adaptive Fourier neural operators with a vision transformer backbone, and PanguWeather [Bi et al., 2022], utilizing a 3d transformer model. These models now achieve performance close to operational weather forecast [Ben-Bouallegue et al., 2023] on some variables like geopotential height that are easier to predict for ML approaches than other variables, e.g. precipitation. All models use the European Center for Medium-range Weather Forecast (ECMWF) reanalysis data ERA5 (used as well in this thesis) as training data and compete with the Integrated Forecast System (IFS) deterministic forecast provided by the ECWMF.

Replacing operational climate models is a harder problem to solve, with almost no existing approaches yet. Here challenges include ensuring the physical consistency of ML approaches to enable multi-year stable runs. Very recently one of the first works proposed a spherical Fourier neural operator approach [Watt-Meyer et al., 2023].

State-of-the-art deep learning methods such as foundation models [Bommasani et al., 2021, Nguyen et al., 2023a, Lessig et al., 2023] or diffusion-based models [Cachay et al., 2023] achieve impressive results for a variety of tasks, such as regional forecasting, subseasonal forecasting, or downscaling.

For a more detailed review of ML for climate and weather modeling in general the reader is referenced to Reichstein et al. [2019] for earlier works, to de Burgh-Day and Leeuwenburg [2023] for a recent review of modeling approaches and for benchmark and datasets in the field we refer to Dueben et al. [2022].

Downscaling There exists extensive work on ML methods for earth system observation and prediction downscaling [Rampal et al., 2024]. The tasks tackled include a variety of different data sources, climate models, different quantities predicted, commonly temperature, precipitations, wind, or solar data. Downscaling has been applied as well as tackling various local regions as well as global applications, different upsampling factors, and including different input variables. New architectures are developed continuously, driven by the fast advancement in deep learning and its application to super-resolution.

Especially early methods include non-DL ML such as linear regression, e.g. applied to satellite precipitation observation downscaling [Sharifi et al., 2019] or random forests [Pang et al., 2017] for temperature downscaling over southern China. Now almost all of the approaches moved away from those methods and are built on deep learning.

In deep learning, different architectures have been employed for downscal-

ing, usually following state-of-the-art methods in computer vision. The state-of-the-art for super-resolution, when not focused on a specific application task, is mostly measured on a few benchmark datasets such as Set14 [Zeyde et al., 2012], BSD100 [Martin et al., 2001], and Urban100 [Huang et al., 2015]. Starting with convolutional networks [Dong et al., 2014], advanced through residual [Ledig et al., 2016a] and generative adversarial approaches [Wang et al., 2018], the most successful methods now are transformer-based. Currently, transformers employing a shifted window scheme (SwinIR) [Liang et al., 2021] have become very successful including Hybrid Attention Transformers (HAT) [Chen et al., 2022, Zhang et al., 2022]. Compared to super-resolution, there are different metrics and requirements for models in downscaling, so the optimal architecture may vary. For details on differences between super-resolution and downscaling, see Section 2.3.1.

The most common method that emerged as the standard in downscaling is a convolutional neural network (CNN). Different variants of CNNs are used, apart from a standard CNN, U-Nets [Ronneberger et al., 2015] and residual CNNs are powerful tools. first application of a CNN in downscaling [Vandal et al., 2017] used three stacked 3-layer SRCNNs to increase daily precipitation from 100km spatial resolution to 12.5km successively over the United States, including an elevation map as additional input. Sha et al. [2020] use the U-Net, originally developed for semantic segmentation, to downscale minimum and maximum temperature.

Generative adversarial networks (GANs) [Goodfellow et al., 2014b] are becoming a very popular architectural choice, making it possible to give a probabilistic prediction and including more high-frequency information in the output [Harris et al., 2022]. GAN-based works tackle precipitation downscaling [Wang et al., 2021b, Watson et al., 2020, Chaudhuri and Robertson, 2020], but other quantities such as wind and solar data are also considered. Stengel et al. [2020] perform 50x upsampling using a successive architecture, similar to Vandal et al. [2017]. The discriminator then acts on the intermediate medium-resolution prediction as well. Harris et al. [2022] show how GANs outperform CNNs in probabilistic precipitation downscaling, comparing a Wasserstein GAN [Arjovsky et al., 2017] with a variational auto-encoder combined GAN applied to predict radar measurements of precipitation given a UK weather model output.

Very recently diffusion-based models [Sohl-Dickstein et al., 2015] have shown promise in this field [Mardani et al., 2023, Wan et al., 2023] and vision transformer models as foundation models are fine-tuned to tackle downscaling [Nguyen et al., 2023a].

To date, there has been limited work on spatiotemporal SR with climate data. Some authors have looked at super-resolving multiple time steps

at once, but not increasing the temporal resolution [Harilal et al., 2021, Leinonen et al., 2021], whereas Serifi et al. [2021] increase the temporal resolution by just treating the time steps as different channels and using a standard SR-CNN.

Unified frameworks for comparing methods and benchmarks were introduced by Baño Medina et al. [2020] to assess different SR-CNN setups and by Kurinchi-Vendhan et al. [2021] with the introduction of a new dataset for wind and solar SR. Chen et al. [2020] establish a benchmark for small-scale super-resolution on precipitation. The novel climate benchmark dataset ClimateLearn [Nguyen et al., 2023b] includes, next to forecasting benchmark tasks, a downscaling setup learning a mapping from CMIP6 models to ERA5 as a target. A first benchmark dataset for climate model scale downscaling is currently under development [Langguth et al., 2024]. Here, different quantities of the ERA5 dataset such as temperature, precipitation, wind, and solar data are supposed to be brought to the resolution of a higher-resolution observational dataset. For a full review on deep learning for downscaling we refer to the recent overview work Rampal et al. [2024].

Emulation Replacing climate model components with machine learning approaches and, therefore, decreasing a model's computing time has shown promising results, but can struggle with lack of physical consistency [Gentine et al., 2018a] and with the back-integration into climate models.

There are several works on emulating convection. Both random forest approaches [O'Gorman and Dwyer, 2018] and fully-connected neural networks [Rasp et al., 2018b, Gentine et al., 2018a, Beucler et al., 2020b] have been explored. In most approaches, the ML model is trained on data generated from a more complex convection scheme, e.g. 2d cloud-resolving models [Gentine et al., 2018a], that are too expensive to run in a global climate model. In Gentine et al. [2018a] it is emphasized that an issue is the lack of moisture and energy conservation of the NN in terms of a potential integration of the emulator into a climate model.

Multiple consecutive neural networks, one predicting the sign of changes, one the magnitude, have been used to emulate a bin microphysical model for warm rain formation processes [Gettelman et al., 2021] that usually results in a 400x slowdown maintaining the same speed as the control run. Chantry et al. [2021] use a fully-connected neural network to emulate gravity wave drag – the propagation of large-scale waves through the atmosphere, producing accurate and stable forecasts over long timescales. In Bertoli et al. [2023] a random forest and multiple NN approaches are trained to emulate radiative transfer in an offline test case. Whereas the random forest shows

the lowest bias, its memory requirements quickly become prohibitive.

Besides atmospheric processes there exist other subsystems of the climate that can be emulated. One subfield is building ocean emulators that model quantities such as the sea-surface temperature and can be coupled to atmospheric models. Next to LSTMs, reservoir computing – a special kind of recurrent neural network – has emerged as a lightweight and stable emulator [Arcomano et al., 2023, Harder et al., 2023b] trained on reanalysis such as ERA5 sea surface temperature data. A few works exist on land-surface modeling using an NN to predict surface fluxes learned from model data [Meyer et al., 2021].

In addition to many application cases, there now exist benchmark datasets for climate model emulation [Watson-Parris et al., 2022, Cachay et al., 2021, Yu et al., 2023]. Cachay et al. [2021] developed ClimART, a benchmark dataset for radiative transfer based on the Canadian ESM. The first benchmark dataset for full climate projections is ClimateBench [Watson-Parris et al., 2022]. ClimateBench gives different emission pathways of aerosol and greenhouse gases as inputs to then predict temperature and precipitation on a global scale for multiple decades. The ClimateSet dataset [Kaltenborn et al., 2023] contains the inputs and outputs of 36 climate models from the Input4MIPs and CMIP6 archives that can be used to train an ML emulator on several climate models instead of just one to create an emulator that can quickly project new climate change scenarios. Yu et al. [2023] take a similar approach, providing a large emulation benchmark dataset including 5.7 billion pairs of multivariate input and output vectors targeted for hybrid ML-physics modeling. Nguyen et al. [2023b] present the Pytorch framework ClimateLearn that includes benchmark data built on ERA5 and CMIP6 models for forecasting and downscaling tasks.

When considering submodel emulation most works only tackle the offline emulation part and do not go the technical challenging step of incorporating the emulator back into an earth system model. Brenowitz et al. [2020] show that an NN – emulating the aggregate effect of moist physics – that performs well offline struggles with performance online, i.e. a stable coupled run.

Aerosol Emulation There exists various work on aerosol satellite observations [Hameed et al., 2022, Lu et al., 2020], but limited work on aerosol emulation compared to subprocesses like convection. All emulation works consider fairly simple ML architectures, such as random forests or fully-connected neural networks. Silva et al. [2020a] compare several methods, including deep neural networks, XGBoost [Chen and Guestrin, 2016], and ridge regression as an emulator for aerosol activation, including the esti-

mated activation from the Twoomey scheme as another input, predicting the residual. Also, random forest approaches have been used to successfully derive the cloud condensation nuclei (CCN) number concentrations (for an explanation of CCN see Section 2.5) from multi-campaign atmospheric measurements [Nair and Yu, 2020], limited though to an offline proof-of-concept. Schreck et al. [2022] predict secondary organic aerosol formation by comparing a recurrent neural network approach and a fully-connected approach, learning from modeled data from the Generator for Explicit Chemistry and Kinetics of Organics in the Atmosphere (GECKO-A) model. They find the recurrent approach to produce more stable and longer-running box simulations compared to the fully-connected NN. Their evaluation is limited to an offline setup. Very recently, another fully-connected neural network emulation approach was developed that describes the coagulation of aerosols with the models trained on data of a sectional model to replace the said sectional [Okonkwo et al., 2023]. Geiss et al. [2023] emulate radiative properties of aerosol, such as short-wave absorption and extinction, using a randomly wired neural network. Geiss et al. [2023] find improved performance compared to fully-connected architectures but limit themselves when it comes to integration into a climate model, as libraries such as Fortran-Keras-Bridge [Ott et al., 2020] do not support these layers.

Constrained Learning for Climate Modeling First works on ML for climate science have attempted to enforce certain physical constraints via soft penalties in the loss [Beucler et al., 2019] inspired by the so-called physical-informed neural networks (PINNs) that solve differential equations by adding them to the loss function [Raissi et al., 2019]. Soft-constraining has been used more frequently in deep learning for weather and climate. Kashinath et al. [2021], Kriegmair et al. [2021] add a penalizing term for mass conservation, also employing an NN to model convection achieving improved stability for their forecast.

Some recent work applies hard constraints within DL for climate models. [Beucler et al., 2021] uses linearly constrained neural networks for convection using completion methods, here a subset of the output is predicted and the remaining ones are calculated according to the constraints. Zanna and Bolton [2020, 2021] use a final fixed convolutional layer to achieve momentum and vorticity conservation in an ML ocean model. The fixed convolutional layer acts as a finite difference stencil. A different line of work incorporates constraints into machine learning based on flux balances [Sturm and Wexler, 2020, 2021, Yuval et al., 2020]. These strategies use domain knowledge of how properties flow to ensure the conservation of different quantities; instead of

predicting tendencies directly, fluxes between properties are predicted. Here a non-trainable stoichiometric matrix is multiplied in the final layer. Hess et al. [2022] introduce one global constraint to be applied to bias-correct[2] the precipitation prediction generated by a GAN, by rescaling the final output with one scalar. Outside climate science, recent work has emerged on enforcing hard constraints on the output of neural networks by adjusting the optimisation procedure [Donti et al., 2021].

In super-resolution for turbulent flows, MeshfreeFlowNet [Jiang et al., 2020] employs a physics-informed model that adds PDEs as regularisation terms to the loss function. Parallel to our work, the first approaches employing hard constraints for climate-related downscaling were developed: Geiss and Hardin [2023] introduce an enforcement operator applied to multiple CNN architectures for scientific datasets such as ERA5 cloud fraction, satellite radiance data and radar data, with LR counterparts created synthetically by average pooling. Here a scaled additive readjustment layer is integrated as a last layer. Similarly, a CNN with a multiplicative renormalisation layer is used to downscale atmospheric chemistry data in Geiss et al. [2022].

There have been a large number of approaches applying machine learning to climate and weather modeling tackling a variety of scales, tasks, and variables. The field benefits from the rapid advancements in deep learning, from where methods are applied with just slight modification. Despite recent success in the field, we identify a lack of methods to ensure the physical consistency of ML modeling approaches and see opportunities to improve downscaling and online performance in emulation.

1.2.2 Our Work

This thesis aims to advance both the field of constrained DL in general as well as the application fields of downscaling and aerosol emulation (see Figure 1.3).

Constrained Deep Learning We develop novel hard-constraining methods such as readjustment layers that include additive, multiplicative, and softmax constraint layers, as well as a correction layer. We are also the first to rigorously test and compare all constraining methods across different problem setups, datasets, and architectures.

[2]Bias correction is the process of scaling climate model outputs to account for their systematic errors, in order to improve their fitting to observations.

So far most of the applications of deep learning in climate and weather modeling apply deep learning architectures without including any physical constraints or guarantees [Reichstein et al., 2019], struggling with instabilities [Brenowitz et al., 2020] and unphysical predictions [Wang and Tian, 2022]. We are among the pioneers in introducing the concept of constraints into climate modeling and applying hard or soft constraints for downscaling and aerosol emulation.

Many works only consider soft constraints [Kriegmair et al., 2021, Chu and Thuerey, 2017]. While soft constraints are flexible and can be sufficient, they do not provide any guarantees on constraints, unlike our proposed hard constraint layers, as demonstrated in this thesis. Additionally, soft constraints can lead to more unstable training, "require additional tuning, and potentially decrease accuracy as we show in this work (see Section 4.4). We discuss and compare hard and soft constraints experimentally, providing the first benchmarking of different constraining approaches.

Now, there exist first works on hard-constraining, all formulated and applied to specific application cases [Geiss et al., 2022, Hess et al., 2022, Zanna and Bolton, 2020]. Here we develop a generally applicable formulation and framework. Our completion layer (CompL) is a generalisation of the mechanism introduced by Beucler et al. [2020b]. The correction layer (CorL) can be viewed as a generalisation of the ReLU layer, which has been incorporated before for the special case of enforcing the positivity of a prediction [Beucler et al., 2020b]. A specific case of our multiplicative layer as a global constraint (see Section 4.1.2) is applied by Hess et al. [2022] for bias correction. Here, we formulate a more general version of a multiplicative constraint layer (MultCL), that is flexible enough to be applied to any application case and can enforce multiple local constraints. Parallel works [Geiss and Hardin, 2023, Geiss et al., 2022] introduce two layers that enforce consistency in super-resolution applications, which take a similar approach compared to our readjustment layers: a slightly different multiplicative approach and a scaled additive approach. A detailed comparison of our approaches and Geiss and Hardin [2023] can be found in Harder et al. [2023a]. The additive constraint layer (AddCL) and soft-max constraint (SmCL) layers can only be found in our work. They show the best performance across metrics and architectures in downscaling. SmCL uniquely ensures both equality and inequality constraints at the same time.

Downscaling Our main contribution in the field of downscaling, is developing and including constraints that work well in the setting of super-resolution. With these constraints, we then advance many deep learning

architectures that have been applied in this field without any constraints, potentially violating laws of physics [Wang and Tian, 2022].

Besides bringing constraining into the various methodologies for climate super-resolution, we develop a new spatio-temporal architecture, combining a deep voxel flow architecture [Liu et al., 2017] and a convolutional recurrent NN, advancing previous works super-resolving multiple time steps at once, but not increasing the temporal resolution [Harilal et al., 2021, Leinonen et al., 2021], or increasing the temporal resolution by just treating the time steps as different channels and using a standard SR-CNN Serifi et al. [2021].

Additionally, we are the first to apply deep learning for NorESM super-resolution between two climate simulations run at different resolutions with the same setup. Finally, our constraint layers are not limited to climate-related data, we showcase how our new methodology can improve other fields of super-resolution for instance enhancing satellite imagery.

Aerosol Emulation Not much work exists in aerosol emulation, compared to other fields like clouds and convection [Beucler et al., 2020a]. This work is the only one on emulating aerosol microphysics models like M7 [Vignati et al., 2004], distinguishing itself from other approaches such as emulating the single process of aerosol activation [Silva et al., 2020a] containing multiple variables and complex processes.

There are only rare cases that include constraints in emulation [Beucler et al., 2020b] and even fewer that include hard constraints, which makes our work an innovative and impactful effort in the area of emulation, testing the success of constraints in aerosol emulation.

Besides this novel application and developing and including constraints, we are among the few works that go beyond pure ML development in Python, bringing the emulator closer to deployment. We show challenges and design choices for the implementation of an ML emulator within a global climate model such as ICON [Zängl et al., 2015].

We can summarize our main contributions as follows:

- This is the first framework and detailed comparison of different constraining methods in deep learning for climate and weather modeling.
- We develop and successfully apply novel hard-constraining layers in different scenarios, using readjustment techniques. These layers enforce (physical) constraints for deep learning architectures and can improve predictive performance.
- We advance deep learning for downscaling by demonstrating how our

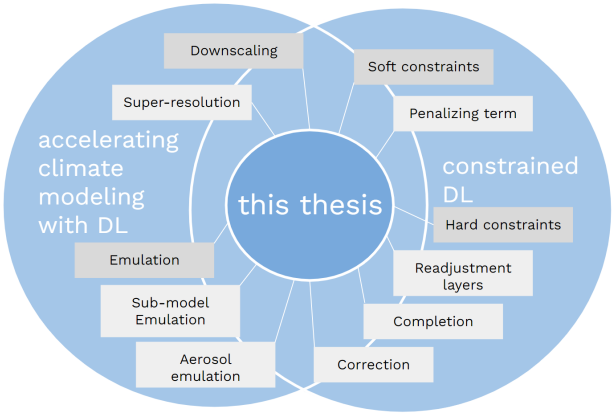

Figure 1.3: The topics of this thesis lie in the overlap of constrained DL and DL to accelerate climate modeling, including downscaling, emulation, aerosol emulation, soft and hard constraining methods, readjustment layers and completion and correction mechanisms.

constraints enhance the performance for a variety of settings and introduce a novel spatio-temporal architecture.

- This work is the only approach to emulate a full aerosol microphysics model and is pioneering with the use of hard constraints in the emulation field.
- The Fortran implementation of the neural network emulator and its integration into and evaluation in ICON gives new insights into the challenges of the deployment of neural networks for climate modeling.

1.3 Thesis Organisation

This work consists of six chapters. After motivating and outlining our work and reviewing existing efforts in Chapter 1, this thesis starts with introducing the foundations of relevant methods with an emphasis on deep learning, then also going over downscaling, emulation, and aerosol modeling (see Chapter 2). For a reader knowledgeable in deep learning, downscaling, emulation or aerosol modeling the respective section can be skipped.

Chapter 3 frames the problem of constraining deep learning architectures, considering both equality and inequality constraints. This chapter introduces

our own newly developed methods for constraining. We show how soft constraints are incorporated through a penalizing term in the loss function that is minimized during training and what different ways there are to hard-constrain neural networks with a final layer, like completion, correction, or readjustment layers.

Chapters 4 and 5 then go through two detailed application cases: Downscaling and aerosol microphysics emulation. In both cases, we outline how constraints are integrated, describe the datasets used for the experiments conducted, and present the results achieved.

The downscaling application case focuses on demonstrating the effect of constraints on a large variety of datasets and deep learning architectures. For aerosol microphysics emulation, in addition to offline experiments, we discuss our test case of implementing the neural network in Fortran and running a global climate simulation with it. The thesis is then concluded with a summary of this work and its overall contribution, as well as outlining future avenues of research.

Chapter 2

Foundations

In this chapter, we introduce all the methods and concepts used within the thesis on a general level. As it is the main tool of this work, we focus on deep learning, explaining all the different components being utilized here. We start by describing machine learning, which forms the basis of deep learning. Then we give a brief description of downscaling, emulation, and aerosol modeling to provide background information for our two application cases, constrained downscaling and constrained aerosol emulation. An introduction to our developed constraint methodologies can be found in the next chapter, Chapter 3.

2.1 Machine Learning

Machine learning (ML) is a computational tool that enables systems to automatically learn and improve from experience without being explicitly programmed [Murphy, 2012]. Algorithms and models are developed to enable computers to discover patterns in data, allowing them to make predictions or decisions. ML algorithms learn from data, identify hidden relationships within that data, and adapt their own behavior over time. ML models iteratively learn by solving an optimisation problem.As opposed to standard optimisation processes, though, the final objective is to perform well on new examples, not only the data seen during optimisation, to enable machines to generalize from specific examples. If the performance is only good on training data, the data used to optimize, and bad on the testing data, the data used for final evaluation, this is known as overfitting, which needs to be avoided.

Supervised vs. Unsupervised ML In machine learning, the distinction between supervised and unsupervised tasks often lies within the type of data

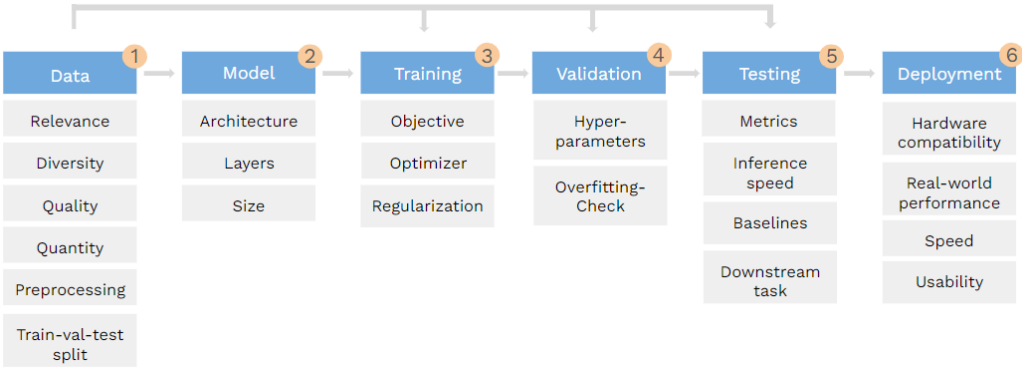

Figure 2.1: An overview of an ML pipeline targeted towards application.

available and thus influences the learning process. In supervised learning, the algorithm is trained on a labeled dataset, where each input is paired with its corresponding output or target. The objective is to learn a mapping from inputs to outputs, enabling the algorithm to make predictions on new, unseen data. Unsupervised learning works with unlabeled data, where the algorithm explores the inherent structure or patterns within the data without explicit guidance on the output. Unsupervised tasks often include clustering, dimensionality reduction, and density estimation. In this work, we only focus on supervised tasks, but our methodological advancements in constrained deep learning are applicable to unsupervised tasks as well.

Classification vs. Regression Tasks In machine learning, the distinction between classification and regression tasks depends on the predicted output. Classification tasks involve predicting the categorical class or label of an input and assigning it to predefined categories. The goal is to learn a decision boundary that separates different classes in the feature space. Examples include spam detection or image classification, where the output is a discrete label. On the other hand, regression tasks focus on predicting a continuous numeric value, aiming to learn a mapping from inputs to a real-valued output. Tasks like predicting house prices or stock prices fall into the regression category. While both classification and regression involve supervised learning and share common algorithmic foundations, the key difference lies in the type of output: discrete labels for classification and continuous values for regression. In this work, we focus on regression tasks, learning the continuous values of physical variables such as aerosol masses or atmospheric water content. A classification model, though. is used at one point when we

employ a generative adversarial network (see Section 2.2.2).

2.1.1 Machine Learning Pipeline

When using machine learning for an application task, it consists of the following steps that need to be considered and that come with their own challenges (see Figure 2.1 for an overview):

- Data: Choosing the right data that represents the final application, most accurately is crucial. The split for validation and testing datasets should be chosen so they represent a generalizability challenge that is close to the one encountered when deploying the model. The data quality highly determines the model's future success and needs to be ensured through careful data analysis and potential cleaning. Whereas non-DL models can rely on feature engineering, choosing the right set of input variables is important. For deep learning, this is often done automatically by the model. For more details on dataset consideration see Section 2.1.3.

- Model: A model needs to be chosen carefully, keeping dataset size, dimensions, data structure, future application, computational capacity, and hardware requirements in mind. Common ML models are described in Section 2.1.4 such as linear regression, random forests, and Gaussian processes for non-DL tasks. For deep learning, i.e., neural network (NN) models, we refer to Section 2.2.2. NNs as opposed to other ML methods are often chosen, in settings where the learning task is high-dimensional or very complex and a lot of data is available to train.

- Training: We choose an objective function to decide what is minimized during training, e.g., the mean-squared error. An optimizer is then applied that decides how the ML parameters are adjusted to iteratively improve the models' predictions on the training set. The most common optimizer for deep learning approaches, the Adam optimizer is described in Section 2.2.3, which builds on the simpler stochastic gradient descent.

- Validation: The separate validation dataset serves several purposes. It makes sure we do not overfit while training and also determines what we choose as our hyperparameters. During training the validation error can be tracked and the training can be stopped once the validation increases due to overfitting (early-stopping). To choose parameters for

the optimizer or the machine learning architectures (number of layers, etc.), the validation set is used to determine the best setup by calculating the evaluation metrics on the validation set (see Section 2.2.4).

- Testing: Choosing metrics that accurately represent the performance of the trained model is the challenge to make the testing phase useful. Usually, multiple metrics are evaluated, including the objective function. The testing phase gives the chance to compare to simple baselines or existing approaches to determine which model shows the most promise to move toward deployment (see Section 2.2.4).

- Deployment: This stage is often not included in research-related ML. It is about bringing the ML model to a real-world application task. First, it needs to be tested whether the performance on the offline test case translates to good performance online, which is often a challenging generalization task. Then there are additional challenges such as software and hardware considerations, in our case transferring from a Pytorch/GPU[1] setting to a Fortran/CPU[2] setting. Considerations such as usability and speed become even more important for deployment compared to development.

2.1.2 Formalisation

Here, we introduce the notations of machine learning used throughout this chapter and the entire thesis. We consider the setting, where $x \in \mathbb{R}^{n_{\text{in}}}$ is the input vector and $y \in \mathbb{R}^{n_{\text{out}}}$ the final output vector of the ML model. The ML model is given by a function $f_\theta : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}}$, parameterized by the parameter $\theta \in \mathbb{R}^l$, so $f_\theta(x) = y$. Given the targets $\hat{y} \in \mathbb{R}^{n_{\text{out}}}$ our supervised learning task is solving following optimisation problem for all training pairs $(x^i, \hat{y}^i)_{i=1,\dots,n}$ for a training set of size n

$$\min_{\theta \in \mathbb{R}^l} \sum_{i=1}^{n} \mathcal{L}(f_\theta(x^i), \hat{y}^i). \tag{2.1}$$

$\mathcal{L}$ is a cost function, with the mean-squared error (MSE) being the most common choice, and is used throughout this thesis. We only consider the supervised learning task in this thesis, which means we always have the target vector $\hat{y}$ provided.

[1] GPU stands for graphics processing unit and is commonly used in ML.

[2] CPU stands for central processing unit and is commonly used in climate modeling.

2.1.3 Datasets

Ensuring to have a suitable and high-quality dataset is usually the first step that is tackled in an ML or DL problem. Here we briefly go through some points that need to be considered when getting the data ML-ready, especially in the context of climate data:

- Domain Relevance: The dataset should be relevant to the problem domain and reflect real-world scenarios to ensure that the model learns meaningful patterns and relationships.
- Data Diversity and Representativeness: Ensuring that the dataset covers a wide range of scenarios, variations, and extreme cases improves the model's ability to generalize to unseen data. This is especially important in a climate science context, where there is a huge variation in the data depending on location and time.
- Quality of Data: High-quality data needs to be both relevant and diverse as said above. Additionally, the data needs to be clean, including no corrupted values, and input and targets need to be matched correctly.
- Quantity of Data: Additionally, a sufficient quantity of diverse data ensures that the model learns robust patterns and features representative of the entire dataset. The quantity highly depends on the dimensionality of inputs and targets. In combination with a high diversity, a high enough quantity of data is crucial to prevent overfitting.
- Data Preprocessing: Proper preprocessing steps such as normalisation, standardisation, and handling missing values are important to ensure that the data is in a suitable format for training. Preprocessing such as normalisation leads to faster training and better performance [Nawi et al., 2013]. Preprocessing also involves data augmentation techniques to increase the diversity of the dataset, especially when dealing with limited data.
- Train-Validation-Test Split: Proper partitioning of the dataset into training, validation, and test sets is essential to assess the model's performance accurately. The training set is used to train the model, the validation set is used to tune hyperparameters and monitor for overfitting, and the test set evaluates the model's performance on unseen data. For future deployment the train-val-test split must resemble the generalisations task that is going to be present in deployment, e.g. a

model that is going to be applied for future data, should be trained on older data than it is tested.

2.1.4 Models

Three common ML methods used for regression problems that are not neural networks and are applied in this thesis to emulate aerosol properties are linear regression, random forests, and gradient boosting.

Linear Regression Linear regression is a statistical method used in machine learning to model the relationship between the target vector, y, (also called dependent variable) and the input vector, x, (also called independent variables or features) by fitting a linear equation through training data. The linear regression equation is expressed as

$$f_\theta(x) = y = \beta_0 + \sum_{i=1}^{n} \beta_i x_i. \tag{2.2}$$

Here, β_0 is the intercept, $\beta_i, i = 1, \ldots, n_{\text{in}}$ are the slope coefficients. The goal of linear regression is to determine the optimal values for $\theta = (\beta_0, \ldots, \beta_n)$ such that the sum of squared differences between the observed and predicted values (residuals) is minimized. Linear regression most commonly uses gradient descent methods, for more details see 2.2.3.

Linear regression is a popular choice in many fields of application, as well as climate science, due to its computational efficiency, simplicity, and interpretability. Linear regression though can lack expressive ability given that it only models linear relationships. It is often used as a first baseline, given a lower bound on performance more complex ML approaches should exceed.

Random Forests Random forest regression is a machine learning ensemble method that combines the predictions of multiple decision trees to improve overall predictive accuracy and robustness [Ho, 1995]. Unlike linear regression, which fits a linear equation to the data, random forest regression is a non-linear model capable of capturing more complex relationships within the dataset.

A decision tree in machine learning is a predictive model that resembles an upside-down tree, where each internal node represents a decision based on a specific feature, and each leaf node corresponds to the predicted outcome. The tree structure is created through a process called recursive partitioning, where at each step, the algorithm selects the best feature to split the data,

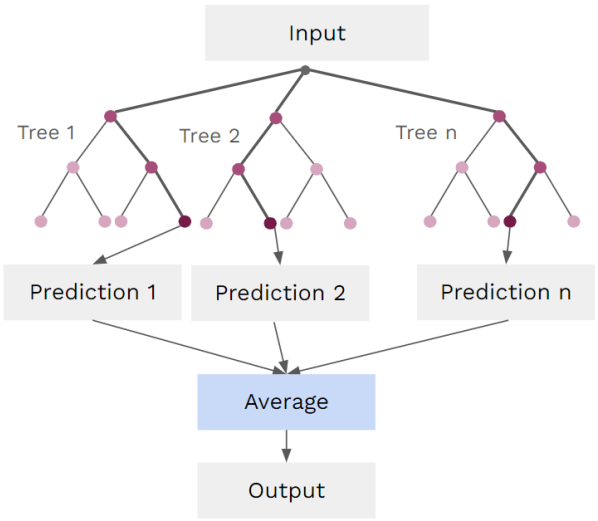

Figure 2.2: A schematic view of a random forest prediction. The input goes through multiple decision trees and in the end the prediction of these trees is averaged to the final prediction.

aiming to maximize the homogeneity of samples within each branch. The decisions made at each node are based on criteria such as Gini impurity [Shi, 2007] or information gain for classification tasks, or variance reduction or root-mean-squared error for regression tasks. By following the branches of the tree from the root to a leaf, an input can be classified or assigned a predicted value.

Each decision tree is trained on a subset of the dataset and makes independent predictions. Randomness is introduced in two main ways: by selecting a random subset of features for each tree and by bootstrap aggregation or bagging, i.e. creating the random subsets of the training data. During the prediction phase, the output of the random forest is determined by aggregating the predictions of individual trees. For regression tasks, this means averaging the predicted values from each tree, resulting in a more robust and accurate prediction than any individual tree alone. A schematic visualisation of a random forest prediction can be found in Figure 2.2.

Random forest regression is particularly well-suited for capturing nonlinear relationships and mitigating overfitting. It is a popular choice for climate emulators, given its simple design and training as well as its ability to conserve properties in the data, such as mass [O'Gorman and Dwyer, 2018]. Additionally, random forests give a feature importance for each input,

making it an interpretable method. On the downside random forests can have high memory requirements and slow inference [O'Gorman and Dwyer, 2018].

Gradient Boosting Gradient boosting [Friedman, 2001] is another ensemble learning technique used for both classification and regression tasks, successfully applied in competition and benchmarks such as ClimateBench [Watson-Parris et al., 2022]. Like random forests, it builds a strong predictive model by combining the predictions of multiple weak learners. However, unlike random forest, which builds independent trees in parallel, gradient boosting constructs trees sequentially, with each tree correcting the errors of the previous ones.

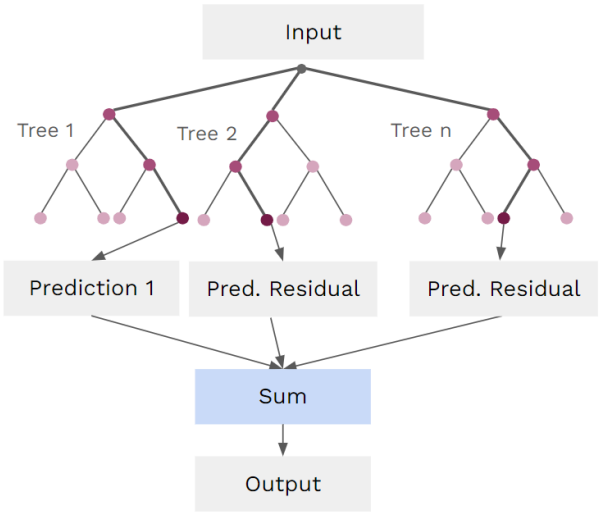

Figure 2.3: A schematic view of a gradient boosting prediction. The input goes through multiple decision trees that are fitted on the residual of the previous trees prediction. The decision trees' predictions are then added up to produce the final prediction.

Gradient boosting begins with a simple model, often a shallow decision tree, as the initial approximation of the target variable. Subsequent trees are then built to correct the errors of the combined predictions of the existing ensemble. The learning process involves adjusting the parameters of each weak learner to minimize a specified loss function, typically the mean-squared error for regression problems. A schematic visualisation of a gradient boosting prediction can be found in Figure 2.3.

The key idea in gradient boosting is to fit each new tree to the negative gradient of the loss function with respect to the ensemble's current prediction. This process is repeated iteratively, with each new tree focusing on the remaining errors that the ensemble has not yet captured effectively. The learning rate parameter controls the contribution of each tree to the ensemble, and regularisation techniques are often employed to prevent overfitting.

The final prediction is obtained by summing the predictions of all trees in the ensemble. Gradient boosting is known for its high predictive accuracy, adaptability to different types of data, and ability to capture complex relationships in the data. In the context of climate modeling or other complex systems, gradient boosting can be a valuable ML method for generating accurate predictions on tabular data.

2.2 Deep Learning

Deep learning (DL) [Goodfellow et al., 2016] represents a subset of machine learning that focuses on the development and training of neural networks. Usually, deep learning is associated with neural networks including multiple layers, also known as deep neural networks.

A neural network is a computational model inspired by the structure and function of the human brain. It consists of interconnected nodes, also called neurons, organized into layers. Each connection between nodes is associated with a weight, representing the strength of the connection. The network receives input data, processes it through a series of layers by adjusting the weights of connections based on the input, and produces an output. Through a process called training, the network learns to adjust its weights to minimize the difference between its predictions and the actual outcomes, enabling it to generalize and make accurate predictions on new, unseen data.

In the following, we discuss different neural network layers used in this work. We then go through full architectures such as convolutional neural networks and generative adversarial neural networks. Finally, we discuss the training procedure and its components, objective function, optimizer, and regularisation as well as the final evaluation of DL models.

2.2.1 Layer Types

Input/Output Layer The input layer is the initial stage of a neural network where raw data is fed into the model for processing. This layer consists of nodes, each representing a feature or attribute of the input data. The number of nodes in the input layer corresponds to the dimensionality of the

input, meaning the number of features or variables. For example, in image classification, each node in the input layer might represent the intensity of a pixel. The values of these nodes are the input signals that propagate through the network.

The output layer is the final stage of a neural network where the processed data is transformed into the desired format for the specific task. This layer typically consists of nodes, with each node representing a possible outcome or prediction related to the task being performed. The number of nodes in the output layer depends on the nature of the problem, such as the number of classes in classification tasks or the number of predicted variables in regression tasks. For instance, in image classification, each node in the output layer may represent the probability of the input image belonging to a particular class.

Input and output layers are visualized in Figure 2.4.

Hidden Layer A hidden layer in a neural network is an intermediate stage between the input and output layers where the raw input data is transformed through a series of computations. Hidden layers can have many different forms, like dense layers, convolutional layer, or recurrent layers (see below).

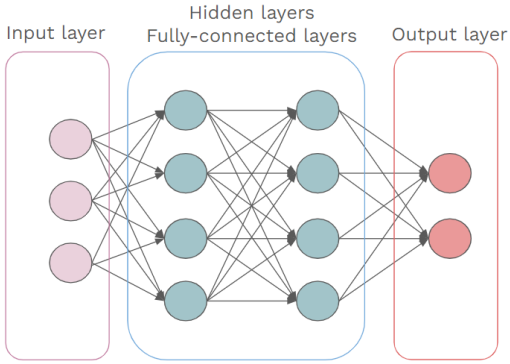

Figure 2.4: A schematic view of a neural network with input layer, hidden layers, that are fuly-connected layers in this case, and output layer. A prediction is created by moving an input through the layers applying linear transformation and non-linear activations.

Fully-Connected Layer In deep learning, a fully-connected or dense layer is a fundamental component where each neuron is connected to every neuron in the previous and subsequent layers, forming a fully-connected network.

It was first introduced as the perceptron [Rosenblatt, 1958]. Represented mathematically, the output y_i of a neuron in the layer is computed as:

$$y_i = f\left(\sum_{j=1}^{N} w_{ij} \cdot x_j + b_i\right). \tag{2.3}$$

Here, N is the number of neurons in the previous layer, w_{ij} represents the weight connecting the j-th neuron in the previous layer to the i-th neuron in the current layer, x_j is the input from the j-th neuron, b_i is the bias term for the i-th neuron, and $f(\cdot)$ is the activation function. Fully-connected layers are visualized in Figure 2.4.

Activation Layer Activation layers or activation functions introduce non-linearity to the model, allowing neural networks to capture complex relationships within the data. One common activation function is Rectified Linear Unit (ReLU) [Nair and Hinton, 2010], defined as

$$f(x) = \max(0, x). \tag{2.4}$$

ReLU sets all negative values to zero, facilitating efficient learning and mitigating the vanishing gradient problem. Leaky ReLU [Maas et al., 2013] is a variant that allows a small negative slope for negative inputs, defined as

$$f(x) = \max(\alpha x, x), \tag{2.5}$$

where α is a small positive constant addressing the problem of dying ReLUs [Goodfellow et al., 2016]. Another activation function is the hyperbolic tangent (Tanh), given by

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \tag{2.6}$$

which squashes input values between -1 and 1, aiding in learning features with zero mean. Sigmoid is another activation function expressed as

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{2.7}$$

mapping input values to the range (0, 1).

The softmax layer is a specialized activation function often used in the output layer of neural network models for multi-class classification problems. It is defined as

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}, \tag{2.8}$$

where x_i is the input to the softmax function for class i, and the denominator is the sum of the exponential values of all inputs across the classes in the output layer.

Pooling Layer A pooling layer acts as a downsampling mechanism, reducing the spatial dimensions of the input data to enhance computational efficiency and promote translation invariance [Goodfellow et al., 2016]. The two most common types of pooling are max pooling and average pooling. In max pooling, the input is segmented into non-overlapping regions, retaining only the maximum value within each region. For a more general max pooling operation with a pooling size of $n \times n$, the output Y for any cell (i, j) is calculated as

$$Y_{i,j} = \max_{k=0}^{n-1} \max_{l=0}^{n-1} X_{ni+k,nj+l}. \tag{2.9}$$

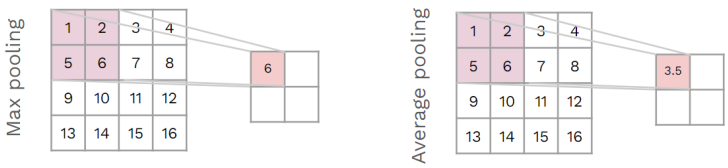

Figure 2.5: An example for max and average pooling. On the left: max pooling takes the maximum value in a region decreasing the size. On the right: average pooling takes the average value in a region decreasing the size

Conversely, average pooling calculates the mean value within each region. Thus, for a pooling size of $n \times n$, the output Y is given by

$$Y_{i,j} = \frac{1}{n^2} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} X_{ni+k,nj+l}. \tag{2.10}$$

An example is visualised in Figure 2.5.

Batch Normalisation Layer A batch normalisation (batch-norm) layer is a technique [Ioffe and Szegedy, 2015] designed to stabilize and accelerate the training of neural networks by normalizing the input of each layer within a mini-batch (for a definition of mini-batch see Section 2.2.3). This is achieved by subtracting the mean and dividing by the standard deviation of the input. Mathematically, for each dimension i of the input x in a mini-batch, a batch-

norm layer performs the following operations:

$$\text{Batch mean: } \mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.11a}$$

$$\text{Batch variance: } \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{2.11b}$$

$$\text{Normalisation: } \hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.11c}$$

$$\text{Scaling and shifting: } y_i = \gamma\hat{x}_i + \beta \tag{2.11d}$$

Here, m is the mini-batch size, μ_B and σ_B^2 are the batch mean and variance, $\hat{x}_i$ is the normalized input, and y_i is the final output after scaling and shifting with learnable parameters γ and β. The addition of a small constant ϵ is to avoid division by zero.

Convolutional Layer A convolutional layer is the key component in convolutional neural networks (CNNs) designed for processing grid-like data, such as images [Goodfellow et al., 2016]. The layer employs learnable filters or kernels to scan across the input data, capturing local patterns and features. Mathematically, the convolution operation for a 2D input X and a filter W is expressed as:

$$Y_{i,j} = \sum_{m,n} X_{i+m,j+n} \cdot W_{m,n} + b \tag{2.12}$$

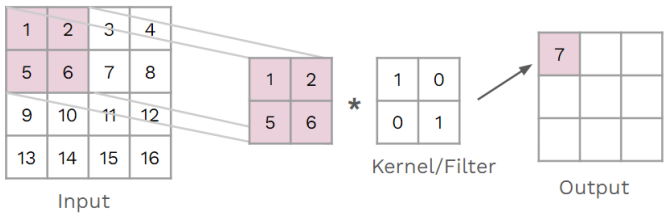

Figure 2.6: An example for a convolution operation. For each region elementwise multiplication with the kernel and then summation of the entries is done to obtain the output value.

Here, $Y_{i,j}$ is the output at position (i, j), $X_{i+m,j+n}$ represents the input at position $(i + m, j + n)$, $W_{m,n}$ denotes the filter weights, and b is the

bias term. The convolutional layer introduces the concepts of stride and padding. Stride determines the step size of the filter movement, influencing the output size, while padding involves adding extra values around the input, preventing information loss at the edges. In climate science, convolutional layers are valuable for extracting spatial patterns from satellite imagery or climate model data, aiding in tasks like super-resolution. An example for a convolutional operation can be found in Figure 2.6.

Upsampling Layer An upsampling layer is a component used to increase the spatial resolution of the input data. A common technique for upsampling is transpose convolution, also known as deconvolution, where each element in the input is expanded into a larger receptive field [Goodfellow et al., 2016].

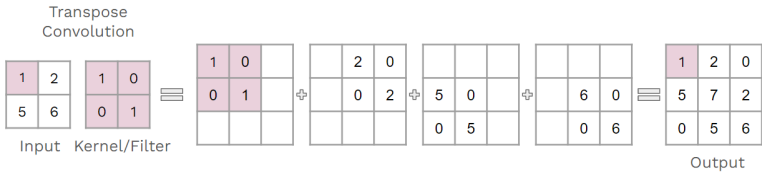

Figure 2.7: An example for a transpose convolution operation. First each entry of the input is multiplied with the kernel, then shifted and added up to the final, higher-dimensional output.

Upsampling layers may also use interpolation methods such as bilinear interpolation or nearest neighbors to fill in the gaps between existing values, enhancing spatial resolution. In tasks like super-resolution or downscaling in climate modeling, upsampling layers play a crucial role in reconstructing high-resolution information from lower-resolution inputs, allowing models to generate detailed outputs from coarser data. An example for a transpose convolutional operation can be found in Figure 2.7.

Residual Layer Residual layers are a building block of neural networks that include a shortcut connection, allowing the model to learn and emphasize residual information—differences between the input and its representation. Residual layers became popular through the introduction of ResNets [He et al., 2016]. Mathematically, for a residual layer, the output y is calculated as:

$$y = f(x) + x. \tag{2.13}$$

Here, x is the input to the layer, $f(x)$ represents the learned transformation by the layer, and $+$ denotes element-wise addition. The primary

idea is that instead of learning to approximate the target directly, the model learns to capture the residual information, making it easier for the network to converge and preventing the vanishing gradient problem. In tasks like super-resolution or downscaling in climate modeling, residual layers facilitate the extraction of fine-scale details and contribute to improving the quality of predictions by focusing on the residual features. A schematic view of a residual layer can be found in Figure 2.8.

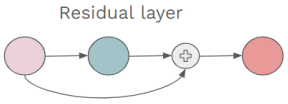

Figure 2.8: A schematic view of a residual layer. The input skips a layer and is added to the output of that layer, in that way a residual is learned.

Recurrent Layer A recurrent layer is a type of layer designed for processing sequential data, making it well-suited for tasks involving time-series information [Hochreiter and Schmidhuber, 1997]. Unlike traditional feedforward layers, recurrent layers have connections that loop back on themselves, allowing the network to maintain memory of past inputs. The output Y_t of a recurrent layer at time t is computed as a function of the current input X_t and the previous hidden state H_{t-1}:

$$H_t = f(W_{hh} \cdot H_{t-1} + W_{hx} \cdot X_t + b_h) \tag{2.14a}$$
$$Y_t = W_{yh} \cdot H_t + b_y. \tag{2.14b}$$

Here, W_{hh}, W_{hx}, W_{yh} are weight matrices, b_h and b_y are bias terms, and $f(\cdot)$ is an activation function. The recurrent layer's ability to retain information from previous time steps is crucial for modeling dependencies in sequential data. In climate science, recurrent layers are important for processing time-series data, such as temperature or precipitation records, enabling models to capture temporal patterns and make predictions based on historical information.

Gated Recurrent Unit Gated Recurrent Units (GRU) are a type of recurrent layer designed to capture long-range dependencies in sequential data while addressing some of the challenges, such as the vanishing gradient problem, faced by traditional recurrent layers [Cho et al., 2014]. The GRU introduces gating mechanisms that regulate the flow of information within the

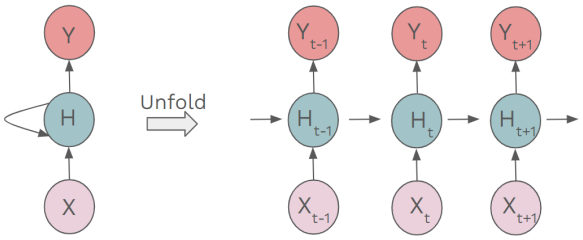

Figure 2.9: A schematic view of a recurrent layer. Each time step is an input to the network, along with a memory, the hidden state (H).

network. The hidden state H_t at time t in a GRU is updated using the following formulas:

$$z_t = \sigma(W_z \cdot [H_{t-1}, X_t]) \tag{2.15a}$$
$$r_t = \sigma(W_r \cdot [H_{t-1}, X_t]) \tag{2.15b}$$
$$\tilde{H}_t = \tanh(W_h \cdot [r_t \odot H_{t-1}, X_t]) \tag{2.15c}$$
$$H_t = (1 - z_t) \odot H_{t-1} + z_t \odot \tilde{H}_t. \tag{2.15d}$$

Here, X_t is the input at time t, W_z, W_r, and W_h are weight matrices, σ is the sigmoid activation function, $\odot$ denotes element-wise multiplication, and H_{t-1} and H_t represent the hidden states at time $t-1$ and t, respectively. The update gate z_t controls how much information to retain from the previous hidden state, and the reset gate r_t determines how much of the past information to forget.

Convolutional GRU Convolutional Gated Recurrent Unit (ConvGRU) layers [Shi et al., 2015] merge convolutional layers with GRUs. In a ConvGRU, the standard GRU operations—reset gate, update gate, and new memory generation—are modified to include convolution operations instead of only matrix multiplications. This modification allows the model to maintain spatial information across the sequence. The update equations for a ConvGRU can be represented as follows:

$$z_t = \sigma(W_z * X_t + U_z * H_{t-1} + b_z) \tag{2.16a}$$
$$r_t = \sigma(W_r * X_t + U_r * H_{t-1} + b_r) \tag{2.16b}$$
$$\tilde{H}_t = \tanh(W * X_t + r_t \odot (U * H_{t-1}) + b) \tag{2.16c}$$
$$H_t = (1 - z_t) \odot H_{t-1} + z_t \odot \tilde{H}_t. \tag{2.16d}$$

Here, $*$ denotes the convolution operation, $\odot$ represents the element-wise multiplication, σ is the sigmoid activation function, and *tanh* is the hyper-

bolic tangent activation function. W, U, and b are the weights and biases associated with the input X_t and previous hidden state H_{t-1}, respectively.

Optical Flow In the context of frame interpolation, optical flow [Horn and Schunck, 1981] is used to estimate the motion between two consecutive frames at the pixel level. If $I(x, y, t)$ represents the intensity of a pixel at coordinates (x, y) at time t, the optical flow assumption states that the intensity of a pixel remains consistent over time, even as it moves. This can be expressed as:

$$I(x, y, t) = I(x + dx, y + dy, t + dt). \tag{2.17}$$

where dx, dy represent the pixel's displacement between times t and $t + dt$. Using this principle, optical flow algorithms compute the motion vectors for each pixel, enabling the synthesis of new frames that smoothly transition between the original ones.

Optical flow layers can be applied in various domains, including video frame interpolation, where missing frames are generated by estimating the motion between existing frames. In climate science, optical flow can be utilized for analyzing the temporal evolution of climate patterns, tracking changes in weather systems over time. This enables tasks such as the interpolation of climate data between time points predicted.

2.2.2 Architectures

Now that we learned about all necessary layer types, we discuss how these can be combined into full models, going through architectures such as fully-connected neural networks, convolutional neural networks, generative adversarial networks, and recurrent neural networks.

Feed-Forward Neural Networks

A feed-forward neural network is a type of artificial neural network where connections between the units do not form cycles [Goodfellow et al., 2016]. This architecture is characterized by the flow of information moving in only one direction—from the input layer, through one or more hidden layers, to the output layer. Each layer is made up of nodes, or neurons, which apply an activation function to the weighted sum of their inputs. Fully-connected neural networks, convolutional neural networks, and generative adversarial neural networks are all kinds of feed-forward neural networks.

Fully-Connected Neural Networks

Fully-connected Neural Networks (FCNNs) or multi-layer perceptrons (MLPs), also referred to as artificial neural networks (ANNs), vanilla NNs, dense NNs, or just neural networks (NNs), are the foundational architecture in the field of deep learning. They consist of input and output layers and all hidden layers are fully-connected layers (see Section 2.2.1). Fully-connected neural networks are a popular choice for climate emulation tasks [Beucler et al., 2020b, Gentine et al., 2018b, Brenowitz and Bretherton, 2018, Gentine et al., 2018a, Gettelman et al., 2021], when working with tabular data, given their expressive power and computational efficiency.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [LeCun et al., 1998] represent a specialized class of neural networks designed to process and analyze structured grid data, particularly well-suited for tasks like image recognition, object detection, and computer vision. CNNs leverage the power of convolutional layers to automatically extract hierarchical features from input data.

A CNN consists of several convolutional blocks, that combine a convolutional layer, a pooling layer and an activation layer. At the end of a CNN we often can find a fully-connected layer to produce the final prediction. A a convolutional neural network might also include residual layers, especially for tasks like super-resolution.

The defining characteristics of a CNN are feature hierarchy, weight sharing and translation invariance. CNNs automatically learn hierarchical representations of features through the convolutional layers. Lower layers capture simple features like edges and textures, while deeper layers assemble these features into more complex structures, allowing the network to learn high-frequency details. Weight sharing is a key concept in CNNs, where the same set of filters is applied across different spatial locations, promoting the extraction of local patterns CNNs exhibit translation invariance, meaning they can recognize patterns regardless of their position in the input space. This property is crucial for tasks like object recognition, where the location of an object in an image should not affect the network's ability to identify it.

CNNs have revolutionized the field of computer vision, providing a robust and efficient framework for extracting meaningful features from structured grid data. Often working with spatial gridded data they have become a popular tool for climate data too. In downscaling, they are used to extract patterns in the low-resolution input that give insights into what the high-resolution target contains [Vandal et al., 2017].

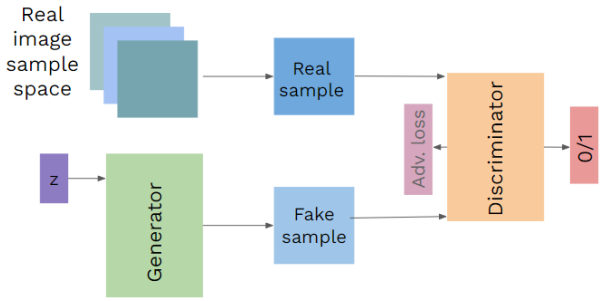

Figure 2.10: A schematic view of a generative adversarial network. Two NNs are included. The generator receives random noise and predicts a fake sample. The discriminator then learns to distinguish real samples from fake samples.

Generative Adversarial Neural Networks

Generative Adversarial Networks (GANs) represent a powerful class of deep learning models designed for generative tasks, such as image synthesis, style transfer, and data augmentation. Introduced by Goodfellow et al. [2014a], GANs consist of two neural networks – a generator and a discriminator – engaged in a adversarial training process. A schematic view of a GAN is shown in Figure 2.10.

Components of GAN:

- Generator: The generator is tasked with creating data instances that are indistinguishable from real data. It takes random noise as input and transforms it into data samples. The generator tries to learn the underlying distribution of the training data to produce realistic outputs.
- Discriminator: The discriminator, on the other hand, acts as a binary classifier that distinguishes between real data samples and those generated by the generator. It is trained to correctly label the origin of the input as either real or generated.

Adversarial Training Process:

- Objective: The objective of GANs is to train the generator to generate realistic data such that the discriminator is unable to differentiate between real and generated samples. Simultaneously, the discriminator aims to improve its discrimination capabilities.

- Minimax Game: The training process of GANs can be viewed as a minimax game between the generator and discriminator. The generator tries to minimize the probability of the discriminator correctly classifying generated samples, while the discriminator seeks to maximize its accuracy in distinguishing between real and generated samples.

- Mathematical Intuition: Let G represent the generator and D the discriminator. The objective function for GANs can be expressed as:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

  Here, $p_{\text{data}}(x)$ is the distribution of real data, $p_z(z)$ is the distribution of noise, x represents real data samples, and $G(z)$ denotes generated samples.

- Equilibrium: Ideally, at equilibrium, the generator produces data that is indistinguishable from real data, and the discriminator cannot confidently differentiate between the two.

In climate modeling, including downscaling, GANs emerged as a common choice, especially when a probabilistic prediction is useful [Harris et al., 2022].

Conditional GAN A conditional GAN (cGAN) [Mirza and Osindero, 2014a] is an extension of the standard GAN that incorporates additional information to guide the generation process. In a cGAN, both the generator and discriminator are conditioned on some auxiliary information, often represented as additional input variables. Mathematically, the objective function for a cGAN is modified to include this additional information:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x,y}[\log D(x|y)] + \mathbb{E}_z[\log(1 - D(G(z|y)))]. \tag{2.18}$$

Here, x is real data, y is the auxiliary information, z is a random noise vector, D is the discriminator, and G is the generator. The generator now takes both random noise and auxiliary information as input to produce synthetic samples. This architecture is a popular choice for image-to-image translation tasks [Isola et al., 2016]. In super-resolution or downscaling the auxiliary information is given by the low-resolution input data. A schematic view of a cGAN is shown in Figure 2.11.

Supervised cGAN/GAN In a cGAN/GAN with a supervised learning component, the model aims to generate outputs conditioned on auxiliary information, while simultaneously minimizing the MSE or MAE loss between

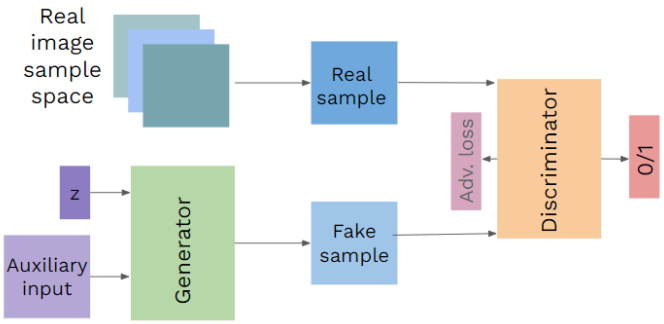

Figure 2.11: A schematic view of a conditional generative adversarial network. Two NNs are included. The generator receives random noise plus an additional auxiliary input and predicts a fake sample. The discriminator then learns to distinguish real samples from fake samples.

the generated outputs and the ground truth. The objective function for such a cGAN with an additional content loss term can be expressed as:

$$\min_G \max_D V(D,G) + \lambda \cdot \mathbb{E}_{x,y}[\|y - G(z|y)\|_2^2]. \tag{2.19}$$

Here, the first part of the objective function is the standard adversarial loss, as in a traditional cGAN. The second part is the MSE loss term, where y is the ground truth, and λ is a balancing parameter.

If an image-to-image translation or specifically a super-resolution task is coming with matching pairs of inputs and targets this extension is a natural choice [Isola et al., 2016] and is commonly used for downscaling [Harris et al., 2022]. A schematic view of a supervised GAN is shown in Figure 2.12.

Recurrent Neural Networks

Recurrent Neural Network (RNN) is a type of neural network designed to process sequential data by maintaining a hidden state that captures information from previous steps. Unlike feedforward networks, RNNs have connections that form a temporal loop, allowing them to capture dependencies over time. The hidden state at each time step is updated based on the current input and the previous hidden state. Next to basic recurrent layers and RNN can include GRU layers as described in the previous section.

RNNs are suitable for tasks involving sequential data, such as time-series analysis in climate science, where they can capture temporal patterns and dependencies within the data. They can be combined with convolutional

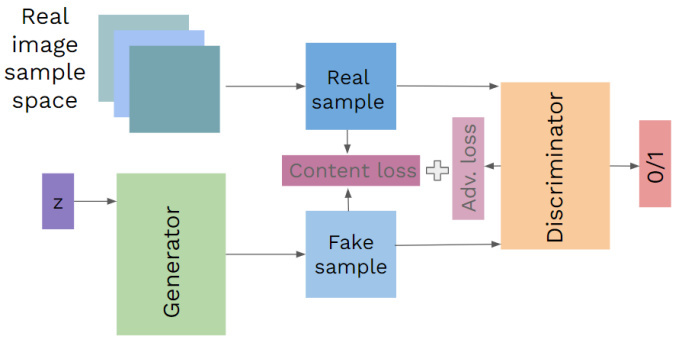

Figure 2.12: A schematic view of a supervised generative adversarial network. Two NNs are included. The generator receives random noise and predicts a fake sample. The discriminator then learns to distinguish real samples from fake samples. The adversarial loss is extended with a supervised content loss term, often the MAE/MSE between real and fake sample.

layers to accommodate spatio-temporal data [Leinonen et al., 2021]. For our downscaling task, we use a convolutional RNN with a mixture of convolutional, residual, ConvGRU, and upsampling layers, inspired by Leinonen et al. [2021].

Encoder-Decoder Architectures

An encoder-decoder architecture is a type of neural network design commonly used for tasks involving sequence-to-sequence mapping [Sutskever et al., 2014], such as machine translation or image captioning. They also exist as convolutional encode-decoders, such as U-Net for semantic segmentation [Ronneberger et al., 2015]. The encoder part of the architecture processes the input data and transforms it into a fixed-dimensional representation called a latent space or embedding. This encoding captures important features and patterns from the input data. The decoder part then takes this encoded representation and generates an output. For our spatio-temporal super-resolution task, we use an encoder-decoder architecture to extract a lower dimensional representation of the input time-series. A schematic view of an encoder-decoder architecture is shown in Figure 2.13.

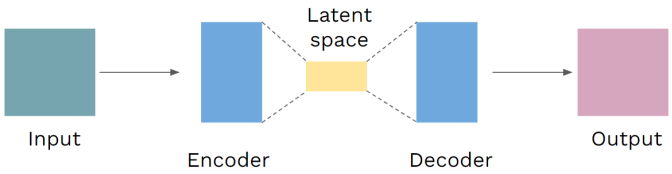

Figure 2.13: A schematic view of an encoder-decoder architecture. The input is passed through an encoder that decreases the dimension to a latent space representation. The decoder is the counterpart that then increases the dimension again.

2.2.3 Training Procedure

Data Preprocessing

Data preprocessing is a critical step in training neural networks, ensuring the model learns efficiently. Two fundamental techniques for preprocessing are normalisation and standardisation, which help in scaling the input data. Other preprocessing techniques include data augmentation, useful in a low-data regime. But as we are not limited by the amount of data, it is not applied here.

Normalisation Normalisation adjusts the data to a specific range, often [0, 1], making it easier for the network to converge. It is computed for a given feature by subtracting the minimum value and dividing by the range of the feature values. Mathematically, if x is an original value, x_{norm} is the normalized value, x_{min} and x_{max} are the minimum and maximum values of that feature, respectively, the formula for normalisation is:

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}. \tag{2.20}$$

Standardisation Standardisation, on the other hand, adjusts the data so that it has a mean of zero and a standard deviation of one. This technique does not bind values to a specific range, which may be beneficial for features with outliers or when the distribution is not Gaussian. The standardisation of a value x to x_{std} involves subtracting the mean μ of the feature and dividing by the standard deviation σ:

$$x_{\text{std}} = \frac{x - \mu}{\sigma}. \tag{2.21}$$

In special cases, a log transformation for training might be a good choice too. This is e.g. done when super-resolving precipitation data [Wang et al., 2021a], where the distribution is a log-normal distribution.

Objective Function

The choice of the objective function or loss function is crucial for the success of training a deep learning model. In regression tasks, where the goal is to predict continuous values, the mean-squared error (MSE) is commonly used as the loss function. For a set of predictions $\hat{y}_i$ and corresponding ground truth values $\hat{y}_i$, MSE is defined as

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2, \tag{2.22}$$

where N is the number of samples.

In classification tasks, where the aim is to assign data points to specific classes, Cross-Entropy Loss, also known as log loss, is frequently employed. For binary classification, the binary cross-entropy loss is given by

$$\text{Binary Cross-Entropy} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(y_i) + (1-\hat{y}_i)\log(1-y_i)], \tag{2.23}$$

where $\hat{y}_i$ is the predicted probability of class 1. These loss functions quantify the dissimilarity between model predictions and ground truth values, guiding the optimisation process to iteratively update model parameters for better performance.

Optimizer

Optimizers play an important role in training deep learning models by adjusting the model's parameters during the learning process to minimize the difference between predicted and actual values. Among the various optimizers, Adam [Kingma and Ba, 2015] (short for Adaptive Moment Estimation) has gained popularity for its efficiency in balancing speed and accuracy and is used here throughout the thesis. As Adam builds on gradient descent and stochastic gradient descent, we introduce these briefly as well.

Gradient Descent Gradient descent is an optimisation algorithm used in training machine learning models, aiming to find the minimum of a cost function by iteratively adjusting model parameters. It computes the gradient,

or the partial derivatives, of the cost function with respect to each parameter and updates the parameters in the opposite direction of the gradient. The update rule for parameter w is given by:

$$w = w - \eta \cdot \nabla J(w).$$

Here, η is the learning rate, controlling the step size in each iteration, and $\nabla J(w)$ represents the gradient of the cost function $J(w)$.

Stochastic Gradient Descent Stochastic gradient descent (SGD) is a variant of gradient descent that processes one training example at a time, making it computationally more efficient for large datasets. The update rule for SGD is similar, but it uses the gradient of the cost function for a single training example:

$$w = w - \eta \cdot \nabla J(w; x_i, \hat{y}_i).$$

Here, x_i and $\hat{y}_i$ are the features and targets of the i-th training example. SGD is particularly useful in combination with large datasets where processing the entire dataset in each iteration can be computationally expensive.

Mini-Batch Mini-batch gradient descent is a compromise between SGD and gradient descent. Instead of using the entire dataset or a single sample to compute the gradient, it uses a small, randomly selected subset of the training data, called a mini-batch. The size of a mini-batch can vary but is typically between 16 and 512 samples. This approach reduces the variance in parameter updates compared to SGD, leading to more stable convergence, while still being computationally more efficient than batch gradient descent. The use of mini-batches allows for the utilisation of optimized matrix operations in hardware accelerators like GPUs, further enhancing the training speed [Goodfellow et al., 2016]. If you have a training dataset of N samples and choose a mini-batch size of m, you would divide the dataset into $\lceil \frac{N}{m} \rceil$ mini-batches. The gradient of the loss function is then computed and the model's parameters updated for each mini-batch, iteratively improving the model with each step.

Adam Adam is built on SGD, it combines ideas from both momentum-based optimisation [Nesterov, 1983] and root mean square propagation [Hinton, 2012]. It maintains two moving averages for each parameter: the first moment (m_t, mean) and the second moment (v_t, uncentered variance). These moving averages are used to adaptively adjust the learning rates for each parameter during training. The update rule for the parameter w using Adam is given by:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla J(w)$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla J(w))^2$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$w = w - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Here, β_1 and β_2 are exponential decay rates for the moving averages, t is the iteration step, $\nabla J(w)$ is the gradient of the cost function, η is the learning rate, and ϵ is a small constant to prevent division by zero. The first and second moments are used to compute biased estimates ($\hat{m}_t$ and $\hat{v}_t$), and these estimates are then utilized to update the parameters.

Regularisation

Regularisation techniques are employed to prevent overfitting and enhance the generalisation capability of models.

L1 regularisation [Tibshirani, 1996], or Lasso regularisation, incorporates a penalty equal to the absolute value of the magnitudes of the model's weights into the loss function, thereby encouraging sparsity in the model weights:

$$L_{\text{L1}} = \lambda \sum_i |w_i|,$$

where w_i denotes the weights of the model, and λ is a coefficient that regulates the regularisation strength.

L2 regularisation [Hoerl and Kennard, 1970], also known as Ridge regularisation, involves adding a penalty based on the square of the weights' magnitudes to the loss function. This encourages the distribution of weights to be small and evenly spread, rather than allowing few weights to become very large:

$$L_{\text{L2}} = \lambda \sum_i w_i^2.$$

Weight decay [Goodfellow et al., 2016] is very similar to L2 regularisation. For many optimisation algorithms (like classic stochastic gradient descent), applying weight decay is mathematically equivalent to L^2 regularisation. However, the distinction becomes important in the context of adaptive learning

rate optimizers like Adam. In these cases, applying weight decay as a separate step from the gradient update (true weight decay) can lead to different behavior compared to incorporating the weight penalty into the loss function (as in L^2 regularisation). True weight decay modifies the weights directly after the adaptive learning rate has been applied, which can lead to different, and sometimes more desirable, training dynamics [Loshchilov and Hutter, 2017].

Dropout [Srivastava et al., 2014], a different form of regularisation, enhances model robustness by randomly omitting a subset of neurons during the training process. This prevents neurons from becoming overly reliant on the presence of specific other neurons, effectively reducing overfitting:

$$\text{Dropout: output} = \text{input} \times \text{mask},$$

where "mask" is a binary vector, with each element having a probability p of being one, and $1 - p$ of being zero, thereby randomly deactivating a portion of neurons.

Hyperparameters

In deep learning, hyperparameters are external configuration settings that influence the training process and architecture of a neural network, and they are distinct from model parameters that are learned during training. Common hyperparameters include the learning rate (η), which determines the step size in updating model weights during optimisation, and the batch size (B), which specifies the number of samples processed together in each training iteration. Another crucial hyperparameter is the number of epochs (E), representing the number of times the entire training dataset is passed through the network. Additionally, the choice of activation functions, regularisation strength (λ), and architecture-related parameters, such as the number of layers and units, are considered hyperparameters. Finding an optimal set of hyperparameters is a crucial task for effective model training and generalisation.

Hyperparameters are typically chosen through a combination of expert knowledge, empirical testing, and automated search techniques. Initially, domain knowledge and previous experience can guide the selection of reasonable starting values. However, because the optimal hyperparameters can vary widely between different datasets and model architectures, empirical testing is often necessary. This testing is usually performed via grid search, where a predefined range of values for each hyperparameter is systematically evaluated, or random search, which randomly selects hyperparameter values within specified ranges and is often more efficient than grid search

[Bergstra and Bengio, 2012]. Additionally, more sophisticated methods like Bayesian optimisation [Snoek et al., 2012], genetic algorithms [Bäck, 1996], and gradient-based optimisation have been developed to efficiently explore the hyperparameter space.

2.2.4 Evaluation

Depending on the task there are different metrics used to evaluate the performance of a deep learning model. Usually a variety of quantitative measures are used for a thorough evaluation in parallel. The metric can be the same function as the objective function, such as MSE or mean absolute error (MAE). Other important metrics, especially in scientific applications, are bias, R^2-score, and correlation. We go into detail on metrics for the two application tasks that are considered: downscaling and emulation (see Section 2.3.5 and Section 2.4.4). If we consider a multi-variable output with variables operating on different scales and we need one metric that aggregates all the variables metrics, we use normalized versions of e.g. RMSE or bias.

Predictive accuracy compared to a ground truth might not be the only quantity to consider. Speed during both training and especially inference can be as or even more important depending on the application. Additionally, in tasks like emulation stability of the trained model is an important quality to take into account. This often is connected to generalisation ability, which is crucial when applied to different times or locations. Moving to deployment evaluation can include downstream tasks, e.g. in downscaling, counting the occurrences of extreme events that are relevant for a user.

Another important component of the evaluation phase is comparison to baselines and state-of-the-art models, to evaluate if and how the proposed new architecture adds value to the field.

2.3 Downscaling

Downscaling (DS) in climate science and super-resolution (SR) in machine learning (ML) refers to a function from low-resolution (LR) input data to a high-resolution (HR) target data; in SR the high-resolution prediction is referred to as super-resolved (SR) data. Downscaling via established statistical methods—statistical downscaling—has been used long by the climate science community to increase the resolution of climate data [Maraun and Widmann, 2018] and recently more and more tackled by the machine learning community.

Classical downscaling is divided into two subfields, dynamical downscaling and statistical or empirical downscaling. Such as empirical/statistical downscaling, dynamical downscaling is a technique used in climate modeling to refine global climate simulations at a regional or local scale. Dynamical downscaling involves using regional climate models (RCMs) with higher spatial resolutions to simulate climate conditions for specific areas. These RCMs are nested within GCMs and utilize their output as boundary conditions. By incorporating local topography and geography, RCMs can provide more accurate and detailed information about regional climate patterns, helping researchers understand local climate variations and potential impacts, such as changes in precipitation, temperature, and extreme weather events.

2.3.1 SR in Computer Vision vs. Downscaling

Despite some similarities, there are a variety of characteristics that distinguish super-resolution in computer vision (CV-SR) and downscaling in climate science.

Terminology In super-resolution and downscaling, there are – sometimes confusing – differences in the terms used in machine learning or climate science. In machine learning/super-resolution the process of increasing the data's resolution is referred to as *upsampling*, whereas in climate science it is *downscaling*. Analogously, decreasing the resolution is called *downsampling* in ML and upsampling in an Earth system context.

Data Next to the obvious difference that SR in computer vision works with different data, i.e. images of faces, scenes, etc., and downscaling targets physical quantities such as temperature, precipitation, solar radiation, or winds, there are other differences concerning the data. The low-resolution version in SR for computer vision can have several forms of degradation applied to it, such as blurring, subsampling, or adding noise. Climate data, when at lower resolution is usually through wider grid spacing in simulations. While climate data can be transformed onto a regular grid, resembling the image pixel format, it originally comes from a variety of different grids or point measurements.

Settings Super-resolution and downscaling both include a few subtasks. Single-image super-resolution (SISR) in computer vision is the most common task and refers to enhancing the resolution of one given low-resolution image. While this is also a setting in downscaling, e.g. where a low-resolution

prediction of rainfall is super-resolved, downscaling often includes a variety of different input variables and might not even include the predicted quantity itself, being closer to a prediction task than SR in computer vision. Multi-frame SR includes multiple input images, usually taken from the same scene. Multi-frame SR is mostly a common task when working with satellite imagery [Martens et al., 2019], which shows similarities to downscaling setups like perfect prognosis (see below) that include a variety of variables as inputs. Video super-resolution or video frame interpolation is similar to super-resolving spatio-temporal climate data. For more details on the different tasks within downscaling, see Section 2.3.2.

Objective The overall objectives of CV-SR and DS are often different. Whereas in CV-SR the goal is to create images that look better and less blurry, downscaling handles physical quantities, so it is less about qualitative measures, but about correctness. This is reflected in the metrics, whereas in CV-SR structural similarity (e.g. measured with SSIM) and peak-signal-to-noise ratio are the standard measures in downscaling bias, mean-squared error or probabilistic measures such as CRPS are commonly integrated (see Section 2.3.5 for details). In downscaling it can be crucial to have a probabilistic prediction, e.g. connected to rainfall predictions. Depending on the authors and audience, metrics from computer vision might still be used for downscaling in a machine learning focused environment.

2.3.2 Different Tasks

There are different tasks and setups in downscaling: super-resolution, perfect prognosis, model output statistic, weather generators, and (spatio-) temporal downscaling. In this thesis, we will consider the super-resolution case and spatio-temporal downscaling. We briefly describe all the different setups: Perfect prognosis, model output statistics, super-resolution, and (spatio-)temporal super-resolution to clarify how the setup we are considering distinguishes itself from other approaches.

Perfect Prognosis In perfect prognosis, there are usually multiple input fields given, that differ from the target quantity. Perfect prognosis uses an empirical model to learn a mapping between large-scale global climate model variables, such as geopotential height, pressure, and air temperature, and local-scale quantities observed at the surface, e.g. surface temperature or precipitation. The training data for perfect prognosis are observational data. It works under the assumption that the learned relationship from ob-

servational data can then be transferred to climate model data, applying the trained model for future climate projections, that only include the larger-scale variables.

Model Output Statistics Model output statistics (MOS) is a technique for downscaling that includes correcting biases[3]. The setup is similar to perfect prognosis, i.e. MOS lears a mapping from larger-scale variables to local-scale variables, but the difference lies in the data source. MOS directly trains on larger-scale quantities from climate model outputs, which then introduces the additional challenge of learning the bias between models and observations. The target variables remain observational.

Super-Resolution In super-resolution the input variable is the same quantity as the output variable. Here, either high-resolution data is taken and low-resolution counterparts are created by downsampling the HR data or LR and HR come from different simulations of the same model. In the latter case, there is some bias correction part included. If the LR data comes from a simulation and the HR from observations, it is considered model output statistics (see below).

(Spatio-)Temporal Downscaling This setup extends spatial downscaling by including the temporal dimension. Here either only the temporal dimension is increased or spatial and temporal dimensions are enhanced simultaneously.

2.3.3 Data

First, data used for downscaling has to address the general points for datasets in ML formulated in Section 2.1.3: we need to ensure the data is relevant to the problem we are trying to solve, this involves e.g. having the right resolution and the right variables. The training data also need to cover seasonal, day-time dependent, or location-dependent differences. Next to having high-quality data, we need at least several thousand, better several ten thousand samples to perform downscaling successfully, as it is usually a high-dimensional problem.

The dataset used for a specific downscaling problem depends on the kind of task we are tackling. The task can define the data source. For perfect prognosis, the data comes from observations or reanalysis data. For MOS

[3]Bias correction is the process of scaling climate model outputs to account for their systematic errors, in order to improve their fitting to observations.

LR samples come from model data, whereas the target data is obtained from higher-resolution observations too. For super-resolution there can either be a synthetic setup, where the HR data is given by HR observations or modeled data, the LR is then created by downsampling the HR data or HR and LR include the same quantity but come from simulations at different scales or observations with varying resolutions. If LR and HR data come from different sources an important point is to align the data points. Here it can be necessary to regrid data or even employ an unsupervised approach.

2.3.4 Architectures

Traditionally, downscaling is tackled by statistical methods, but now deep learning shows the potential to outperform these methods [Baño-Medina et al., 2021]. Various architectures can be applied depending on the specific requirements of the downscaling task. Here, we go through the most common architectures that we also incorporate in Chapter 4 later.

CNNs To extract and recognize spatial patterns in gridded climate data, CNNs are a natural fit and are commonly used for downscaling [Vandal et al., 2017]. Here, the input is the low-resolution climate variable (or multiple variables) that is transformed to its super-resolved counterpart. A necessary feature of SR-CNNs (super-resolution CNNs) is an upsampling layer to increase the resolution of the spatial field unless the input is preprocessed by using bicubic interpolation or similar strategies. Most SR-CNNs differ from classification CNNs in that they do not employ pooling layers to decrease the dimensionality; the U-Net architecture is an exemption with its encoder-decoder architecture. For super-resolution residual layers are a useful building block, given that the LR input and HR target share a lot of the same information. An example of an SR-CNN architecture using a transpose convolution is shown in Figure 2.14. For more details on CNNs see Section 2.2.2.

GANs In climate modeling, one low-resolution input may correspond to multiple potential high-resolution counterparts. To address this challenge, it has become more common to employ GANs for climate variable downscaling. GANs consist of a generator that generates high-resolution samples and a discriminator that evaluates the realism of those samples. During training, the generator learns to produce SR representations that closely resemble true high-resolution data, while the discriminator distinguishes between generated and real HR samples. This adversarial training process refines the generator's

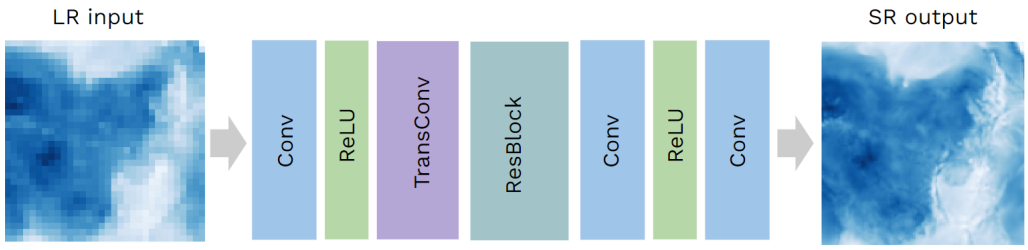

Figure 2.14: An example for an super-resolution convolutional neural network architecture. the input s passed through a convolutional layer to extract spatial pattern, the ReLU introduces non-linearities. A transpose convolution then increases the spatial dimension. In that higher dimension a residual block is included to learn fine-grained details.

ability to produce realistic, high-resolution outputs, improving downscaling of climate variables [Harris et al., 2022]. A schematic view of an SR GAN architecture is shown in Figure 2.15. For more details on GANs see Section 2.2.2.

RNNs Even when only spatial SR is performed the underlying climate data often comes with a temporal dimension that can be utilized. The previous timestep can provide useful information for predicting the spatially super-resolved data. Here, architectures are used that combine recurrent layers with convolutions, such as ConvGRUs [Leinonen et al., 2021]. For more details on RNNs see Section 2.2.2.

Frame Interpolation Networks As a common tool for video super-resolution or video frame interpolation, frame interpolation networks can be applied for temporal downscaling. Here, often optical flow is used to predict an intermediate frame. This can be applied to predicting an intermediate time step in downscaling, potentially combined with deep leanring. We incorporate an architecture like this, employing deep voxel flow [Liu et al., 2017]. A combination of a ConvGRU and a frame interpolation network is shown in Figure 2.16.

2.3.5 Metrics

Super-resolution/downscaling methods are evaluated using various metrics to assess the quality and fidelity of the generated high-resolution (HR) out-

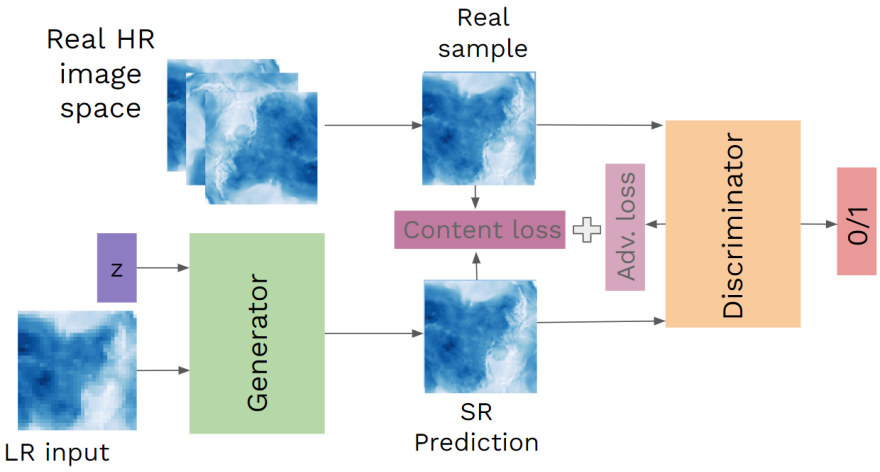

Figure 2.15: A schematic view of a supervised conditional generative adversarial network applied to super-resolution. Two NNs are included. The generator received random noise and a LR input and predicts a a SR version. The discriminator then learns to distinguish real samples HR samples from SR samples.

puts. We note the predicted values with y and the target with $\hat{y}$. Here, are some commonly used metrics, both from computer vision and meteorological evaluation:

- Structural Similarity Index (SSIM) [Wang et al., 2004] measures the structural similarity between the original and generated images. Formula:
$$\text{SSIM}(y, \hat{y}) = \frac{(2\mu_y\mu_{\hat{y}} + C_1)(2\sigma_{y\hat{y}} + C_2)}{(\mu_y^2 + \mu_{\hat{y}}^2 + C_1)(\sigma_y^2 + \sigma_{\hat{y}}^2 + C_2)}, \tag{2.24}$$
where μ is the mean, σ is the standard deviation, and C_1, C_2 are constants.

- The Multi-Scale Structural Similarity Index (MS-SSIM) [Wang et al., 2003] is an extension of the SSIM metric that incorporates information from multiple scales to provide a more comprehensive evaluation of image quality. It considers three components: luminance (l), contrast (c), and structure (s). At each scale, SSIM is calculated, and then the scores are combined. The MS-SSIM formula is given as follows:
$$\text{MS-SSIM}(y, \hat{y}) = l_i(y, \hat{y})^\alpha \cdot \prod_{i=1}^{L} \left[c_i(y, \hat{y})^\beta \cdot s_i(y, \hat{y})^\gamma\right]. \tag{2.25}$$

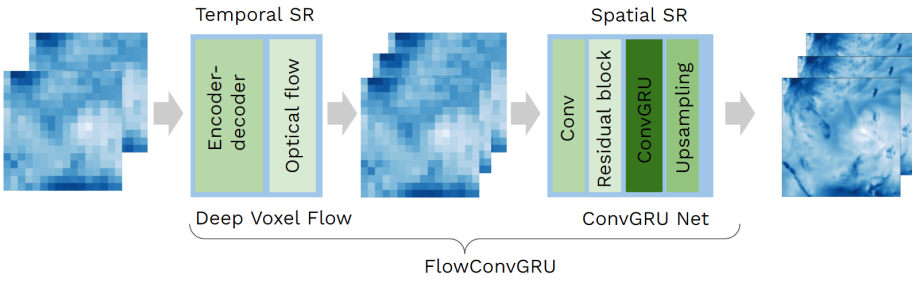

Figure 2.16: An example of a combination of frame interpolation network and spatial SR network for spatio-temporal super-resolution. Two LR time steps are passed through the Deep Voxel Flow NN to output an intermediate time step. These three time steps are then spatially super-resolved through a ConvGRU.

Here, L is the number of scales. $l_i(y, \hat{y})$, $c_i(y, \hat{y})$, and $s_i(y, \hat{y})$ are the luminance, contrast, and structure scores at scale i. α, β, and γ are parameters that control the relative importance of the luminance, contrast, and structure components.

Each of the components is calculated as follows:

$$l_i(y, \hat{y}) = \frac{2\mu_{y_i}\mu_{\hat{y}_i} + C_1}{\mu_{y_i}^2 + \mu_{\hat{y}_i}^2 + C_1}$$

$$c_i(y, \hat{y}) = \frac{2\sigma_{y_i}\sigma_{\hat{y}_i} + C_2}{\sigma_{y_i}^2 + \sigma_{\hat{y}_i}^2 + C_2}$$

$$s_i(y, \hat{y}) = \frac{\sigma_{y_i\hat{y}_i} + C_3}{\sigma_{y_i}\sigma_{\hat{y}_i} + C_3}.$$

Here, y_i and $\hat{y}_i$ are the downsampled versions of y and $\hat{y}$ at scale i. μ_{y_i}, $\mu_{\hat{y}_i}$, σ_{y_i}, $\sigma_{\hat{y}_i}$, and $\sigma_{y_i\hat{y}_i}$ are the mean, standard deviation, and cross-covariance of y_i and $\hat{y}_i$. C_1, C_2, and C_3 are small constants to avoid division by zero. The MS-SSIM metric offers a more nuanced evaluation by considering multiple scales, capturing both global and local structural information in images.

- Peak Signal-to-Noise Ratio (PSNR) [Gonzalez and Woods, 2006] measures the ratio of the maximum possible power of a signal to its error:

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right), \tag{2.26}$$

where MAX is the maximum possible pixel value and MSE is the mean squared error.

- Pearson Correlation [Rodgers and Nicewander, 1988] measures the linear relationship between the original and generated data:

$$\text{Correlation} = \frac{\text{cov}(y, \hat{y})}{\sigma_y \sigma_{\hat{y}}}, \tag{2.27}$$

where cov is the covariance, and σ is the standard deviation.

- Root Mean Squared Error (RMSE) quantifies the average magnitude of the differences between corresponding values of the original and generated data:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}, \tag{2.28}$$

where n is the number of samples.

- Mean Absolute Error (MAE) measures the average absolute differences between corresponding values of the original and generated data:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{2.29}$$

where n is the number of samples.

- Mean Bias represents the average difference between the original and generated data:

$$\text{Mean Bias} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i), \tag{2.30}$$

where n is the number of samples.

- Variance measures the degree of spread or dispersion of the generated data:

$$\text{Variance} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})^2, \tag{2.31}$$

where n is the number of samples and $\bar{\hat{y}}$ is the mean of the generated data.

- Fractional skill score (FSS) [Roberts and Lean, 2008] is a statistical metric used to assess the spatial accuracy of model predictions, commonly employed in fields such as meteorology and climate science. The FSS quantifies the agreement between predicted and observed fields at different spatial scales. Mathematically, the FSS is defined as:

$$FSS = 1 - \frac{\sum_{i=1}^{s}(\hat{y}_i - \hat{\hat{y}}_i)^2}{\sum_{i=1}^{s}(\hat{y}_i - \bar{\hat{y}})^2}, \tag{2.32}$$

  where the mean squared difference is calculated over a defined neighborhood size, s, and the squared difference between the observed field and its mean represents the variance of the observed field. FSS values range from 0 to 1, with higher scores indicating better agreement between predicted and observed fields at the specified spatial scale.

- The Continuous Ranked Probability Score (CRPS) [Broecker and Smith, 2007] is a metric used to evaluate the accuracy of probabilistic forecasts. For a given forecast probability distribution F and the true outcome $\hat{y}$, the CRPS is calculated as follows:

$$\text{CRPS}(F, \hat{y}) = \int_{-\infty}^{\infty} [F(z) - \mathbf{1}(z \geq \hat{y})]^2 \text{dz}. \tag{2.33}$$

  Here, $F(z)$ is the cumulative distribution function (CDF) of the forecast distribution at point z. $\mathbf{1}(\cdot)$ is the indicator function, which equals 1 if the condition inside is true and 0 otherwise. y is the observed value. The CRPS measures the discrepancy between the forecast distribution and the observed value. A lower CRPS indicates a better agreement between the forecast and the actual outcome. This metric is particularly useful for assessing the reliability of probabilistic forecasts in various fields, including weather and climate predictions.

2.4 Emulation

Emulation in the context of climate or weather refers to replacing whole models or model parts and is also referred to as surrogate modeling. Emulation is a conceptually simple task, but there are a few details that matter, e.g. what data to learn from, the amount of data needed, the choice of architecture, and the importance of physical consistency. We here go through the different kinds of tasks that can be considered, the common architectures and metrics used.

2.4.1 Different Tasks

Depending on which part of the numerical/physical model we want to replace and what kind of data we train on, there are different setups for emulation tasks.

Sub vs. Full Model Emulation With emulation, we can either replace a full model [Watson-Parris et al., 2022] or a model part [Silva et al., 2020b]. Whereas full model emulation usually implies that a more complex function needs to be learned, thus requiring a more complex architecture, sub-model emulation comes with its own challenges. Whereas the full model emulation usually implies that a more complex function needs to be learned here and a more complex architecture is thus required, sub-model emulation comes with its own challenges. If a submodel is replaced by an ML emulator it often leads to instabilities when interacting with the physical model. Here it is especially important to obey physical laws to achieve successful coupled runs.

Direct Replacement vs. Higher-Fidelity Model One possibility of emulation is to replace a model (part) by training on input and output pairs generated by running the original model. This does not increase the accuracy of representing any subprocess but is targeted to increase only the simulation speed with the emulator. Another possibility that can provide a better representation of processes is when the emulator is trained on a more sophisticated model's data [Rasp et al., 2018a] than the model being replaced or even directly learns from observations [Nair and Yu, 2020]

Offline vs. Online Model We distinguish between offline emulation and online emulation. Offline emulation describes running an emulator as a standalone model, without feeding its output back into a bigger model. Online emulation includes coupling the emulator to an ESM, that uses the emulators' predictions for different submodels within the ESM. Even though the final application of an emulator is to be an online model, it is important to consider the difference between online and offline models, especially when it comes to evaluation.

2.4.2 Data

First, data used for emulation has to address the general points for datasets in ML formulated in Section 2.1.3: the domain relevance is automatically given when training directly on data of a model we are trying to replace.

For training on higher-resolution model data or even observations though, it needs to be considered carefully that the data represents the exact process we are emulating. A very crucial point is the data diversity, if there are not enough cases represented in the training data the emulator can easily get unstable when integrated back into the climate model. In most emulation cases the training data is simulated data, which gives to possibility to generate lots of data points, so quantity is rarely the limit in emulation. The data splitting in train-val-test subsets needs to be done by considering the final application: will the emulator encounter new locations, new times, and new pressure levels when coupled? The split should resemble this. Data preprocessing is important, especially when multiple variables are predicted that may span different ranges. In order to have a useful optimisation during training where all variables are represented equally in the combined loss function normalisation or standardisation is necessary.

No matter what emulation task is considered training and offline evaluation data can come from different sources, it can be the exact inputs and outputs as the original, or additional input data that help the ML model, it can be data from a more complex or higher-res model or directly observational data, being the closest to the ground truth.

2.4.3 Architectures

Depending on the structure of the underlying data, whether it is tabular data, gridded spatial data, time series, or spatio-temporal data the architectural preferences vary. Another big factor is whether a probabilistic or deterministic prediction is desired.

For Tabular Data For deterministic architectures learning from tabular data linear regression, random forests, gradient boosting, or fully-connected neural networks are the most common choices [Silva et al., 2020b]. Is the problem probabilistic but tabular or low-dimensional Gaussian processes are a popular tool too [Vicent et al., 2023].

For Spatial Data For spatial data, CNNs are the most common choice [Wider et al., 2023]. If probabilistic predictions are beneficial the CNN as a generator might be extended with a discriminator network to obtain a GAN [Brochet et al., 2023]. If the data is not on a regular grid, graph neural networks [Bronstein et al., 2016] have recently emerged as a popular architecture [Ngo et al., 2023].

For Temporal Data If time series are emulated an RNN structure can be incorporated to account for the history. Here, LSTMs are a common choice as well as GRUs. More recently transformer and reservoir computing [Arcomano et al., 2023] architectures have been employed for time-series emulation too. If the data has both temporal and spatial structure combination of RNNs and CNNs are common choices as well as more recently transformer architectures.

As we work with tabular data and require a deterministic prediction for our emulation task, we consider linear regression, random forests, gradient boosting, or fully-connected neural networks as architectural choices (see Chapter 5).

2.4.4 Metrics

Here we focus on metrics for tabular, deterministic target data, as that is the only emulation case we are considering. To evaluate emulation often different metrics than for downscaling are used while keeping the standard RMSE, MAE, and bias metrics. Additional metrics include most commonly the R^2-score. If the target includes multiple variables on different scales, metrics like RMSE, MAE, and bias need to be either calculated individually or on normalized values to be able to aggregate in one score. The R^2-score comes with the advantage of being dimensionless and independent from linear scaling.

R^2-Score The coefficient of determination, often denoted as R^2-score, is a metric used to assess the goodness of fit of a regression model. It quantifies the proportion of the variance in the dependent variable that is predictable from the independent variables given by

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}. \tag{2.34}$$

Here n is the number of samples, $\hat{y}_i$ is the true value, y_i is the predicted value and $\bar{y}$ is the mean of the true values. The numerator represents the sum of squared differences between the target and predicted values, while the denominator represents the sum of squared differences between the true values and their mean. The R^2-score has a maximum one and can be arbitrarily negative. A zero already indicates that the model does not explain any variability, and one indicates perfect prediction.

Other metrics for emulation can provide insights into the violation of physical constraints, such as the negative fraction of naturally positive quantities or a violation of mass (see Section 5.4).

2.5 Atmospheric Aerosols

Atmospheric aerosols are tiny solid or liquid particles suspended in the air. They play a crucial role in Earth's atmosphere and climate system. These aerosols originate from various sources such as natural processes like volcanic eruptions, wildfires, and sea spray, as well as human activities including industrial emissions and vehicle exhaust. Aerosols come in diverse forms including dust, soot, and sulfates. Once released into the atmosphere, aerosols can scatter and absorb sunlight, influencing the planet's energy balance. Additionally, they serve as nuclei for cloud formation, affecting cloud properties and precipitation patterns. Understanding the distribution, composition, and behavior of atmospheric aerosols is essential for accurately modeling and predicting climate change, air quality, and weather patterns.

We will discuss below in more detail how aerosols influence the climate and in which way they are represented in current climate models.

2.5.1 Aerosol-Climate Interactions

Aerosol radiative forcing is the predominant source of uncertainty in understanding the anthropogenic impact on the current climate [Bellouin et al., 2020]. The cooling effect induced by aerosols masks a portion of the positive radiative forcing attributed to greenhouse gas emissions, and potential future measures aimed at reducing air pollution might unveil a stronger observed warming. Aerosols influence climate through two primary mechanisms: aerosol-radiation interactions and aerosol-cloud interactions [IPCC, 2013].

Aerosol-Radiation Interactions The behavior of aerosols in the atmosphere involves the scattering or absorption of radiation, depending on the composition of the particles. Notably, black carbon aerosols originating from fossil fuel combustion exert a warming effect by efficiently absorbing radiation, while sulfates released from volcanic eruptions induce cooling by exhibiting lower absorption and predominantly scattering characteristics.

Clouds Clouds play an important role in the Earth's radiation budget, covering about two-thirds of the Earth's surface. Clouds, made of tiny water

droplets or ice crystals forming around particles like salt or dust, significantly impact Earth's climate by affecting the planet's energy balance in multiple ways [Forster et al., 2021]. Clouds reflect a significant portion of the incoming short-wave radiation from the sun back into space. This is known as the albedo effect, which has a cooling influence on the Earth's surface and lower atmosphere. The extent of this cooling effect depends on the cloud type, altitude, and thickness. For example, thick, bright clouds (such as cumulus or stratocumulus) are more effective at reflecting sunlight than thin, high clouds. Clouds absorb long-wave radiation emitted from the Earth's surface and then re-emit it in all directions, including back towards the surface. This process, often referred to as the greenhouse effect, results in a warming effect because it traps heat in the atmosphere. High, thin clouds are particularly effective in trapping outgoing long-wave radiation, contributing to a net warming effect. In contrast, low clouds have a lesser effect on long-wave radiation compared to their impact on short-wave radiation.

Aerosol-Cloud Interactions Aerosols, by acting as cloud condensation nuclei (CCN), can significantly alter cloud properties. In a CCN-limited regime, elevated aerosol concentrations lead to an increased number of CCN, resulting in a greater yet smaller-sized population of cloud droplets for a fixed volume of water. The presence of smaller droplets enhances the albedo of clouds [Twomey, 1974] and has the potential to extend the lifetime of clouds [Albrecht, 1989].

2.5.2 Aerosols in Climate Models

Many climate models commonly regard aerosols as external variables—input once and maintained as constants throughout the entire modeling process. In instances where models do incorporate aerosol properties, the distinction between various aerosol types may be limited, often considering only the aggregate mass. To achieve greater precision in aerosol representation, sophisticated aerosol-climate modeling systems, for instance ICON-HAM, have been devised. Detailed models come with the cost of increased computational time: ICON-HAM can be run at 150 km resolution for multiple decades. But to run storm-resolving models, for example, the goal is ideally a 1km horizontal grid resolution and still be able to produce forecasts up to a few decades. If we want to keep detailed aerosol descriptions, a significant speedup of the aerosol model is needed.

2.5.3 Aerosol Microphysics Model M7

The M7 model [Vignati et al., 2004] employs seven log-normal modes to characterize aerosol properties, wherein particle sizes are distributed across four modes: nucleation, aitken, accumulation, and coarse modes. The aitken, accumulation, and coarse modes can exist in either soluble or insoluble states. M7 encompasses various processes, including nucleation, coagulation, condensation, and water uptake, which result in the redistribution of particle numbers and mass among the distinct modes. Moreover, M7 accounts for five distinct aerosol types—Sea salt (SS), sulfate (SO4), black carbon (BC), primary organic carbon (OC), and dust (DU). It is important to note that M7 operates independently for each grid box, without explicitly modeling spatial relationships. A full list of input and output variables can be found in the appendix (see Appendix B.2).

2.5.4 Aerosol-Climate Model ICON-HAM

The ICON (ICOsahedral Non-hydrostatic) model [Zängl et al., 2015] is a climate and weather prediction model developed jointly by the German Weather Service (Deutscher Wetterdienst, DWD) and the Max Planck Institute for Meteorology (MPI-M). As a global model, ICON employs an innovative icosahedral grid system to cover the Earth's surface, enabling a uniform resolution across the globe and avoiding the pole problem associated with traditional latitude-longitude grids. This non-hydrostatic[4] model is designed to simulate atmospheric processes with high precision across a wide range of scales, from global weather patterns down to localized extreme weather events. ICON-HAM [Salzmann et al., 2022] integrates the ICON model with the sophisticated aerosol model HAM [Stier et al., 2004], wherein the microphysical core contains the M7 model.

[1]Throughout the thesis, the following abbreviations will be employed: NS=nucleation soluble, KS/KI=aitken soluble/insoluble, AS/AI=accumulation soluble/insoluble, CS/CI=coarse soluble/insoluble

[4]The term "non-hydrostatic" refers to a class of atmospheric models that do not assume hydrostatic equilibrium in their equations of motion.

Chapter 3

Constraining Deep Learning Architectures

In this chapter, we describe our developed constraining methodologies. We introduce the notation for the generalized linear constraints we are considering, including equality and inequality constraints. We then show how the given constraints can be incorporated both in a soft and in a hard manner. We discuss the advantages and drawbacks of soft constraints. For hard constraints, we go through the several different layers we developed, such as a completion layer, an additive constraint layer, a multiplicative constraint layer, and a softmax constraint layer for equality constraints and a correction layer for inequality constraints. For most cases, we consider incorporating either equality or inequality constraints at a time, but we conclude with a brief discussion of combining the two.[1]

We consider the DL setting, recalling the ML formulation from Section 2.1.2, where $x \in \mathbb{R}^{n_{\text{in}}}$ is the input vector and $y \in \mathbb{R}^{n_{\text{out}}}$ the final output vector of the NN. The NN is given by a function $f_\theta : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}}$, parametrized by the weights and biases $\theta \in \mathbb{R}^l$, so $f_\theta(x) = y$. Given the targets $\hat{y} \in \mathbb{R}^{n_{\text{out}}}$ our learning task is solving following optimisation problem for all training pairs $(x^i, \hat{y}^i)_{i=1,\dots,n}$ for a training set of size n

$$\min_{\theta \in \mathbb{R}^l} \sum_{i=1}^{n} \mathcal{L}(f_\theta(x^i), \hat{y}^i). \tag{3.1}$$

$\mathcal{L}$ is some cost function with the mean-squared error (MSE) being the most common choice and used throughout this thesis as well. We only consider

[1]Parts of this chapter have been published in P. Harder, A. Hernandez-Garcia, V. Ramesh, Q. Yang, P. Sattegeri, D. Szwarcman, C. Watson, and D. Rolnick. Hard-constrained deep learning for climate downscaling. *Journal of Machine Learning Research*, 24(365):1–40, 2023a

the supervised learning task for constraining in this thesis, which means we always have the target vector $\hat{y}$ given. The introduced constraining methods are also adjustable for unsupervised tasks.

3.1 Constraining Setup

For this work, we focus on a generalized linear subset of constraints, as it covers all our application cases and can be incorporated as hard constraint layers. We introduce notations for both inequality constraints (noted with **in**) and equality (noted with **eq**) constraints simultaneously.

Let $(I_j^{(\text{eq})})_{j=1,\dots,n_p}$ and $(I_j^{(\text{in})})_{j=1,\dots,m_p}$ be partitions of $\{1,\dots,n_{\text{out}}\}$, dividing the output variables into n_p and m_p disjoint subsets. This partitioning is done to be able to enforce different constraints per subset. We define $k_j := |I_j|$ as the number of variables per subset.

Let $g_i^{(\text{eq})}, g_i^{(\text{in})} : \mathcal{D} \subset \mathbb{R} \to \mathbb{R},\ i \in I_j$ be invertible functions and $h_j^{(\text{in})}, h_j^{(\text{eq})} : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}$ arbitrary functions. The set of equality constraints is given by

$$\sum_{i \in I_j^{(\text{eq})}} g_i^{(\text{eq})}(y_i) + h_j^{(\text{eq})}(x) = 0 \tag{3.2}$$

for each $j = 1,\dots,n_p$. This means for each subset of the partition we have one constraining equation.

The set of inequality constraints is given by

$$\sum_{i \in I_j^{(\text{in})}} g_i^{(\text{in})}(y_i) + h_j^{(\text{in})}(x) \geq 0 \tag{3.3}$$

for each $j = 1,\dots,m_p$.

For $F, g : \mathbb{R}^{n_{\text{out}}} \to \mathbb{R}^{n_p}$ with $F^{(\text{eq})}(y) = (\sum_{i \in I_1^{(\text{eq})}}, \dots, \sum_{i \in I_{n_p}^{(\text{eq})}})^T$, $g = ((g_i^{(\text{eq})})_{i \in I_1^{(\text{eq})}}, \dots, (g_i^{(\text{eq})})_{i \in I_{n_p}^{(\text{eq})}})^T$ and $h^{(\text{eq})} = (h_1^{(\text{eq})}, \dots, h_{n_p}^{(\text{eq})})^T$ we can write Equation (3.2) more compactly in vectorized form:

$$F^{(\text{eq})}(g^{(\text{eq})}(y)) + h^{(\text{eq})}(x) = 0 \tag{3.4}$$

and analogously for the inequality contraints Equation (3.3,

$$F^{(\text{in})}(g^{(\text{in})}(y)) + h^{(\text{in})}(x) \geq 0. \tag{3.5}$$

It is assumed that for the true dataset, the constraints are satisfied (or at least close to being satisfied), i.e., for target $\hat{y}$

$$F^{(\text{eq})}(g^{(\text{eq})}(\hat{y})) + h^{(\text{eq})}(x) = 0$$

and

$$F^{(\text{in})}(g^{(\text{in})}(\hat{y})) + h^{(\text{in})}(x) \geq 0.$$

There can be cases where the constraints are not exactly, but just approximately satisfied in the training data. Unless they are too far from satisfaction, it can still be useful to enforce constraints (see Section 4.2). This can be formulated as

$$F^{(\text{eq})}(g^{(\text{eq})}(\hat{y})) + h^{(\text{eq})}(x) = \epsilon$$

and

$$F^{(\text{in})}(g^{(\text{in})}(\hat{y})) + h^{(\text{in})}(x) \geq -\epsilon,$$

for small, positive values of ϵ.

3.2 Soft Constraining

Given equality and inequality constraints, both can be added as additional terms to the loss function, resulting in a new loss $\mathcal{L}^{(\text{constrained})}$

$$\begin{aligned}\mathcal{L}^{(\text{constrained})}(y, \hat{y}) = \mathcal{L}(y, \hat{y}) \\ + \mu \cdot ||F^{(\text{eq})}(g^{(\text{eq})}(y)) + h^{(\text{eq})}(x)|| \\ + \gamma \cdot ||\text{ReLU}(-(F^{(\text{in})}(g^{(\text{in})}(y)) + h^{(\text{in})}(x)))||.\end{aligned} \tag{3.6}$$

Here $||\cdot||$ is a norm, usually the L^2 norm. The first term on the right-hand side is the standard supervised regression loss such as MSE, the second term penalizes violations of the equality constraints, and the third term penalizes inequality constraint violations, any positive values of $-(F^{(\text{in})}(g^{(\text{in})}(y)) + h^{(\text{in})}(x)))$, i.e. any negative values of $F^{(\text{in})}(g^{(\text{in})}(y)) + h^{(\text{in})}(x))$.

Often, we only consider equality constraints or inequality constraints one at a time and optimize $\mathcal{L}^{(\text{eq})}$

$$\mathcal{L}^{(\text{eq})}(y, \hat{y}) = \mathcal{L}(y, \hat{y}) + \mu \cdot ||F^{(\text{eq})}(g^{(\text{eq})}(y)) + h^{(\text{eq})}(x)||, \tag{3.7}$$

or

$$\mathcal{L}^{(\text{in})}(y, \hat{y}) = \mathcal{L}(y, \hat{y}) + \gamma \cdot ||\text{ReLU}(-(F^{(\text{in})}(g^{(\text{in})}(y)) + h^{(\text{in})}(x)))||. \tag{3.8}$$

The ReLU function ($\text{ReLU}(x) = \max(0, x)$) is commonly used in DL and the most simple choice, but can be replaced with other functions that are zero for negative values, e.g. the softplus[2] function.

[2] $\text{softplus(x)} = \frac{1}{\beta}\log(1 + \exp(\beta \cdot x))$

It is important to note that the two penalizing terms do not depend on the target vector $\hat{y}$, this implies that such soft constraints can also be included in an unsupervised learning task. Additionally, when utilizing penalizers in a supervised context that can include constraints depending on the target vector as well, given that the regularisation term is only active during training. Including the target vector for constraints is not possible for hard constraining methods as proposed here.

The type of equation or inequality that is taken into account here can be arbitrarily complex, it can be e.g. a differential equation and is not limited to the generalized linear case we are considering here.

Tuning Here, a challenge is to tune the parameters μ and γ. We experience a high sensitivity for the choices. A good starting point to choose them so the different loss terms have similar magnitudes compared to the MSE loss term. On the one hand, if the values of μ or γ are chosen too high it is common to converge to a trivial solution such as $y = x$ (for special cases). On the other hand, if they are chosen too small, the constraints remain significantly violated. One way to deal with this issue could be scheduling of μ and γ, where they start very small at the beginning of training and then get increased. Similarly to the common learning rate scheduling [Darken et al., 1992], this could take advantage of the changing loss landscape during the progression of training and is left for future work.

In summary, soft-constraining is a very flexible but potentially unstable constraining method, which does not give a guarantee on the constraints to be satisfied.

3.3 Hard Constraining Methods

We introduce different hard constraining methods, starting with multiple constraining methods that address equality constraints and after that one method that addresses inequality constraints. All hard constraining methods work on the following principle: A standard NN, $\tilde{f}_\theta : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}}$, architecture is applied and gives the intermediate prediction $\tilde{y} \in \mathbb{R}^{n_{\text{out}}} = \tilde{f}_\theta(x)$. The intermediate output is then modified in a final layer, the *constraint layer*, $c : \mathbb{R}^{n_{\text{out}}} \to \mathbb{R}^{n_{\text{out}}}$, producing the final output

$$c(\tilde{y}) = y.$$

The general constrained NN $f_\theta : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}}$ is given by

$$f_\theta = c \circ \tilde{f}_\theta.$$

We want to highlight that a strength of this is that it is indepedent of the architecture used for the NN. Another thing to note, is that c, the constraint layer, does not include any tunable parameter. This constrained layer could be extended to include learnable parameters, which is left for future exploration.

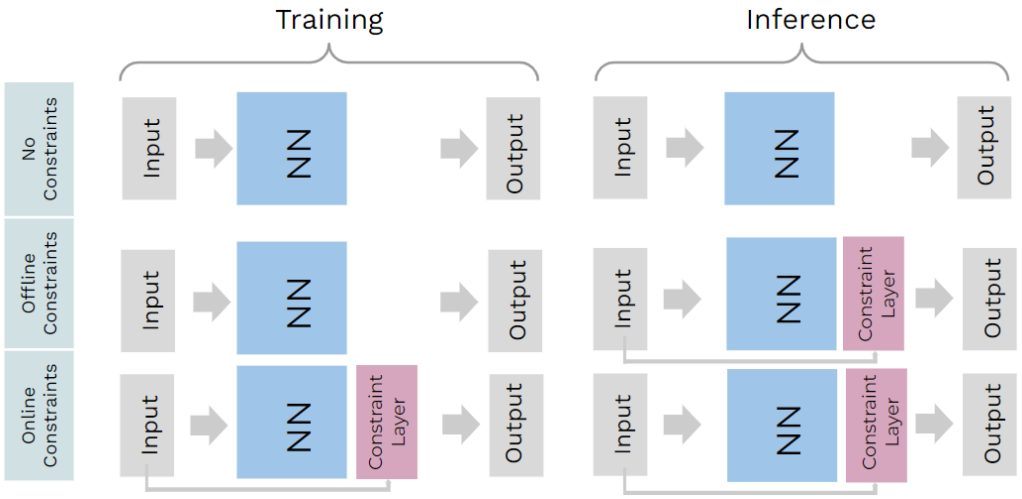

Figure 3.1: The difference between applying a constraint layer offline or online and unconstrained, standard training. The first row is the standard usage of a NN wihtout any constraints activated. The second row shows the offline application, where the constraint layer is only activated during inference. The third row then is the online application, where the constraint layer is active during both training and inference.

Constraint Formulation

Online vs. Offline Unlike soft constraints, hard constraints can be either *offline constraints* or *online constraints* (see Figure 3.1). An application offline means we train our model without any constraints (or potentially a soft constraint) and then only add the constraint layer for inference. An online constraint layer is part of the NN both during training and inference. Most of the constraint layers developed by us can be either applied offline or online, the criterion for that is that the constraining only applies minimal changes and the constraining methodology does not change an already perfect prediction, i.e. $\tilde{y} = \hat{y}$.

3.3.1 Additive Layer

For this and the next subsections we will consider only equality constraints and skip the **(eq)** in the notation for clarity.

The additive constraint layer (AddCL) readjusts the intermediate prediction $\tilde{y}$ by adding the term $-a\cdot(h(x)+F(\tilde{y}))$ to it, where $a := (1/k_1, \ldots, 1/k_{n_p})^T$. The output of the AddCL is defined as

$$y^{\text{AddCL}} := g^{-1}(\tilde{y} - a \cdot (F(\tilde{y}) + h(x)), \tag{3.9}$$

with using the vectorized formulation and g^{-1} as a notation for $((g_i^{-1})_{i\in I_1}, \ldots, (g_i^{-1})_{i\in I_{n_p}})^T$. If you consider the (common) case where $g \equiv \text{id}$ then $-\frac{1}{n}(h(x) + F(\tilde{y})) \approx 0$, for a good enough intermediate prediction. In that case we could apply AddCL offline.

Let us rewrite the AddCL in its detailed indices form

$$y_i^{\text{AddCL}} := g_i^{-1}(\tilde{y}_i - \frac{1}{k_j}(\sum_{k\in I_j} \tilde{y}_k + h_j(x))) \tag{3.10}$$

for $i \in I_j$ and $j = 1, \ldots, n_p$.

By construction, AddCL's output satisfies Equation (3.2):

$$\begin{aligned}
\sum_{i\in I_j} g_i(y_i^{\text{AddCL}}) &= \sum_{i\in I_j}(\tilde{y}_i - \frac{1}{k_j}(\sum_{k\in I_j} \tilde{y}_k + h_j(x))) \\
&= \sum_{i\in I_j} \tilde{y}_i - \sum_{i\in I_j} \frac{1}{k_j} \sum_{k\in I_j} \tilde{y}_k - \sum_{i\in I_j} \frac{1}{k_j} h_j(x) \\
&= \sum_{i\in I_j} \tilde{y}_i - \sum_{k\in I_j} \tilde{y}_k - h_j(x) \\
&= -h_j(x).
\end{aligned}$$

3.3.2 Multiplicative Layers

The multiplicative constraint layer (MultCL) works similarly to AddCL, though the adjustment is done by rescaling, multiplication, and not addition

$$y_i^{\text{MultCL}} := g_i^{-1}(\tilde{y}_i \cdot \frac{-h_j(x)}{\sum_{k\in I_j} \tilde{y}_k}) \tag{3.11}$$

for $i \in I_j$ and $j = 1, \ldots, n_p$. In its vectorized formulation the multiplicative constraint can be written as

$$y^{\text{MultCL}} := g^{-1}(\tilde{y} \cdot \frac{-h(x)}{F(\tilde{y})}). \tag{3.12}$$

Again, by construction MultCL's output satisfies Equation (3.2):

$$\begin{aligned}\sum_{i\in I_j} g_i(y_i^{\text{MultCL}}) &= \sum_{i\in I_j} g_i(g_i^{-1}(\tilde{y}_i \cdot \frac{-h_j(x)}{\sum_{k\in I_j}\tilde{y}_k})) \\ &= \sum_{i\in I_j} \tilde{y}_i \cdot \frac{-h_j(x)}{\sum_{k\in I_j}\tilde{y}_k} \\ &= -h_j(x).\end{aligned}$$

The multiplicative approach can be extended by introducing a function $t : \mathbb{R} \to \mathbb{R}$ and we obtain

$$y = g^{-1}(t(\tilde{y}) \cdot \frac{-h(x)}{F(t(\tilde{y}))}). \tag{3.13}$$

If $-h(x)$ and $t(y)$ are positive for all inputs, this constraint layer not only guarantees the satisfaction of Eq. (3.2), but also the positivity of your output.

One possible and positive choice of the function t is the exponential function. Including that gives a constraint layer that is an adaptation of the softmax layer (a common activation function for multi-class classification problem). Here, the standard softmax layer is scaled with $-h(x)$ and we refer to it as the softmax constraints layer (SmCL)

$$y^{\text{SmCL}} := g^{-1}(\exp(\tilde{y}) \cdot \frac{-h(x)}{F(\exp(\tilde{y}))}) \tag{3.14}$$

SmCL can include an either tunable or learnable parameter c that scales the exponential

$$y^{\text{SmCL},c} = g^{-1}(\exp(c \cdot \tilde{y}) \cdot \frac{-h(x)}{F(\exp(c \cdot \tilde{y}))}). \tag{3.15}$$

This is not explored in this work, but left for future experimentation.

3.3.3 Completion Method

Unlike readjustment layers, the completion method does not treat every variable y_i within a subset of the partition in the same way. We need to first choose an index $i_c \in I_j$, then the completion layer (CompL) is defined as

$$y_{i_c}^{\text{CompL}} := g_{i_c}^{-1}(-(\sum_{k\in I_j\setminus\{i_c\}} g_k(\tilde{y}_k) + h_j(x))) \tag{3.16}$$

for $j = 1, \ldots, n_p$. The other variables $\tilde{y}_i$ for $i \in I_j \setminus \{i_c\}$ stay unchanged: $y_i^{\text{CompL}} = \tilde{y}_i$.

This, again by construction, satisfies Eq. (3.2):

$$\begin{aligned}\sum_{i\in I_j} g_i(y_i^{\text{CompL}}) &= g_{i_c}(y_{i_c}^{\text{CompL}}) + \sum_{i\in I_j\setminus\{i_c\}} g_i(y_i^{\text{CompL}}) \\ &= g_{i_c}(g_{i_c}^{-1}(-(\sum_{k\in I_j\setminus\{i_c\}} g_k(y_k) + h_j(x)))) + \sum_{i\in I_j\setminus\{i_c\}} g_i(y_i) \\ &= -(\sum_{k\in I_j\setminus\{i_c\}} g_k(y_k) + h_j(x)) + \sum_{i\in I_j\setminus\{i_c\}} g_i(y_i) \\ &= -h_j(x).\end{aligned}$$

We set one variable per subset as the residual of the constraint depending on the other variables. This method does not depend on any intermediate prediction $\tilde{y}$, it can be both applied offline and online.

There are multiple ways of choosing the index i_c:

- Choose completely **randomized** (every forward pass in the NN it can be different).
- Choose **random once** and keep it fixed.
- Choose index of **worst performing** variable.
- Choose index such that additional **inequality constraints** are satisfied (if possible).

3.3.4 Correction Method

Now, we switch to incorporating an inequality constraint with a hard constraint layer. Let us define $R_j(x, y) := \sum_{i\in I_j} g_i(y_i) + h_j(x)$ as the contraint residual. The correction layer (CorL) is then defined as the following (skipping **(in)** in the notation)

$$y_i^{\text{CorL}} := \begin{cases} \tilde{y}_i & R_j(x,y) \geq 0 \\ g_i^{-1}(-\frac{1}{k_j} h_j(x)) & \text{else} \end{cases}$$

for $i \in I_j$ and $j = 1, \ldots, n_p$.

For the most commonly appearing inequality constraints, where there is no sum included, i.e. $I_j = \{j\}$ for $j = 1, \ldots, n_p$,

$$g_j(y_j) + h_j(x) \geq 0. \tag{3.17}$$

Same as for the soft constraining, we then utilize the ReLU function:

$$y_j^{\text{CorL}} = g_j^{-1}(\text{ReLU}(g_j(\tilde{y}_j)) + h_j(x)) - h_j(x)) \tag{3.18}$$

for $j = 1, \ldots, n_p$.

In vectorized form, this can be written as

$$y^{\text{CorL}} = g^{-1}(\text{ReLU}(g(y) + h(x)) - h(x)) \tag{3.19}$$

The same as the completion method, it can be applied offline and online. The offline application is a commonly used method referred to as clipping.

3.4 Combining Methods

There are cases where it is necessary or beneficial to have both inequality and equality constraints including the same variables. For this case, we can combine the proposed constraints methods. Soft constraints can, as discussed before, easily include multiple constraints by adding several terms to the loss function.

Soft+Hard Any hard equality constraint layer can be combined with a soft inequality constraint, and any hard inequality constraint layer can be combined with the soft equality regularisation loss. We apply e.g. a aerosol mass conservation soft contraint together with a correction layer for positive masses in our emulation application.

SmCL For simple enough inequality constraints such as positivity of the output, SmCL can enforce both equality constraints and positivity. We use this e.g. to conserve water mass as an equality constraint and then also ensure water mass positivity alongside.

There are more ways to combine equality and inequality constraints, their implementation and experimentation are left for future exploration, but we will briefly describe the ideas behind them. We can combine correction and completion methods.

CorL+CompL One option is that we apply the correction layer first and achieve guaranteed inequality constraints. CompL then leaves us the choice of index i_c, here we can choose i_c (if it exists) so that the inequality constraints are still satisfied for y_{i_c}. We iterate over $i \in I_j$ until we find i_c such that $y_{i_c}^{\text{CompL}}$ satisfies the inequality constraints.

CompL+redistribtion of residual Another possibility is enforcing the equality constraint first, via CompL, then enforcing the inequality constraint by distributing the residual among different variables such that they stay in the feasible region with respect to the inequality constraints.

Chapter 4

Physics-Constrained Downscaling

After we have developed the constraint methodology in Chapter 3, we now apply this new tool to real-world application tasks. Introduced in Chapter 2, we will start with the task of downscaling, increasing the resolution of climate data. We show what constraints exist naturally for downscaling. Taking the constraining tools from the previous chapter, we reformulate them for our specific application case. We then describe the datasets, such as the European Center for Medium-Weather Forecast (ECMWF) reanalysis data water content, Weather Research and Forecast model data, and Norwegian ESM data. We go through the architectures, including convolutional neural networks, generative adversarial neural networks, and recurrent neural networks that are being employed here. After the experimental setup, we conclude by going through the results, demonstrating the successful application of our constraint layers.[1]

4.1 Constrained Downscaling

When modeling physical quantities such as precipitation or water mass, principled relationships such as mass conservation can naturally be established between low-resolution and high-resolution samples. Here, we develop a setup in which we can utilize the methods introduced in the previous chapter for downscaling. We first give a technical description of the downscaling setup

[1]Parts of this chapter have been published in P. Harder, A. Hernandez-Garcia, V. Ramesh, Q. Yang, P. Sattegeri, D. Szwarcman, C. Watson, and D. Rolnick. Hard-constrained deep learning for climate downscaling. *Journal of Machine Learning Research*, 24(365):1–40, 2023a

consired here and then restate the constraint layers for this specific case.

4.1.1 Downscaling Setup

We consider downscaling as a supervised learning task, which is the most prominent case, even though there are some unsupervised cases, where there are no matching pairs of LR and HR samples [Groenke et al., 2020]. The input vector is the low-resolution sample $x \in \mathbb{R}^{n_{\text{in}}}$. It is usually a two-dimensional image $x \in \mathbb{R}^{k_{\text{in}}} \times \mathbb{R}^{k_{\text{in}}}$, with $k_{\text{in}}^2 = n_{\text{in}}$, for simplicity, we use the equivalent one-dimensional formulation. The output vector is the HR version of x, $y \in \mathbb{R}^{n_{\text{out}}}$ with $n_{\text{out}} = N^2 \cdot n_{\text{in}}$, where $N \in \mathbb{N}$ is the *upsampling factor*. Any pixel $x_j, y_i \in \mathbb{R}$ describes the value of a physical quantity (e.g. temperature, rain mass) at a given location. For each input pixel $x_j \in \mathbb{R}$ there is a set of output pixels, $(y_i)_{i \in I_j}$, called *super-pixel*, covering the same area. I_j is the set of indices corresponding to the location of x_j in the HR, $|I_j| = N^2 := n$.

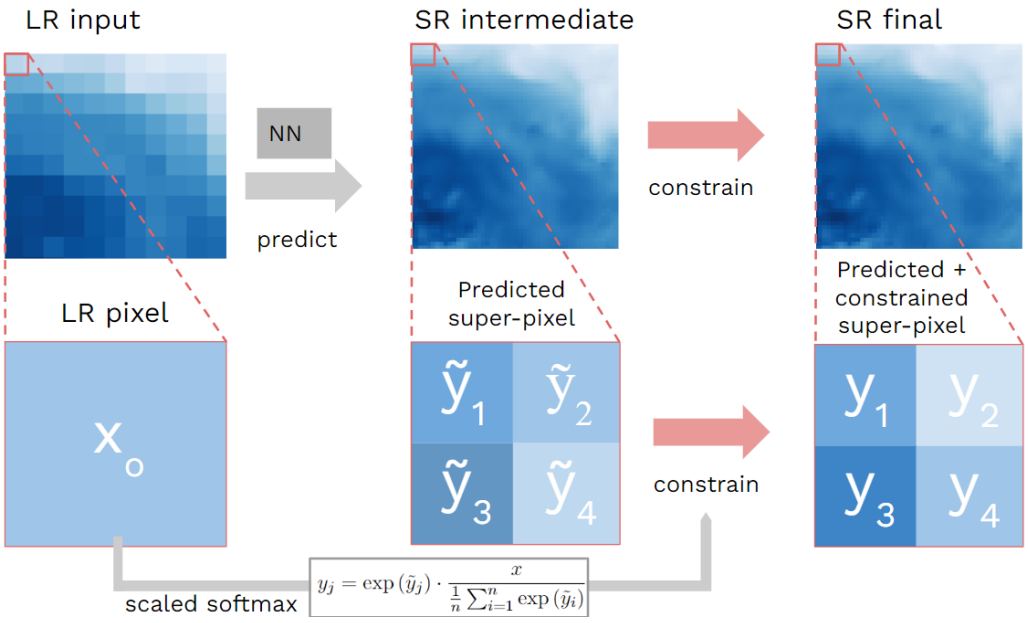

Figure 4.1: Our Softmax constraint layer (SmCL) is shown for one input pixel x and the corresponding predicted 2×2 super-pixel for the case of 2 times upsampling. This layer is added at the end of a NN and enforces given constraints guaranteed by construction. Besides equality constraints, it enforces the positivity of the outputs.

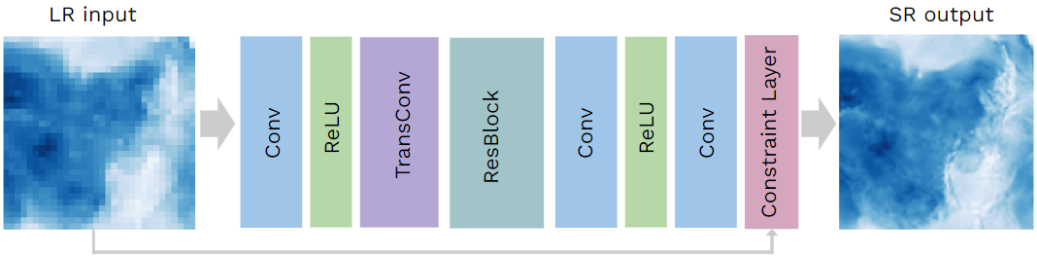

Figure 4.2: The CNN architecture used here for two times upsampling, including the constraint layer (in red). The LR input is passed to the last layer, the constraint layer, to enforce the constraint and produce a consistent HR output.

4.1.2 Constraint Formulation

Let us briefly recall the constraint formulation of the previous chapter. The equality constraints are given by

$$\sum_{i \in I_j^{(\mathrm{eq})}} g_i^{(\mathrm{eq})}(y_i) + h_j^{(\mathrm{eq})}(x) = 0 \tag{4.1}$$

for each $j = 1, \ldots, n_p$, and the inequality constraints by

$$\sum_{i \in I_j^{(\mathrm{in})}} g_i^{(\mathrm{in})}(y_i) + h_j^{(\mathrm{in})}(x) \geq 0 \tag{4.2}$$

for each $j = 1, \ldots, m_p$.

For downscaling, the most commonly appearing constraint is a consistency constraint between LR and predicted HR, i.e., SR. For water mass (per area), this consistency means conserving the LR water mass when predicting the HR sample. In the above form, that is

$$\sum_{i \in I_j} y_i - n \cdot x_j = 0, \tag{4.3}$$

for each $j = 1, \ldots, n_{\mathrm{in}}$. The function h_j here is the negative projection on the $j-$th entry scaled by the square root of the upsampling factor, n, $h_j^{(\mathrm{eq})}(x) = -n \cdot x_j$ and $g_i^{(\mathrm{eq})}$ the identity.

For cases where only positive values are physically plausible, the inequality constraints are

$$y_i \geq 0 \tag{4.4}$$

for each $i = 1, \ldots, n_{\mathrm{out}}$.

Soft constraining Soft constraining can be applied using an L^2-penalty of the difference between the low-res pixel and the mean of the super-pixel. For soft mass conservation we use the notation (soft m.)

$$\mathcal{L}^{(\text{eq})}(y, \hat{y}) = \mathcal{L}(y, \hat{y}) + \mu \cdot \frac{1}{n_{\text{in}}} \sum_{j=1}^{n_{\text{in}}} (\sum_{i \in I_j} y_i - n \cdot x_j)^2. \tag{4.5}$$

To encourage positivity, we can penalize $-y$, the soft negativity constrained (soft n.):

$$\mathcal{L}^{(\text{in})}(y, \hat{y}) = \mathcal{L}(y, \hat{y}) + \gamma \cdot \frac{1}{n_{\text{out}}} \sum_{i=1}^{n_{\text{out}}} \text{ReLU}(-y_i)^2. \tag{4.6}$$

The two different penalizers can also be combined and applied at the same time, soft negativity and mass constraining (soft n.+m.). Soft constraints penalize violations but do not give any guarantees for exact constraint satisfaction.

Readjusment Layers The previously introduced readjustment layers, AddCL, MultCL, and SmCL to achieve guaranteed equality constraints given by Equation (4.3) for downscaling are given by

$$y_i^{\text{AddCL}} = \tilde{y}_i - \frac{1}{n}(\sum_{k \in I_j} \tilde{y}_k - n \cdot x_j) \tag{4.7}$$

$$y_i^{\text{MultCL}} = \tilde{y}_i \cdot \frac{n \cdot x_j}{\sum_{k \in I_j} \tilde{y}_k} \tag{4.8}$$

$$y_i^{\text{SmCL}} = \exp(\tilde{y}_i) \cdot \frac{n \cdot x_j}{\sum_{k \in I_j} \exp(\tilde{y}_k)} \tag{4.9}$$

for each $i \in I_j^{(\text{eq})}$ and $j = 1, \ldots, n_{\text{in}}$, applying one constraint per super-pixel, x_j. To apply MultCL we have to ensure that none of the super-pixels are all zero. For downscaling a quantity as precipitation that might happen, other hard constraints need to be used instead or the variables need to be scaled (see below). The SmCL has the advantage of only predicting positive values, given that x is positive – a valuable feature for predicting physical quantities such as water mass.

Completion Layer The readjustment layers treat every sub-pixel in a super-pixel equally, whereas for the completion method, only one pixel is changed

$$y_{i_c}^{\text{CompL}} = -(\sum_{k \in I_j \setminus \{i_c\}} y_k - n \cdot x_j) \tag{4.10}$$

for a chosen $i_c \in I_j$ and each $j = 1, \ldots, n_{\text{in}}$. In this downscaling setting, i_c is alway chosen randomly, nit fixed, to not bias certain pixels.

Correction Layer For our positivity constraint, the correction layers simplify significantly; we directly discard any negative values of our intermediate prediction $\tilde{y}$

$$y^{\text{CorL}} = \text{ReLU}(\tilde{y}). \tag{4.11}$$

This correction can be applied during training and inference, or only during inference as post-processing. The correction method for this special case is a commonly applied technique already in super-resolution, usually offline and referred to as clipping.

Weighted Constraining There two common cases where a slightly more complicated formulation of constraints is need in downscaling

1. We train on normalized data (almost always the case)
2. Not all LR pixels cover the same area (the case for latitude-longitude grids)

Both cases can be addressed with a weighted formulation of staying in the general framework of

$$\frac{1}{n} \sum_{i \in I_j^{(\text{eq})}} \alpha_i y_i - x_j = 0 \tag{4.12}$$

Remark: the positivity constraint is not affected as long as we scale with a positive factor.

The readjustment layers to enforce the scaled equality constraints then are

$$y_i^{\text{AddCL}} = \frac{1}{\alpha_i}(\tilde{y}_i - (\frac{1}{n}(\sum_{k \in I_j} \tilde{y}_k - x_j)) \tag{4.13}$$

$$y_i^{\text{MultCL}} = \frac{1}{\alpha_i}(\tilde{y}_i \cdot \frac{n \cdot x_j}{\sum_{k \in I_j} \tilde{y}_k}) \tag{4.14}$$

$$y_i^{\text{SmCL}} = \frac{1}{\alpha_i}(\exp(\tilde{y}_i) \cdot \frac{n \cdot x_j}{\sum_{k \in I_j} \exp(\tilde{y}_k)}) \tag{4.15}$$

for each $i \in I_j$ and $j = 1, \ldots, n_{\text{in}}$. For normalisation to obtain data between $[0, 1]$, so $y_i \in [0, 1]$, α is the maximum of the whole dataset in the original scale. For this, it is assumed the minimum in the original dataset is zero. For an area-weighted constraining α_i is the area of the i^{th} sub-pixel of the j^{th} super-pixel in the output.

Global Constraints The constraint layers can be relaxed by increasing the constraint window size and constraining on the mean of the input pixels that fall into that window; this can then impose softer constraints. In the extreme case, this reduces the number of constraints to one and allows using a global constraint. The constraint for the global case is

$$\sum_{i=1}^{n_{\text{out}}} y_i - \sum_{j=1}^{n_{\text{in}}} x_j = 0 \tag{4.16}$$

and can be enforced via

$$y_i^{\text{AddCL}} = \tilde{y}_i - \frac{1}{n}(\sum_{k \in I_j} \tilde{y}_k - \sum_{j=1}^{n_{\text{in}}} x_j) \tag{4.17}$$

$$y_i^{\text{MultCL}} = \tilde{y}_i \cdot \frac{\sum_{j=1}^{n_{\text{in}}} x_j}{\sum_{k \in I_j} \tilde{y}_k} \tag{4.18}$$

$$y_i^{\text{SmCL}} = \exp(\tilde{y}_i) \cdot \frac{\sum_{j=1}^{n_{\text{in}}} x_j}{\sum_{k \in I_j} \exp(\tilde{y}_k))}. \tag{4.19}$$

4.2 Data

To assess our proposed approach, we employ a diverse collection of datasets, including both newly generated datasets and established ones. We create multiple datasets by leveraging the ERA5 reanalysis product, employing average pooling to construct the low-resolution (LR) inputs—a conventional methodology in climate downscaling studies [Serifi et al., 2021, Leinonen et al., 2021]. Additionally, we incorporate datasets derived from outputs of models such as the Weather Research and Forecasting model and the Norwegian Earth System Model, which provide authentic low-resolution simulation data paired with corresponding high-resolution data. Furthermore, we extend our evaluation to non-climatic datasets, specifically lunar satellite imagery. For a comprehensive overview of the various datasets utilized, we refer to Table 4.1.

ERA5 Dataset

The ERA5 dataset [Hersbach et al., 2020] is classified as a *reanalysis* product generated by the European Center for Medium-Range Weather Forecast (ECMWF). This dataset integrates model data with observational inputs from across the globe through a process known as data assimilation, which seeks to determine the most suitable physical model state that aligns with

Table 4.1: Here we show the different datasets we use to test our constraint layers. The names are given to identify the datasets throughout this chapter. Most datasets are based on ERA5 atmospheric water content data and LR is generated synthetically, we include different upsampling factors, an OOD case, and temporal datasets. Additional datasets include the moist static energy (MEn) dataset as well as WRF and NorESM model data. Lunar images give a non-climate application dataset.

Name	Source	Type	Dim. LR/HR
W2	ERA5	water cont.	(1,64,64)/(1,128,128)
W4	ERA5	water cont.	(1,32,32)/(1,128,128)
W8	ERA5	water cont.	(1,16,16)/(1,128,128)
W16	ERA5	water cont.	(1,8,8)/(1,128,128)
OOD	ERA5	water cont.	(1,32,32)/(1,128,128)
WT1	ERA5	water cont.	(3,32,32)/(3,128,128)
WT2	ERA5	water cont.	(2,32,32)/(3,128,128)
MEn	ERA5	water vapor liq. water temp.	(3,32,32)/(3,128,128)
WRF	WRF	temp.	(1,45,45)/(1,135,135)
NorESM	NorESM	temp.	(1,32,32)/(1,64,64)
Lunar	satell.	photons	(1,32,32)/(1,128,128)

observed conditions. ERA5 offers global, hourly data at a resolution of a $0.25° \times 0.25°$ regular latitude-longitude grid, translating to approximately 25 km per pixel in the mid-latitudes. Spanning from 1950 onwards, ERA5 provides comprehensive coverage of weather patterns and phenomena over the specified time period.

Total Water Content Dataset One chosen quantity we focus on is the total column water (W), given in kg/m^2, and describes the vertical integral of the total amount of atmospheric water content, including water vapor, cloud water, and cloud ice, but not precipitation.

Spatial SR Data To acquire our high-resolution data points, we extract a random 128×128 pixel image from each available time step, where each time step consists of a grid of dimensions 721×1440, with roughly 60,000 time steps accessible. For training, we randomly sample 40,000 data points, reserving

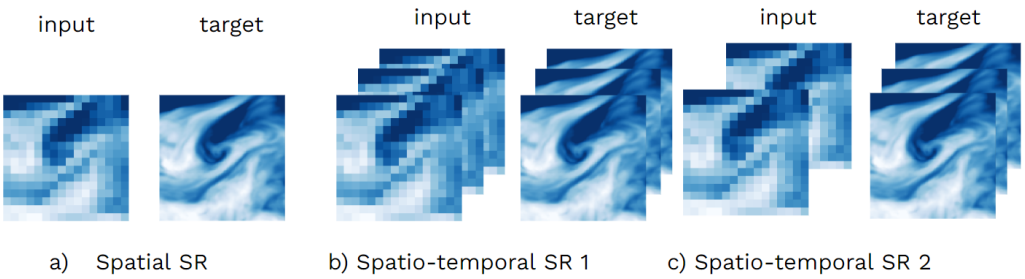

Figure 4.3: Samples of the three different dataset types used in this work. a) A data pair we use for our standard spatial super-resolution task. The input is an LR image, and the target is the HR version of that. b) A data pair to perform SR for multiple time steps simultaneously. The input is a time series of LR images, and the output is the same time series in HR. c) A data pair where SR is performed both temporally and spatially, with two LR time steps as input and three HR time steps as a target.

10,000 each for validation and testing purposes. To generate low-resolution counterparts, we compute the mean over $N \times N$ patches, where N represents our upsampling factor. An illustrative sample pair is depicted in Figure 4.3 a). This approach adheres to physical principles, as the conservation of water content dictates that the water density described within an LR pixel should equate to the average of the corresponding HR pixels. Additionally, in LR-modeled data like WRF data (as shown below), the modeled quantities in a low-resolution run approximate the mean of a high-resolution run, offering further validation for our coarsening strategy.

Spatio-Temporal Datasets To incorporate the temporal progression of our data, we establish two supplementary datasets. In the first dataset, a single sample encompasses three consecutive time steps, with identical time steps assigned to both input and target, albeit at different resolutions. This enables simultaneous spatial super-resolution across multiple time steps, as depicted in Figure 4.3 b). We designate three random 128×128 pixel regions per global image, yielding an equivalent number of instances as outlined previously. Similar to our previous approach, we randomly partition the data, and each time step undergoes downsampling via spatial averaging. Next, we construct a second dataset tailored for enhancing both spatial and temporal dimensions. Similarly to the previous dataset, we extract three images from a series of three consecutive time steps to generate our high-resolution

targets. For the low-resolution inputs, we reduce both the temporal and spatial dimensions. To decrease temporal resolution, we omit the intermediate (second) time step from each sample, performing subsampling. For spatial resolution reduction, we employ the same spatial averaging technique as before. These operations yield two low-resolution inputs, as illustrated in Figure 4.3 c). Notably, temporally coarse-graining via subsampling, rather than averaging, is implemented to prevent future information leakage into previous time steps.

Different Upsampling Factors As we create our LR counterpart ourselves by availability pooling, we can decide on the upsampling factor, keeping the HR at the same resolution and obtaining the LR, respectively. With that we construct cases for the upsampling of $2x$, $4x$, $8x$, and $16x$ to create the W2, W4, W8, and W16 datasets.

OOD Dataset In the datasets outlined earlier, the division into training, validation, and testing sets is performed randomly. To investigate the impact of our constraints on out-of-distribution generalisation, we construct a dataset with a temporal split. In this setup, we anticipate observing patterns in later time steps that deviate from previously observed distributions. Specifically, we train our model on older data and subsequently evaluate its performance on more recent years. For training, we utilize data from the years 1950 to 2000, while validation spans from 2001 to 2010, and the final testing phase covers the years 2011 to 2020. This temporal division enables us to assess the model's ability to generalize to unseen data distributions over time.

Energy Dataset Additionally sourced from the ERA5 data, we compile a second dataset featuring diverse physical variables, to include different constraints. This dataset is designed to conserve moist static energy and water masses while forecasting water vapor, liquid water content, and air temperature. The variables are extracted from the pressure level at 850hPa.

WRF Data

In Watson et al. [2020], the authors introduce a dataset derived from the advanced research version of the Weather Research and Forecasting (WRF) Model. This dataset encompasses hourly operational weather forecast information for Lake George in New York, USA, spanning from January 1, 2017, to March 20, 2020. Further details regarding the model and its configuration

are provided in Watson et al. [2020]. For our investigation, we focus on the temperature at 2m above ground level. Unlike the preceding datasets, this dataset does not include synthetic downsampling. Instead, it consists of two forecasts executed at different resolutions employing distinct physics-based parameterisations: one at a horizontal resolution of 9 km and another one at 3 km. Our objective is to predict the temperature field at the 3 km resolution using the 9 km resolution data. This approach builds upon the work by Auger et al. [2021], who conducted a similar analysis using the same dataset.

NorESM Data

Our NorESM dataset is derived from the second version of the Norwegian Earth System Model (NorESM2), developed by the NorESM Climate Modeling Consortium and based on the Community Earth System Model, CESM2. This dataset is constructed from two distinct runs: NorESM-MM, featuring a 1-degree resolution for model components, and NorESM2-LM, with a 2-degree resolution for atmosphere and land components. We focus on surface temperature (tas) data spanning from 2015 to 2100. For training, we utilize scenarios ssp126 and ssp585, while ssp370 is reserved for validation and ssp245 for testing purposes. Each scenario is cropped into 64×64 and 32×32 pixels, resulting in a dataset containing 12,000 data points for each scenario.

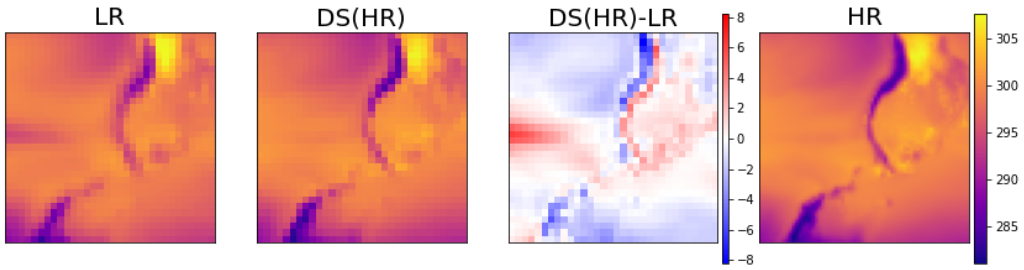

Figure 4.4: A sample from the NorESM temperature training dataset. We compare the low-resolution simulation to the downsampled high-resolution counterpart. It can be observed that the LR and the downsampled HR are significantly different, with an average violation of 2.48K.

Non-Climate Data: Lunar Imagery

Recent research [Delgano-Centeno et al., 2021] focusing on super-resolution for lunar satellite imagery has demonstrated the efficacy of deep learning techniques in enhancing captured data, potentially benefiting future lunar missions. To address the challenge of low-resolution imagery in regions such as the south pole, where high-resolution data is unavailable, a machine-learning-ready dataset has been curated. This dataset includes 220,000 images extracted from the Narrow-Angle Camera (NAC) imagery captured by NASA's Lunar Reconnaissance Orbiter (LRO); further details are provided in Delgano-Centeno et al. [2021]. In this study, we utilize a 4x upsampled version of the dataset to assess the efficacy of our constraint methodologies in enhancing super-resolution performance beyond the realm of climate science. The averaging process employed in this context is justified, as real low-resolution images are generated by aggregating photon counts in low-light regions.

Constraints in our Datasets

When predicting various physical quantities, we must account for different constraints. The majority of our datasets adhere to the downscaling constraints given by Eq. (4.3), which LR-HR pairs satisfy either approximately (for simulations conducted at both LR and HR with quantities adhering to physical conservation laws) or precisely (in instances where the LR version is created via average pooling). Further explanation on these constraints is provided in the subsequent subsections.

Water Content Conservation To predict the total column-integrated water content, we start with the low-resolution water content $Q^{(LR)}$ and aim to derive the super-resolved counterpart $Q^{(SR)}$. The downscaling constraint, or mass conservation constraint (Eq. 4.3), for each LR pixel $q^{(LR)}$ and its corresponding super-pixel $(q_i^{(SR)})_{i=1,\dots,n}$ is expressed as follows:

$$\frac{1}{n}\sum_{i=1}^{n} q_i^{(SR)} = q^{(LR)}. \tag{4.20}$$

Moist Static Energy Conservation One of our objectives involves forecasting column-integrated water vapor, liquid water, and temperature while ensuring conservation of both water mass and moist static energy. As explained earlier, conserving water mass is a straightforward process that involves directly applying our constraining methodology. However, approxi-

mating the (column-integrated) moist static energy S constains the following procedure:

$$S \approx ((1 - Q_v) \cdot c_{pd} + Q_L \cdot c_l) \cdot T + L_v \cdot Q_v, \tag{4.21}$$

where

$$L_v \approx 2.5008 \cdot 10^6 + (c_{pw} - c_L) \cdot (T - 273.16)$$

is the latent heat of vaporisation in (Jkg^{-1}). The water vapor $Q_v[kg \cdot kg^{-1}]$, the liquid water $Q_L[kg \cdot kg^{-1}]$, and the temperature $T[K]$ are being predicted, whereas c_{pd}, c_{pv} and $c_L[J \cdot K^{-1} \cdot kg^{-1}]$ are heat capacity constants.

The following procedure is used to predict these quantities while conserving moist static energy:

1. Given LR $T^{LR}, Q_V^{LR}, Q_L^{LR}$
2. Calculate LR S^{LR} with (4.21)
3. Predict SR $S^{SR}, Q_v^{SR}, Q_L^{SR}$ while enforcing (Eq. (4.3)) using one of our constraint layers
4. Calculate SR T^{SR} using (4.21) and SR $S^{SR}, Q_v^{SR}, Q_L^{SR}$.

This implies we predict T^{SR} not directly, but by predicting S^{SR}. With that we are able to predict the temperature T while ensuring (approximate) energy conservation by applying our constraint layer to the prediction of S^{SR}.

Different Simulations When LR-HR pairs stem from two simulations conducted at varying resolutions, rather than by deriving the local mean of the HR, the downscaling constraint is not inherently met within the data. This scenario is applicable to our WRF and NorESM datasets. Despite the downscaling constraint not being precisely adhered to (refer to Figure 4.5), it is approximately fulfilled, allowing us to employ our constraint layers in the same way as in previous approaches. Even when the real low-resolution data and the downsampled high-resolution data exhibit minor discrepancies, constraining can still enhance predictive quality, as demonstrated in Figure 4.8.

4.3 Experiments

Here, we conduct two types of experiments:

1. Show the applicability of our constraining method to different neural network architectures.

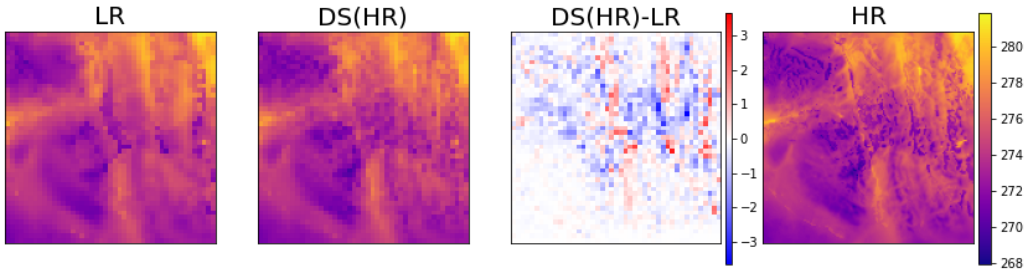

Figure 4.5: A random LR-HR pair from the WRF temperature data. HR and LR come from different runs using the same model at different resolutions. Here we compare the real LR with the LR data created by average pooling of the HR, written as DS(HR). It shows that there is not an exact match between LR and downsampled HR, which makes the success of a constraint layer more difficult. The violation of the downscaling constraint in the WRF dataset is 0.68K on average.

2. Show the applicability of our constraining method to different datasets and different constraint types included in that dataset.

In most of our experiments, we utilize synthetic low-resolution data generated by averaging the original high-resolution samples. This approach mirrors typical practices in evaluating ideal downscaling configurations. Moreover, we investigate scenarios involving pairs of real low-resolution and high-resolution simulations to demonstrate the efficacy of our methods in real-world applications.[2]

Architectures

Now, we thoroughly test our constraint methods throughout a variety of standard deep learning SR architectures as described in detail in Section 2.3.4, including an SR-CNN, conditional GAN, a combination of an RNN and CNN for spatio-temporal SR, and a new architecture combining optical flow with CNNs/RNNs to increase the resolution of the temporal dimension. The standard, unconstrained versions of these architectures then also serve as a comparison for our constraining methodologies. We here give details for the specific setups in which the formerly introduced architectures are being used.

[2]The code can be found at `https://github.com/RolnickLab/constrained-downscaling`. It contains 1200 lines of code written in Python/Pytorch.

SR-CNNs The SR-CNN network that is used here is similar to Lim et al. [2017]. It consists of convolutional layers and ReLU activations. The upsampling is performed by a transpose convolution followed by residual blocks (convolution, ReLU, convolution, adding the input, ReLU). The architecture for 2-times downscaling is shown in Figure 4.2. The architecture starts with a convolutional layer, using 3×3 kernels and stride and padding of one, followed by a ReLU. Then depending on the upsampling factor multiple transpose convolutional layers, with a stride and kernel size of two. Multiple residual blocks then learn the higher-frequency features. The final layers include a 1×1 kernel convolutions. For more details on CNNs, in general see Section 2.2.2 in the context of downscaling see Section 2.3.4. For an explanation of residual layers, convolutions, and transpose convolution see Section 2.2.1.

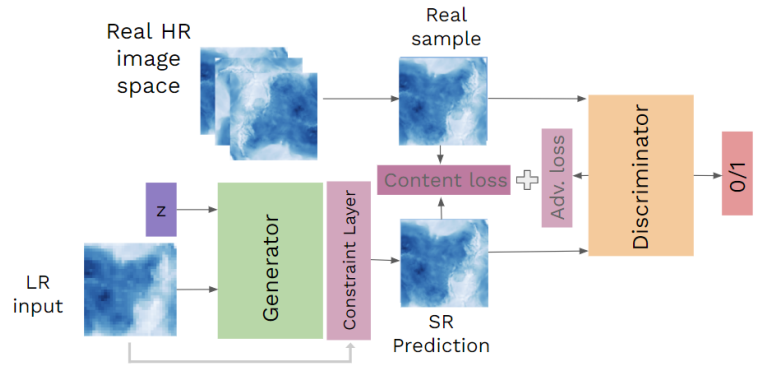

Figure 4.6: The GAN architecture used here including the constraint layer (in red). The LR input is passed to the last generator layer, the constraint layer, to enforce the constraint and produce a consistent HR output.

SR-GAN A common approach for super-resolution tasks is to employ a conditional Generative Adversarial Network (GAN) architecture [Mirza and Osindero, 2014b, Ledig et al., 2016b]. In our implementation, we utilize the CNN architecture introduced earlier as the generator network. The discriminator, adapted from [Ledig et al., 2016b], comprises convolutional layers with a stride of 2, gradually reducing dimensionality at each step, combined with ReLU activations and a final average pooling layer and a sigmoid activation to obtain the predicted scalar between (0,1). Trained as a classifier, it distinguishes between super-resolved (SR) images and real high-resolution (HR)

images using binary cross-entropy loss. The generator takes both Gaussian noise and low-resolution (LR) data as inputs and produces an SR output. Its training involves a combination of MSE loss to aid in reconstruction and adversarial loss provided by the discriminator, following the standard approach of SR GANs [Ledig et al., 2017]. For details and background on GANs and cGANS, in general, see Section 2.2.2, in the context of downscaling see Section 2.3.4. For a description of the cross-entropy loss see Section 2.2.3.

SR-ConvGRU We employ an SR architecture based on the GAN presented by Leinonen et al. [2021], which uses ConvGRU layers to address the spatio-temporal nature of super-resolving a time series of climate data, combining an RNN structure with convolutions. Compared to Leinonen et al. [2021] we here use the generator on its own, both during inference and training time without the discriminator, providing a deterministic approach. The ConvGRU network employed here starts with a 3×3-kernel convolutional layer, followed by a residual block using again 3×3 kernel convolutions and leaky ReLU. After the residual block, the ConvGRU layer is applied, followed by bilinear upsampling and a final convolutional layer. An explanation of RNNs and ConvGRUs can be found in Chapter 2, in Sections 2.2.2 and 2.3.4.

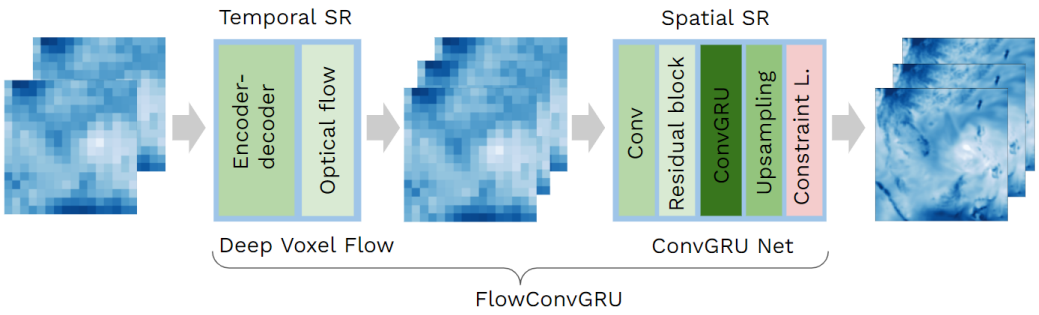

Figure 4.7: Our novel spatio-temporal architecture, combining deep voxel flow and a ConvGRU-based SR NN. The inputs are two LR images at two times, the first part predicts the in-between time step using the Deep Voxel Flow model, the second part increases the spatial resolution of the three time steps using a convolutional GRU net.

SR-FlowConvGRU To enhance the temporal resolution of our data, we utilize the deep flow method [Liu et al., 2017], a deep learning framework

designed for video frame interpolation. This approach merges optical flow techniques with neural networks to interpolate between video frames effectively. We introduce a novel architecture named FlowConvGRU by integrating the deep flow model with the ConvGRU network (see Figure 4.7). Initially, we enhance the temporal resolution with the deep voxel flow network, resulting in a time series of low-resolution (LR) images with a higher frequency. Subsequently, we employ the ConvGRU architecture to boost the spatial resolution of these LR images. The combined neural networks are then trained end-to-end, as depicted in Figure 4.7. To go into more detail on the deep voxel flow architecture: the deep voxel flow network consists of two parts, a first encode-decoder model that is then combined with optical flow to predict the intermediate frame. The encoder-decoder networks consist of convolutional blocks with ReLU, max pooling, and batch normalisation layers. In the encoder, the convolutional layers with kernels of dimension 5 and a stride of two decrease the dimensions, whereas in the decoder bilinear interpolation is used to increase the dimensions again. Then optical flow is combined with the encoder-decoder output to obtain the target frame. For a more detailed explanation of optical flow see Section 2.2.1.

Training

Our models are trained with the Adam optimizer, a learning rate of 0.001, and a batch size of 256. We trained for 200 epochs, which took about 3—6 hours on a single NVIDIA A100 Tensor Core GPU, depending on the architecture. All models use the MSE as their criterion and weight decay as regularisation; the GAN additionally uses its discriminator loss term. All data are normalized between 0 and 1 for training. A description of the Adam optimizer and the definition and role of learning rate and batch size can be found in Section 2.2.3. We do not perform much hyperparameter optimisation here but rely on previously established values.

Baselines

Pixel Enlargement The enlargement baseline consists of scaling the LR input to the same size as the HR by duplicating the pixels. It is included to have reference metrics that reflect how close the LR is to the HR data, without adding any additional assumptions. This baseline conserves mass by construction.

Bicubic Interpolation Serving as a simple baseline, we use bicubic interpolation for upsampling – a common choice [Kurinchi-Vendhan et al., 2021]

– for spatial SR and take the mean of two frames for temporal SR. We considered bilinear and nearest neighbour interpolation as well, but bicubic interpolation showed the strongest performance.

Unconstrained Neural Networks Additionally, we compare against an unconstrained version of the above-introduced standard SR NN architectures – SR-CNN, SR-GAN, SR-ConvGRU, and SR-FlowConvGRU).

4.4 Results

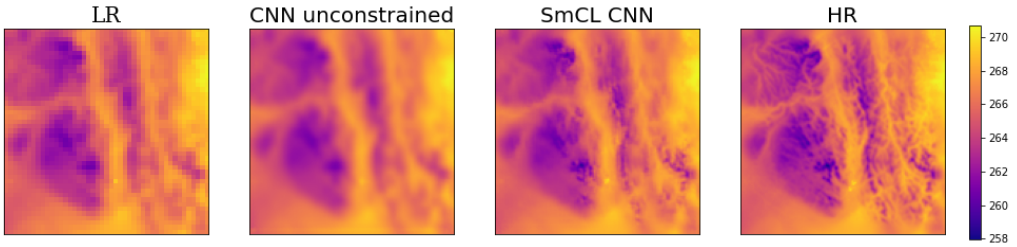

Figure 4.8: A random prediction for the WRF temperature test dataset. We compare unconstrained and softmax-constrained downscaling predictions. It can be seen that in this case, the constraining improves the visual quality significantly including more fine-grain details.

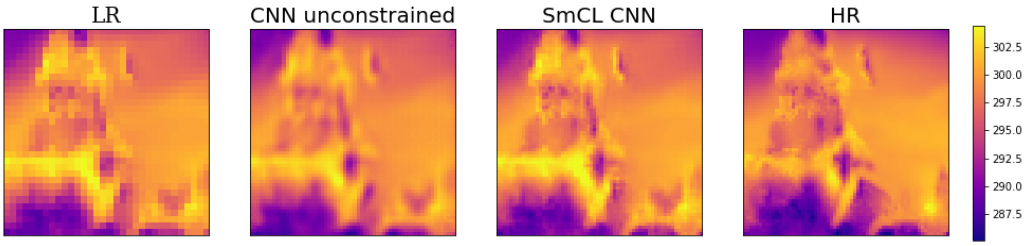

Figure 4.9: A random sample prediction for the NorESM temperature test dataset, we compare an unconstrained CNN and a softmax-constrained CNN here on the downscaling task. The constrained prediction looks more similar to the HR ground truth, including more high-frequency features.

Metrics

To evaluate our results, we use typical metrics for weather and climate super-resolution as introduced in section 2.3.5: root-mean-square error (RMSE,

see Eq. (2.28)), mean absolute error (MAE, see Eq. (2.29)), Fractional Skill Score (FSS, see Eq. (2.32)), and mean bias (see Eq. (2.30)) as well as typical metrics for super-resolution: peak signal-to-noise ratio (PSNR, see Eq. (2.26)), structural similarity index measure (SSIM, see Eq. (2.24)), multi-scale SSIM (MS-SSIM, see Eq. (2.25)), and Pearson correlation (see Eq. (2.27)).

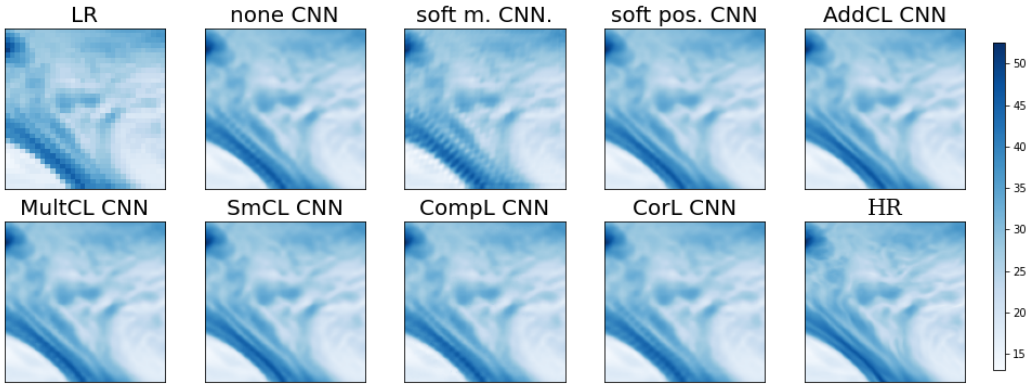

Figure 4.10: One example image from the downscaling water content test set. Shown here are the LR input, different constrained and unconstrained predictions, and the HR image as a reference. For the unconstrained CNN prediction, we can observe some artifacts in the lower left part, which get amplified by applying soft-constraining but decreased using hard-constraining like AddCL or SmCl.

We find that most metrics are highly correlated in our case, therefore some are moved to the appendix (see Table A.2). For the GAN giving a probabilistic prediction, we also use the continuous ranked probability score (CRPS, see Eq. (2.33)). Given that we are interested in the violation of conservation laws and predicting non-physical values, we also look at the average constraint violation, the number of (unwanted) negative pixels, and the average magnitude of negative values. We additionally look at the variance among the pixels within a predicted super-pixel and investigate the difference for constraining methods, see appendix, Table A.5.

Results of Different Constraining Methods Hard-constraining shows exact conservation and appears to enhance performance in most cases when using readjustment layers (see e.g. Table 4.2). For completion or correction methods that also ensure exact satisfaction of given constraints performance from the unconstrained version is either kept constant or decreased.

Table 4.2: Metrics RMSE, MAE, Multi-scale SSIM, mass violation and number of negative pixels per million pixels, (all explained in Section 4.4) for different constraining methods applied to the SR-CNN, calculated over 10,000 test samples of the water content downscaling data. The mean is taken over three runs. The best scores are highlighted in bold blue, second best in bold.

Dat	Model/ Constraint	RMSE ↓	MAE ↓	MS-SSIM ↑	Mass viol. ↓	#Neg per mil. ↓
W4	Enlarge	1.29	0.72	97.6	**0.00**	**0**
W4	Bicubic	0.80	0.40	99.1	0.17	0.5
W4	cnn/none	0.66	0.33	99.4	0.06	2.4
W4	soft m.	0.80	0.41	99.2	0.02	581
W4	soft n.	0.61	0.30	99.5	0.04	**0**
W4	soft n.+m.	0.75	0.37	99.3	**0.00**	0.3
W4	AddCL	**0.58**	**0.29**	**99.5**	**0.00**	1.4
W4	MultCL	0.61	0.30	99.5	**0.00**	**0**
W4	SmCL	**0.58**	**0.29**	**99.5**	**0.00**	**0**
W4	CompL	0.63	0.31	99.4	**0.00**	10
W4	CompL offl	0.66	0.33	99.4	**0.00**	2.7
W4	CorL	0.66	0.32	99.4	0.02	**0**
W4	CorL offl	0.66	0.33	99.4	0.06	**0**

The application of soft-constraining on the other hand does decrease constraint violation, but still maintains a significant magnitude of it, which can be seen in Table 4.2 for example. In addition, soft constraints seem to suffer from an accuracy-constraints trade-off, where depending on the regularisation factor, either the constraint violation is reduced or the accuracy increases, but it struggles to do both simultaneously. A table for different regularisation factors for soft mass constraining is shown in the appendix (see Table A.1). For some cases soft constraints cause instabilities in training and diverge, which happens e.g. for the GAN architecture, see Table 4.3.

Overall, we observe the best performance by AddCL and SmCL methods throughout NN architectures and datasets. SmCL provides the advantage of also ensuring positivity in relevant cases (see e.g. Table 4.2).

MultCL, performs weaker than the other readjustment layers in terms of predictive skills (see Table 4.2), which could be due to instability when inputs come close to zero. The completion method works well for the CNN water content case (see Table 4.2), but struggles for more complex tasks and

Table 4.3: Metrics RMSE, MAE, Multi-scale SSIM, mass violation and number of negative pixels per million pixels, (all explained in Section 4.4) for different constraining methods applied to an SR-GAN, calculated over 10,000 test samples of the 4x upsampling water content data. The mean is taken over three runs. The best scores are highlighted in bold blue.

Data	Model Constraint	RMSE ↓	MAE ↓	CRPS ↓	Mass viol. ↓	#Neg per mil.↓
W4	Enlarge	1.29	0.72	-	**0.00**	**0**
W4	Bicubic	0.80	0.40	-	0.17	0.5
W4	GAN/none	0.63	0.31	**0.15**	0.05	3.5
W4	soft m.	2.81	1.17	0.54	0.90	2613
W4	soft n.	0.71	0.36	0.17	0.08	2.2
W4	soft n.+m.	2.43	1.72	0.81	1.07	**0**
W4	AddCL	**0.60**	**0.31**	**0.15**	**0.00**	7.4
W4	MultCL	0.73	0.41	0.20	**0.00**	**0**
W4	SmCL	**0.60**	**0.31**	**0.15**	**0.00**	**0**
W4	CompL	1.16	0.72	0.33	**0.00**	4225
W4	CompL offl.	0.63	0.32	**0.15**	**0.00**	142
W4	CorL	0.69	0.36	0.19	0.07	**0**
W4	CorL offl.	0.63	0.31	**0.15**	0.04	**0**

architectures, such as time-series models, see Table 4.4. The offline correction method that enforces positivity of the output, keeps the performance for al models, the online version can worsen metrics such as RMSE though (see e.g. 4.5).

Results of Different Architectures We show in Tables 4.2, 4.3, 4.4, 4.5 that adding the constraint layers enforces the constraint and improves the evaluation metrics compared to the CNN case for all architectures (CNN, GAN, ConvGRU, FlowConvGRU). To constrain the GAN leads to less of a performance boost, but AddCL and SmCL still enhance the prediction's quality compared to the unconstrained GAN. Including the temporal dimensions, the constraining improves the predictions much more significantly than in the case with just a single time step (see Tables 4.2 and 4.5). As shown in Table 4.4, the RMSE, for example, decreases from 0.67 in the unconstrained case to 0.50 with constraints.

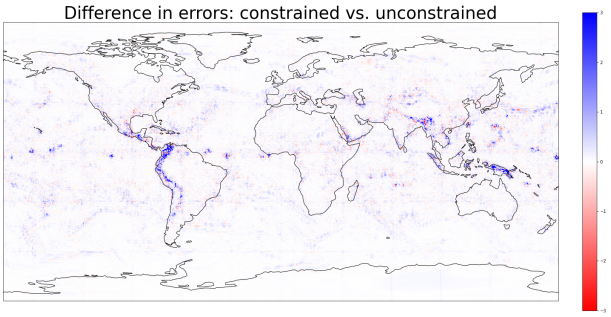

Figure 4.11: The difference in the errors of constrained and unconstrained predictions. Positive values (blue) mean a lower error in the constrained version.

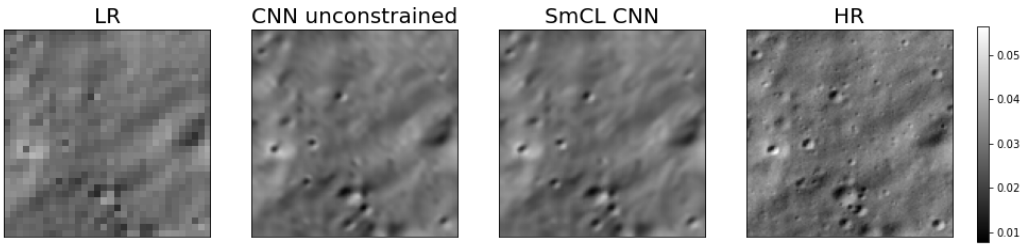

Figure 4.12: A random sample prediction from the lunar downscaling dataset is shown. We compare the unconstrained with the constrained prediction.

Results of Different Datasets and Constraints

Different Upsampling Factors Results The success of our constraining methodology does not depend on the upsampling factor: in Table A.6, we can see that the constraining methods work well and improve all metrics for upsampling factors of 2, 4, 8, and 16. Looking at a visual example in Figure 4.16 we can see that structures that are not correctly reconstructed in the unconstrained prediction, are apparent in the constrained version. For 16-times upsampling we observe very blurry predictions, for both constrained and unconstrained versions. Here adding additional inputs, such as high-resolution topography would be necessary to achieve good downscaling results.

OOD Results When applied to our out-of-distribution dataset, the improvement achieved by adding constraints is very similar to the randomly split data (see results in the appendix in Table A.2).

Table 4.4: Metrics RMSE, MAE, Multi-scale SSIM, mass violation and number of negative pixels per million pixels, (all explained in Section 4.4) for different constraining methods applied to an SR-ConvGRU, calculated over 10,000 test samples of the water content downscaling data. The best scores are highlighted in bold blue.

Data	Model Constraint	RMSE ↓	MAE ↓	MS-SSIM ↑	Mass viol. ↓	#Neg per mil.↓
WT1	Enlarge	1.29	0.72	97.7	**0.00**	**0**
WT1	Bicubic	0.81	0.40	99.2	0.17	2.2
WT1	none	0.67	0.34	99.4	0.10	55
WT1	soft m.	0.74	0.39	99.3	0.18	50
WT1	soft n.	0.78	0.42	99.3	0.20	9.7
WT1	soft n.+m.	0.98	0.56	98.9	0.29	9.8
WT1	AddCL	**0.50**	**0.26**	**99.6**	**0.00**	1358
WT1	MultCL	0.90	0.47	99.0	**0.00**	0.3
WT1	SmCL	**0.50**	**0.26**	**99.6**	**0.00**	**0**
WT1	CompL	1.22	0.54	98.29	**0.00**	4405
WT1	CompL offl	0.93	0.42	99.0	**0.00**	1008
WT1	CorL	0.79	0.44	99.3	0.24	**0**
WT1	CorL offl	0.67	0.34	99.4	0.10	**0**

Moist Static Energy Results Mass is not the only quantity that can be conserved, but other quantities such as moist static energy. We show that moving on to different quantities of the ERA5 dataset, temperature, water vapor, and liquid water. By looking at Table A.7 (see appendix), one can observe similar results for liquid water Q_L and water vapor Q_v as for the total water content: AddCL and SmCL significantly improve results in all measures over the unconstrained CNN, while enforcing energy and mass conservation. For temperature, on the other hand, MultCL performs the strongest, followed by SmCL, whereas AddCL achieves smaller improvements in the scores. This shows that MultCL performs very well, when there are now variables encountered that are close to zero.

WRF Results The WRF temperature dataset differs from others in that it contains low-resolution data points derived from a distinct simulation rather than being downsampled. This characteristic presents a more challenging

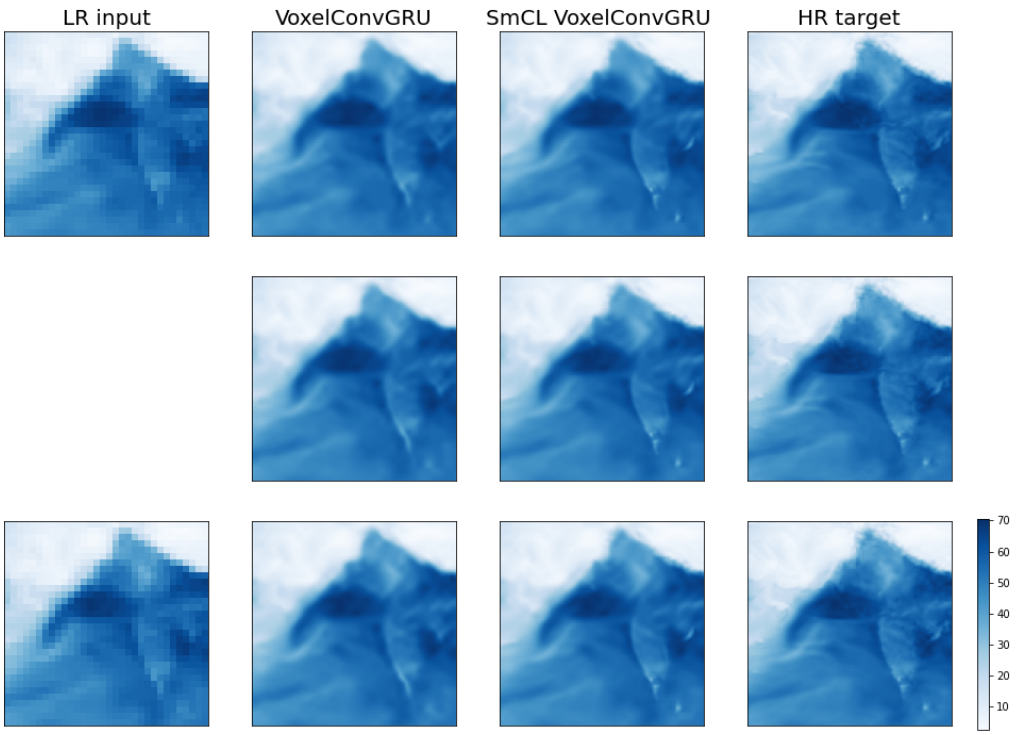

Figure 4.13: One random test sample and its prediction. Shown here are the two LR input time steps, predictions by both a constrained and unconstrained version of the FlowConvGRU, and the HR sequence as a reference.

task for our model. However, Table 4.6 indicates a slight improvement in scores when employing our constraint layer. This outcome might seem counterintuitive given the presence of violations in the training data. Nevertheless, these violations are relatively minor and resemble random noise, thus not introducing significant bias. Consequently, the constraints contribute to simplifying the learning process and enhancing performance. While the slight violations in the original dataset could prompt consideration of soft-constraining, our observations reveal that soft-constraining actually undermines predictive performance, whereas hard-constraining surprisingly proves beneficial. On average, the constraint violation in the original data amounts to an RMSE of 0.68.

NorESM results For the NorESM data the results are shown in Table 4.8: the best scores are in achieved by the unconstrained CNN. This is probably

Table 4.5: Metrics RMSE, MAE, Multi-scale SSIM, mass violation and number of negative pixels per million pixels, (all explained in Section 4.4) for different constraining methods applied to our FlowConvGRU, calculated over 10,000 test samples of the water content downscaling dataset. The best scores are highlighted in bold blue, second best in bold.

Data	Model Constraint	RMSE ↓	MAE ↓	MS-SSIM ↑	Mass viol. per mil.↓	#Neg
WT2	Interp.	0.83	0.43	99.1	0.17	2.1
WT2	none	0.67	0.35	99.4	0.07	18
WT2	soft m	0.70	0.38	99.4	0.08	21
WT2	soft n	0.75	0.40	99.3	0.10	6.8
WT2	soft n+m	1.05	0.60	98.6	0.20	12
WT2	AddCL	**0.51**	**0.28**	**99.6**	**0.00**	37
WT2	MultCL	0.72	0.38	99.3	**0.00**	**0**
WT2	SmCL	**0.51**	**0.28**	**99.6**	**0.00**	**0**
WT2	CompL	1.44	0.66	97.7	**0.00**	6042
WT2	CompL offl.	1.76	0.54	97.1	**0.00**	4394
WT2	CorL	0.72	0.39	99.3	0.11	**0**
WT2	CorL offl.	0.67	0.35	99.4	0.07	**0**

due to the stronger violation of the downscaling consistency constraints between low-resolution and high-resolution data. We can observe a significant difference between the real LR and the HR downsampled, as shown in Figure 4.4. The violation of the constraints here is 2.48K (RMSE) per superpixel on average, which is much higher than for the WRF case (0.68). The visual quality of the prediction, on the other hand, seems to be improved by constraining, an example is shown in Figure 4.9. One potential approach for improvements here could be lat-lon weighted constraining as the data covers a much larger area and thus the areas per grid box might vary more.

Lunar Satellite Imagery Results Finally, we also show that applying our constraint methodology can improve results in other domains, even in cases where there is no physics involved. We see that for lunar satellite imagery, the application of our SmCL improves the traditional metrics, as shown in Table 4.7. The visual quality is slightly improved, see Figure 4.12.

Table 4.6: Metrics RMSE, MAE, Multi-scale SSIM and constraint violation (all explained in Section 4.4) for different constraining methods applied to the SR-CNN applied on the WRF temperature data, calculated over 10,000 test samples. We choose the most common (RMSE, MAE, Multi-scale SSIM) and relevant (constr. viol) for our cases. The mean is taken over three runs. The best scores are highlighted in bold blue, second best in bold.

Data	Constraint	RMSE ↓	MAE ↓	MS-SSIM ↑	Constr. viol. ↓
WRF	Enlarge	1.02	0.65	94.5	**0.00**
WRF	Bicubic	0.97	0.61	95.0	0.07
WRF	none	**0.95**	0.62	94.9	0.18
WRF	soft m.	1.02	0.66	94.6	0.03
WRF	AddCL	0.96	**0.60**	**95.2**	**0.00**
WRF	MultCL	0.97	0.61	95.0	**0.00**
WRF	SmCL	**0.95**	**0.59**	**95.3**	**0.00**
WRF	CompL	0.99	0.62	94.8	**0.00**
WRF	CompL offl.	1.27	0.73	92.9	**0.00**

Table 4.7: Metrics RMSE, MAE, Multi-scale SSIM, and PSNR, (all explained in Section 4.4) for different constraining methods applied to the SR-CNN, calculated over the test samples of the lunar dataset. The mean is taken over three runs. The best scores are highlighted in bold blue.

Data	Model	Constraint	RMSE ↓	MAE ↓	SSIM ↑	PSNR ↑
Lunar	CNN	none	0.0022	0.0015	90.08	37.6
Lunar	CNN	SmCL	**0.0021**	**0.0014**	**90.40**	**37.7**

Perceptual Quality of Predictions In addition to a quantitative enhancement, we can see an improved visual quality for some examples, as shown in Figure 4.10 and 4.16 for the water content data. In Figure 4.10 it is visible that an artifact (diagonal stripes) in the unconstrained prediction gets removed by constraint layers AddCL and SmCL. Figure 4.16 shows that features, such as the diagonal line in the HR image can be better reconstructed with using constraints, especially looking at 8x upsampling. For the WRF

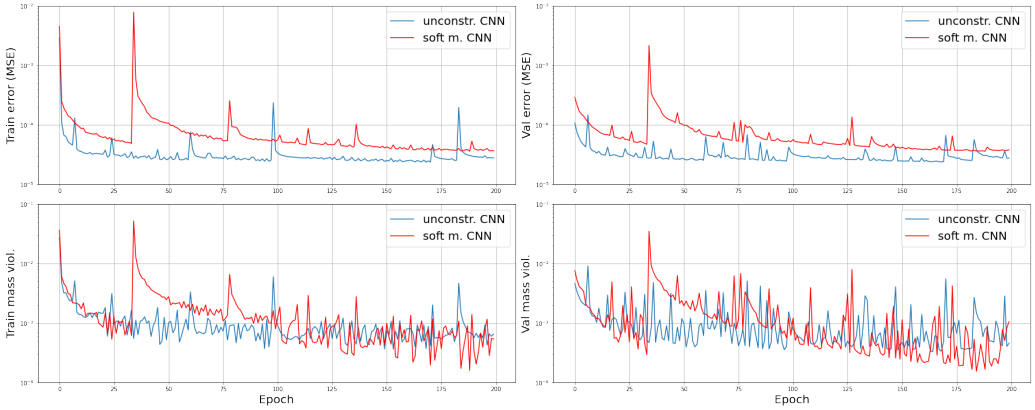

Figure 4.14: The development of training and validation errors with increasing iterations during training as well as mass conservation violation. Shown for an unconstrained CNN and the soft-constrained version applied to the water content data.

Table 4.8: Metrics RMSE, MAE, Multi-scale SSIM and constraint violation (all explained in Section 4.4) for different constraining methods applied to the SR-CNN, calculated over the test samples of the NorESM downscaling dataset. The mean is taken over three runs. The best scores are highlighted in bold blue, second best in bold.

Data	Model Constraint	RMSE ↓	MAE ↓	MS-SSIM ↑	Constr. viol. ↓
NorESM	Enlarge	2.99	1.92	96.0	**0.00**
NorESM	Bicubic	2.91	1.86	96.4	0.07
NorESM	none	**2.35**	**1.56**	**96.9**	1.03
NorESM	soft	2.93	1.87	96.3	0.04
NorESM	AddCL	**2.89**	**1.85**	**96.5**	**0.00**
NorESM	MultCL	**2.89**	1.86	96.4	**0.00**
NorESM	SmCL	**2.89**	**1.85**	**96.5**	**0.00**
NorESM	CompL	2.90	1.86	96.4	**0.00**
NorESM	CompL offl.	4.12	2.27	91.9	**0.00**

temperature forecast data, we see a very significant improvement in the perceptual quality of the prediction. Looking at an example, such as shown in

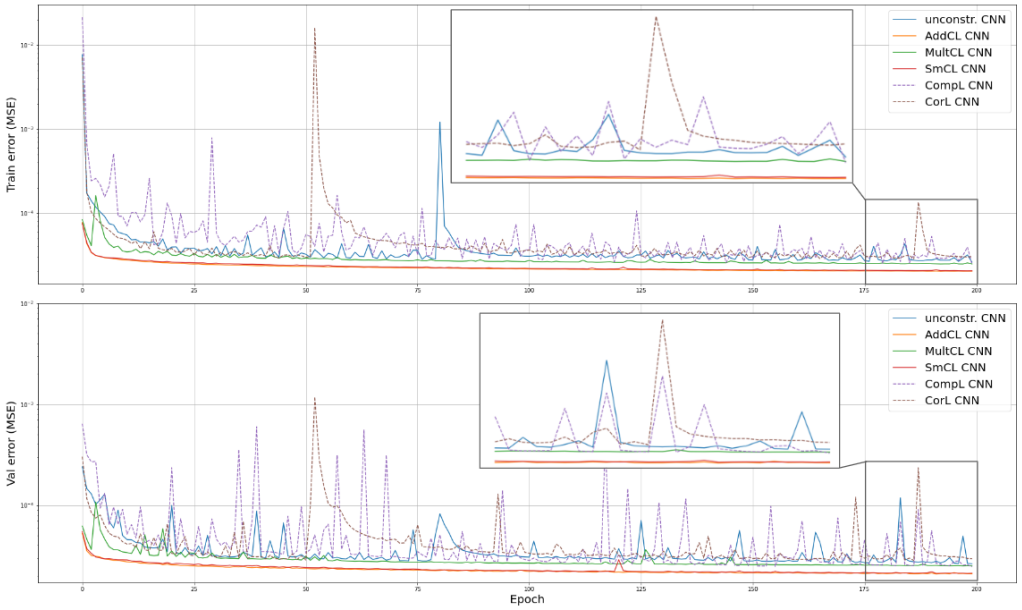

Figure 4.15: The development of training and validation errors with increasing iterations during training. Shown for an unconstrained CNN and different hard-constrained CNNs applied to the water content data. We can observe how the readjustment layers (AddCL, MultCL, SmCL) are accelerating convergence and smoothening the learning curve, both measured in training and validation error.

Figure 4.8, we can see how much more high-frequency detail is added to the prediction when including a constraint layer. For the lunar satellite imagery, Figure 4.12 shows that applying constraints can make the image slightly less blurry.

Spatial Distribution of Errors A common challenge encountered in downscaling methods is the coastal effect, characterized by amplified prediction errors in coastal regions. Additionally, mountain ridges can also present significant challenges. As illustrated in Figure A.1 (see appendix), both the unconstrained and softmax-constrained predictions exhibit heightened errors in coastal and mountainous areas. However, upon closer examination of the difference in errors between the unconstrained and constrained versions depicted in Figure 4.11, it becomes evident that applying constraints results in reduced errors in these regions.

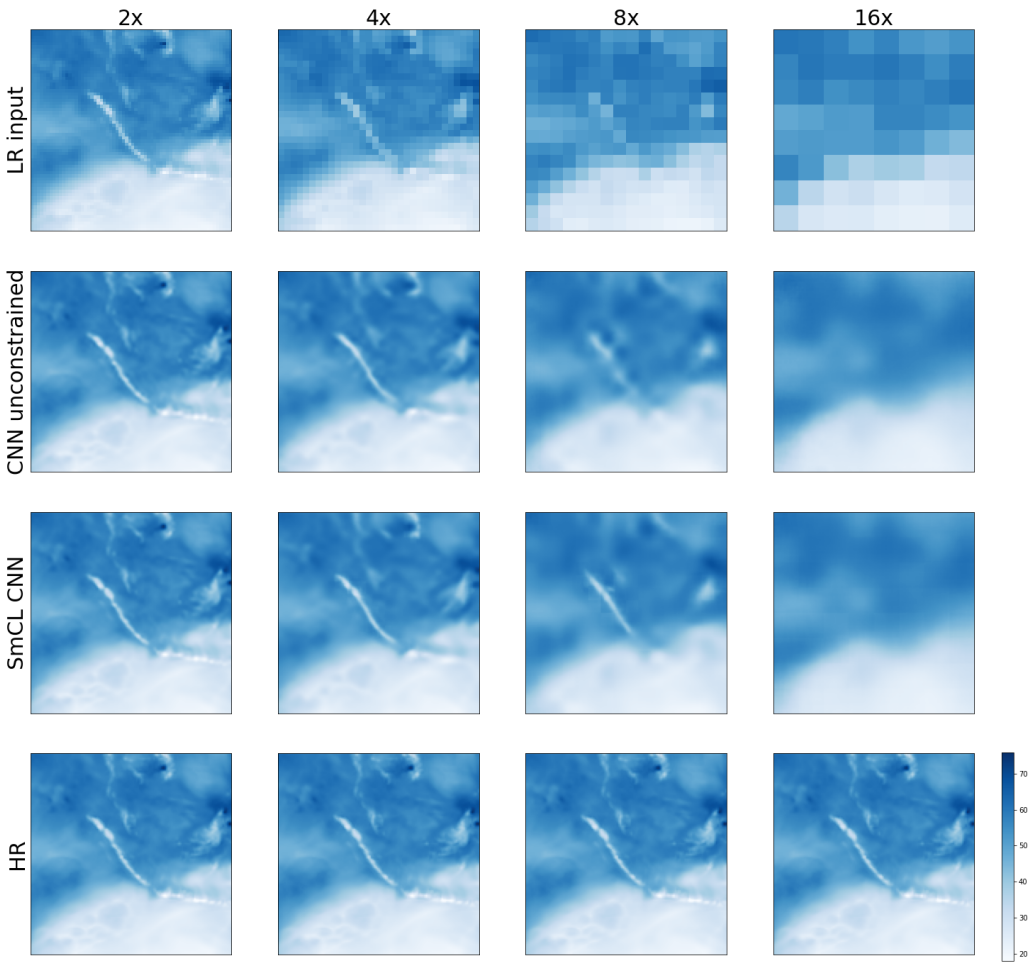

Figure 4.16: Each model was trained for the same target resolution but with a different upsampling factor. The first row shows the LR inputs for each resolution and the last row the corresponding HR ground truth. The second and third rows show the prediction of an unconstrained CNN and with the SmCL, respectively.

Loss Curves Observing how the training and validation MSEs develop during training (see Figure 4.15), we can see that the curves differ significantly among different constraints. Training and validation curves both show the same behaviours. The three readjustment layers AddCL, MultCL, and SmCL all exhibit much smoother curves than the unconstrained CNN, while also converging to a lower minimum. AddCL and SmCL here again show the best performance, being smooth and lower from the beginning of the training. CompL and CorL both show a more unstable behaviour than without

constraints.

For soft constraints in Figure 4.14 we can see that the additional loss term worsens the training and validation curves, converging to a higher local minimum. The constraint violation is improved by soft constraining, starting around epochs 100, but still showing a very unstable behaviour.

Chapter 5

Physics-Constrained Emulation

This chapter goes through the second application case, aerosol microphysics emulation, where we replace a computationally expensive numerical model with an ML approximation while still obeying physical constraints. After we described aerosol modeling and specifically the aerosol microphysics model M7 in Section 2.5, we here follow a similar structure to that in the downscaling chapter by introducing the formal setup, reformulating the constraint methods, then moving to the description of the dataset and experimental setup, and concluding with results. In contrast to the downscaling work, in this chapter, there are two separate sets of experiments and results:[1]

1. Offline experiments and results, i.e., training, validating, and testing on a fixed dataset in Python.

2. Online experiments, i.e. including our trained model in ICON with Fortran and analyzing the performance in global year-long ICON runs.

5.1 Constrained Emulation

To fulfill our goal of replacing M7 with a faster ML version, we want to ensure that the emulator respects the same physical constraints as the original. M7 redistributes aerosol masses and numbers among different modes, modeling their change in time for each grid box independently, and it does not consider sinks or sources. M7 conserves the aerosol mass per species within one grid box. The positivity of the masses, the water content, and the numbers needs to always be ensured. We refer to our emulator as *NeuralM7*, a neural

[1]Parts of this chapter have been published in P. Harder, D. Watson-Parris, P. Stier, D. Strassel, N. R. Gauger, and J. Keuper. Physics-informed learning of aerosol microphysics. *Environmental Data Science*, 1:e20, 2022a. doi: 10.1017/eds.2022.22

network version of the M7 microphysics module. For more details on M7 we refer to Section 2.5.

5.1.1 Aerosol Microphysics Emulation Setup

Let us formulate the learning task for this emulation problem, where we try to approximate the M7 model. An input vector $x \in \mathbb{R}^{n_{\text{in}}}$ consists of all the inputs that are also provided for the original model; this includes atmospheric variables, as well as masses and the number concentration of aerosols, in our case $n_{\text{in}} = 35$. The outputs $y \in \mathbb{R}^{n_{\text{out}}}$ are changes in aerosol masses and numbers (i.e. $n_a = 24$ aerosol variables), as well as the water content ($n_w = 4$), with overall $n_{\text{out}} = 28$ output variables. The first $n_m = 17$ variables for both x and y describe the masses with the species $S = \{$SO4, BC, OC, DU$\}$. The following $n_n = 7$ variables refer to aerosol numbers per mode. A full list of input and output quantities is shown in the appendix (see Appendix B.2).

5.1.2 Constraint Formulation

Let us recall again the constraint formulation of the previous chapter (see Section 3.1) – first the equality constraints

$$\sum_{i \in I_j^{(\text{eq})}} g_i^{(\text{eq})}(y_i) + h_j^{(\text{eq})}(x) = 0 \tag{5.1}$$

for each $j = 1, \ldots, n_p$, and then the inequality constraints

$$\sum_{i \in I_j^{(\text{in})}} g_i^{(\text{in})}(y_i) + h_j^{(\text{in})}(x) \geq 0 \tag{5.2}$$

for each $j = 1, \ldots, m_p$.

For aerosol mass conservation while predicting changes, we have to ensure that changes per species add up to zero - the equality constraints are

$$\sum_{i \in I_s^{(\text{eq})}} y_i = 0, \tag{5.3}$$

with s iterating over the different species S. Here, $h_s^{(\text{eq})} \equiv 0$ and $g_i^{(\text{eq})} \equiv \text{Id}$. $I_s^{(\text{eq})}$) consists of the indices of species' s mass, $(I_s^{(\text{eq})})_{s \in S}$ is a partition of $\{1, \ldots, n_m\}$. $I_s^{(\text{eq})}$ contains n_s variables, $|I_s^{(\text{eq})}| = n_s$.

The positivity constraint includes the input variables $x_i, i = 1, \ldots, n_a$; not the change must be positive but the sum of input plus predicted change

$$y_i + x_i \geq 0 \tag{5.4}$$

for $i = 1, \ldots, n_a$, with $I_j^{(\text{in})} = \{j\}$ for $j = 1, \ldots, n_{\text{out}}$. For water content, which is predicted directly, the output itself has to be positive

$$y_i \geq 0, \tag{5.5}$$

for $i = n_a + 1, \ldots, n_{\text{out}}$. This means $h_i^{(\text{in})}(x) = x_i$ for an aerosol index $i = 1, \ldots, n_a$, $h_i^{(\text{in})} \equiv 0$ for $i = n_a + 1, \ldots, n_{\text{out}}$, and $g_i^{(\text{in})} \equiv \text{Id}$ for $i = 1, \ldots, n_{\text{out}}$.

Soft Constraining The mass violation per aerosol species can be penalized via soft constraints (soft mass/soft m.)

$$\mathcal{L}^{(\text{eq})}(y, \hat{y}) = \mathcal{L}(y, \hat{y}) + \frac{1}{n_s} \sum_{s \in S} a_s (\sum_{i \in I_s} y_i)^2, \tag{5.6}$$

with tunable parameters $a_s \in \mathbb{R}$, to weigh depending on the species. We choose these parameters in a way that at beginning of training the individual terms $\sum_{i \in I_s} y_i$ are all of the same magnitude. For the loss function, $\mathcal{L}$, we again choose the MSE. To encourage positivity, we can penalize $-(y + x)$ for predicting changes in aerosol masses and numbers and penalize $-y$ when predicting water content directly (soft neg./soft n.)

$$\mathcal{L}^{(\text{in})}(y, \hat{y}) = \mathcal{L}(y, \hat{y}) + \frac{1}{n_a} \sum_{i=1}^{n_a} b_i \text{ReLU}(-(y_i + x_i))^2 + \frac{1}{n_w} \sum_{i=n_a+1}^{n_{\text{out}}} w_i \text{ReLU}(-y_i)^2, \tag{5.7}$$

again with tunable parameters $b_i, w_i \in \mathbb{R}$, which are chosen so the individual penalizing terms are roughly the same.

Readjustment Layers The previously introduced additive readjustment layer, AddCL, can be employed to achieve guaranteed aerosol mass conservation, readjusting the intermediate outputs $\tilde{y}_i$

$$y_i^{\text{AddCL}} = \tilde{y}_i - \frac{1}{n_s} \sum_{k \in I_s} \tilde{y}_k, \tag{5.8}$$

for $i \in I_s$ and $s \in S$. The layers MultCL and SmCL cannot be applied directly given that $h_j(x) = 0$, leading to a constant zero prediction. Our variables will be normalized and standardized for our final application in this aerosol emulation task. Therefore, SmCL and MultCL can be applied again. We will review that case below (see Section 5.1.2).

Completion Layer The readjustment layers treat every variable per species equally, whereas for the completion method, only one variable per species s is changed and set as the negative sum of the remaining variables of the same species

$$y_{i_c}^{\text{CompL}} = -\sum_{k \in I_s \setminus \{i_c\}} \tilde{y}_k. \tag{5.9}$$

The other variables $y_k := \tilde{y}_k, k \in I_s \setminus \{i_c\}$ remain unchanged in the constraint layer. Here again, we have to decide on how to select i_c, randomly, by performance or according to an additional inequality constraint. Choosing the index randomly can cause problems, e.g. predictions that result in negative masses. An example could be if the mass is borrowed from a mode that contains very little mass like nucleation mode.

Correction Layer The correction layer for the changes in mass and number takes into account the input x.

$$y_i^{\text{CorL}} = \text{ReLU}(\tilde{y}_i + x_i) - x_i \tag{5.10}$$

for $i = 1, \ldots, n_a$. For water content, we can directly apply a ReLU function

$$y_i^{\text{CorL}} = \text{ReLU}(\tilde{y}_i) \tag{5.11}$$

for $i = n_a + 1, \ldots, n_{\text{out}}$.

Normalized Reformulation Having multiple output variables that model different physical quantities they all span very different ranges. Training on one loss requires all outputs to be in a similar range. We apply z-scaling for each input x_i and output variable y_i. We subtract its mean μ_i^x, μ_i^y and divide it by its standard deviation σ_i^x, σ_i^y.

The constraints for mass conservation are then formulated as follows

$$\sum_{i \in I_s} y_i \cdot \sigma_i^y + \mu_i^y = \sum_{i \in I_s} y_i \cdot \sigma_i^y + \sum_{k \in I_s} \mu_k^y = 0 \tag{5.12}$$

for all species $s \in S$, with $h_s^{(eq)}(x) = \sum_{k \in I_s} \mu_k^y$. For mass and number positivity, we obtain

$$y_i \cdot \sigma_i^y + \mu_i^y + x_i \cdot \sigma_i^x + \mu_i^x \geq 0 \tag{5.13}$$

for $i = 1, \ldots, n_a$ and for water content

$$y_i \cdot \sigma_i^y + \mu_i^y \geq 0, \tag{5.14}$$

for $i = n_a + 1, \ldots, n_{\text{out}}$.

The formulation of the constraint layer, based on the unnormalized formulation from Eq. (5.8) for AddCL changes to the normalized version

$$y_i^{\text{AddCL}} = \frac{1}{\sigma_i^y}(\tilde{y}_i - \frac{1}{n_s}(\sum_{k\in I_s}\tilde{y}_k + \sum_{k\in I_s}\mu_k^y)). \tag{5.15}$$

The general multiplicative readjustment introduced in Section 3.3.2, adjusting the intermediate output by rescaling, can be reformulated for our mass conservation layer in the normalized setup

$$y_i^{\text{MultCL}} = \tilde{y}_i \cdot \frac{n_s \cdot \sum_{k\in I_s}\mu_k^y}{\sigma_i^y \sum_{k\in I_s}\tilde{y}_k}, \tag{5.16}$$

assuming $\sigma_i^y \sum_{k\in I_j}\tilde{y}_k \neq 0$.

The softmax constrained layer (SmCL), previously described in Section 3.3.2 as a different approach to multiplicative rescaling, can also be used to enforce mass conservation here, when applied on the normalized data

$$y_i^{\text{SmCL}} = \exp(\tilde{y}_i) \cdot \frac{n_s \cdot \sum_{k\in I_s}\mu_k^y}{\sigma_i^y \sum_{k\in I_s}\exp(\tilde{y}_k)}, \tag{5.17}$$

The completion layer, CompL, applied in the normalized case changes from previous Eq. (5.9) to

$$y_{i_c}^{\text{CompL}} = \frac{1}{\sigma_{i_c}^y}(-(\sum_{k\in I_s\setminus\{i_c\}}\tilde{y}_k + \sum_{k\in I_s}\mu_k^y)). \tag{5.18}$$

Finally, CorL, in the normalized case changes to

$$y_i^{\text{CorL}} = \frac{1}{\sigma_i^y}(\text{ReLU}(y_i \cdot \sigma_i^y + \mu_i^y + x_i \cdot \sigma_i^x + \mu_i^x) - \mu_i^y + x_i \cdot \sigma_i^x + \mu_i^x) \tag{5.19}$$

for $i = 1, \ldots, n_a$ and for water content

$$y_i^{\text{CorL}} = \frac{1}{\sigma_i^y}(\text{ReLU}(y_i \cdot \sigma_i^y + \mu_i^y) - \mu_i^y), \tag{5.20}$$

for $i = n_a + 1, \ldots, n_{\text{out}}$.

Combining Constraints Applying one constrained layer either enforces our equality (mass conservation) or inequality constraint (positivity), but not both at the same time. Here, it is possible to combine constraints. We combine soft and hard constraints. One combination includes a correction layer for positive enforcement and a mass conservation loss term (CorL+soft

mass). The other hybrid constraining method is a completion layer (online) with a penalizing term for changes resulting in negative masses and numbers (CompL+soft neg.). The two soft constraints, mass conservation loss, and positivity loss are also considered combined by including two extra terms in the loss function (soft n+m). Combinations of CorL and CompL as described in Section 3.4 could be helpful here as well, but are left for future work.

5.2 Data

To build our training and offline validation dataset, we run the ICON-HAM global climate model and write out the aerosol masses and numbers before and after applying M7 within ICON, as well as all other input and output variables of M7.

ICON-HAM Setup We use the ICON-A-HAM 2.3 version developed by Salzmann et al. [2022] and that is available on GitLab for ICON developers and researchers[2]. The grid used is the R2B4 grid[3], with a horizontal resolution of about 160 km. For more details on the ICON setup, see appendix, Section B.3.

Run Details We set up ICON-HAM on the ARCHER2 supercomputing cluster. We run ICON-HAM for five years from 2000-2004 at a timestep of 10 minutes and 47 pressure levels. The experiment is a standardized run of the Atmospheric Model Intercomparison Project (AMIP, Gates et al. [1999]). For more details on the run setup, see appendix, Section B.3.

To have a more manageable dataset, we sub-sample among time and locations for the ML development phase, obtaining around 5 million data points. Here, we need relatively quick training times to experiment with different architectural choices. A description of how the dataset is built exactly and how the subsampling is done can be found in the appendix, Section B.4.

Normalisation We experiment with different normalisation techniques, such as log transformation, normalisation between 0 and 1, and standardisation to the z-scale (minus mean + divided by standard deviation). We

[2] `https://git.iac.ethz.ch/hammoz/icon-hammoz/-/tree/hammoz/marc_salzmann?ref_type=heads`

[3] A RnBk grid divides the icosahedral edges into n parts, followed by k subsequent edge bisections

experience the best results in the original scale when using z-scaling. The scaling is applied to the changes and not the full values.

Data Splits We split the data into training, validation, and testing sets to perform hyperparameter optimisation (such as learning rate, batch size, activation layer, weight decay, number of layers, and nodes per layer) and offline evaluation. Here, we also explore different splittings, such as splitting randomly, in time, or for different locations. To create an offline test case that resembles the application in future times, we choose the split in time, training on the first four years, followed by six months each of validation and testing. Our training dataset contains around 4.4 million samples.

Datset Extension for Online Case For the final online retraining, we extend our dataset to cover more of the distribution and achieve stable runs. We retrain on all available years (including offline validation and training data) and add additional locations. A description of how the dataset is built exactly and how the subsampling is done can be found in the appendix, Section B.4.

5.3 Offline Experiments

Before implementing an emulator in a climate model, careful development and offline testing are crucial to find the architectures that show the most promise to perform well online[4].

Network Architecture We explore different machine learning approaches for emulating the M7 microphysics model, including random forest regression, gradient boosting, and a neural network. For more details on the different ML architectures see Section 2.1.4. Providing more expressivity, the neural network approach appears to be the most successful for this application. We present not only one neural network model here, but different constrained versions built on the same base architecture, and discuss the advantages and disadvantages of these different designs.

We employ a fully connected neural network, where the values are propagated from the input layer through multiple hidden to the output layer, using a combination of linear operations and non-linear activations, see Figure 5.1. We use a ReLU activation function and three hidden layers, each

[4]The code for this work can be found at `https://github.com/paulaharder/aerosol-microphysics-emulation`, it contains about 400 lines of Python code.

hidden layer containing 256 nodes. Using zero hidden layers results in linear regression and does not have the required expressivity for our task; one hidden layer already performs well, but after two layers of the model, we can not see any significant improvements. For more details on neural networks and the different layers see Section 2.2.

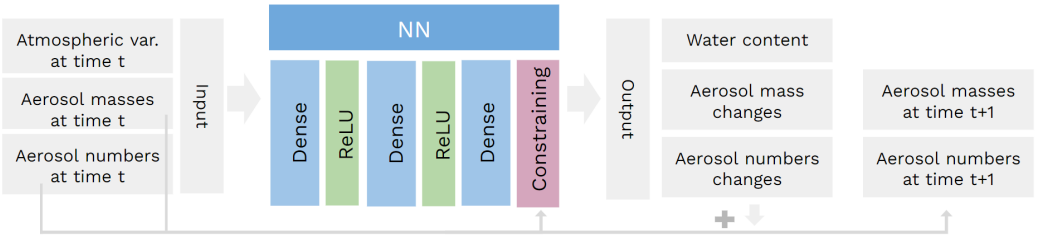

Figure 5.1: The offline data flow and NN architecture used here for M7 emulation, including a constraint layer (in red).

Constraint Formulation

Training We train all our NNs using the Adam optimizer with a learning rate of 10^{-3}, a weight decay of 10^{-9}, and a batch size of 4096. Our objective to optimize during neural network training is specified in Equation (5.6) and Equation (5.7), using an MSE loss in every version and activating the additional loss terms depending on the specified constraint. We train our models for 100 epochs. Training on a single NVIDIA Titan V GPU takes about 6 hours.

Baselines We explore simpler ML models such as linear regression, random forests, and gradient boosting. Given the limited expressivity, these methods do not achieve the same performance compared to NNs. Linear regression and random forests naturally provide less mass violation, making them attractive for emulation tasks. The results of these models can be found in Table 5.1.

5.4 Offline Results

Metrics We consider multiple metrics to obtain an insight into the performance of the different emulator designs, covering overall predictive accuracy, mass violation, and predicting non-physical values. We examine the R^2 score

(see Eq. (2.34)), the normalized RMSE, and a normalized bias for accuracy. To understand mass violation in our predictions we look at mass bias (MaB) and overall mass violation with a normalized RMSE (MaE), where all scores are normalized by the mean over the respective species. The metrics are completed with two scores on negative value predictions: An overall fraction of negative and, therefore non-physical predictions (NFrac) and the average negative extent per predicted value (NMean). We take the mean over three different random initialisations of the underlying neural network for all the different scores and architectures.

Other ML approaches

Apart from a neural network approach, we investigate other ML models to emulate the M7 module. We look at linear regression (LR) and different ensemble methods such as a random forest regressor (RF) and a gradient boosting model (GB), detailed descriptions of these models can be found in section 2.1.4. In Table 5.1 we report the test scores for linear regression, a random forest, and a gradient boosting approach. Being linear combinations of training points, linear regression, and random forest both automatically are close to conserving mass. Overall, the random forest shows the best accuracy among these models. We note that the random forest performs worse than the neural network in terms of predictive accuracy, but could still be considered for future work because of the mass conserving property. On other important factor though, is that RF and GB are both significantly slower than NNs, which makes them less useful as emulators.

Accuracy As shown in Table 5.2 and Figure 5.3, we achieve very good performance with a R^2 up to 0.87 evaluating on the predicted changes. We also achieve a normalized RSME of 0.26, and a normalized bias of 0.00045. The strongest models measured in the R^2 value are the unconstrained NN (None), the offline correction version (CorL offl.) and correction combined with a mass regularisation term (CorL+soft mass). Similar performance is achieved by soft mass-constraint, online correction, and completion versions. Soft positivity constraints reduce performance to a R^2 value of 0.68, unless combined with the completion layer, when it still reaches a good result of 0.80.

For variables, that can have both negative and positive changes, we can observe that the models sometimes struggle with predicting zeros. It can both happen that the true data is zero and the model predicts a non-zero change or that there is a non-zero change in the true data, but the model predicts zero. This shows in the prediction vs. truth plot as a cross structure

Table 5.1: The test scores for different ML approaches (linear regression (LR), random forests (RF) and gradient boosting (GB)) evaluating offline performance, including the unconstrained (NN) and two constrained versions of neural networks. NN+CorL+soft m. uses the correction layer (CorL) for positivity and a penalizing term for mass conservation (soft m.). NN+CompL+soft n. uses the completion layer (CompL) for mass conservation and a penalizing term for positivity (soft n.). MaB=Mass Bias, MaE=Mass Error in RMSE, NFrac= negative fraction, NMean= negative mean. Best in bold blue, second best in bold black.

Model	R^2 ↑	RMSE ↓	Bias ↓	MaB ↓	MaE ↓	NFrac ↓	NMean ↓
LR	0.20	0.65	$1.2e^{-3}$	$\mathbf{7.8e^{-10}}$	$\mathbf{1.3e^{-6}}$	0.26	$3.0e^{-3}$
RF	0.72	0.45	$2.7e^{-4}$	$3.7e^{-9}$	$4.3e^{-6}$	0.23	$3.2e^{-3}$
GB	0.53	0.57	$\mathbf{2.1e^{-4}}$	$6.3e^{-5}$	$2.5e^{-2}$	0.26	$9.0e^{-3}$
NN	**0.87**	**0.27**	$7.3e^{-4}$	$3.0e^{-5}$	$1.5e^{-3}$	0.20	$1.6e^{-3}$
NN +CorL +soft m.	**0.87**	**0.26**	$\mathbf{6.1e^{-4}}$	$2.5e^{-5}$	$3.2e^{-3}$	**0.00**	**0.00**
NN +CompL. +soft n.	0.80	0.29	$1.2e^{-2}$	**0.00**	**0.00**	**0.11**	$\mathbf{5.1e^{-4}}$

around zero, such as for DU AS, the fourth column in Figure 5.3 or less pronounced in the fifth column.

In the appendix (see Tables B.1, B.2 and B.3) we show the R^2 score for every 28 predicted variables individually. All different architectures perform similarly well on most the of variables (see also Figure 5.3). Only two variables achieve a R^2 score below 0.77, which are the coarse mode masses of sulfate and black carbon.

Constraint violations All hard-constrained methods successfully enforce either mass conservation or positivity. Correction and completion methods come with either no or minimal costs in terms of predictive accuracy, while the readjustment layers significantly decrease the R^2 values. Soft negativity constraining improves negative fraction, bringing it down to 0.17-0.11 and a slight decrease in negative magnitude when combined with CompL. Soft mass-constraining lowers the mass RMSE (MaE) but has no effect on the mass bias. The difference in online and offline application of CompL and

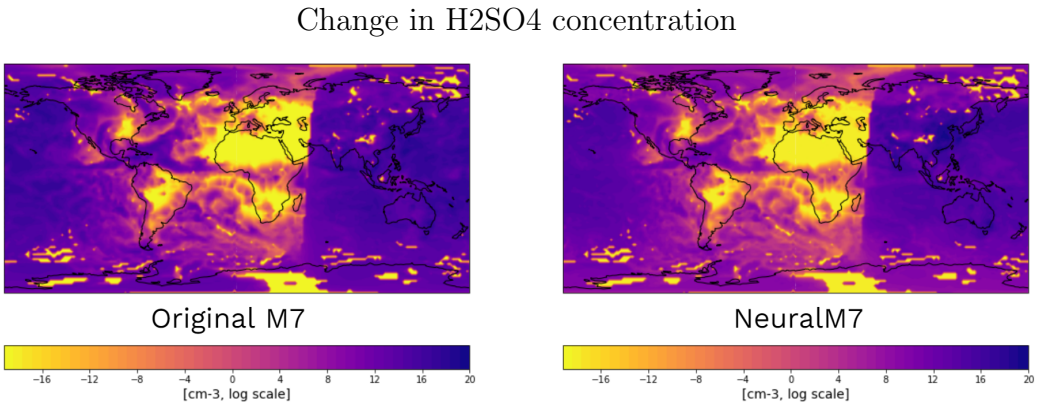

Figure 5.2: The change in H2SO4 concentration modeled by the M7 module for the first time step of the test data is plotted on the left (ground truth). The change predicted by our NN emulator NeuralM7 (base version - no constraints) is plotted on the right. Both plots show the change in concentration on a logarithmic scale.

CorL is almost negligible, with a slight better performance of the offline versions.

Runtime comparison We conduct an offline runtime analysis by comparing the Python runtime for the emulator with the Fortran runtime of the original aerosol model to evaluate the speed-up potential of NeuralM7. We take the time for one global time step, which is about 570,392 data points to predict. For the M7 model, the 31 vertical levels are calculated simultaneously, and for the emulator, we predict the one time step at once, using a batch size of 571,392 and taking into account the time for transforming variables and moving them on and off the GPU. We use a single NVIDIA Titan V GPU and a single Intel Xeon 4108 CPU. As shown in Table 5.3 we can achieve a large speed-up of over 11,000x in a pure GPU setting. Including the time it takes to move the data from the CPU onto the GPU and back, the acceleration is 64x compared to the original model. In case of no available GPU, the NN emulator is still 2.8 times faster.

5.5 Online Experiments

Here, we describe how the online experiments are conducted. First, we retrain the NN, move the weights through the pipeline described below (also see Figure 5.4), and then conduct an ICON-HAM run.

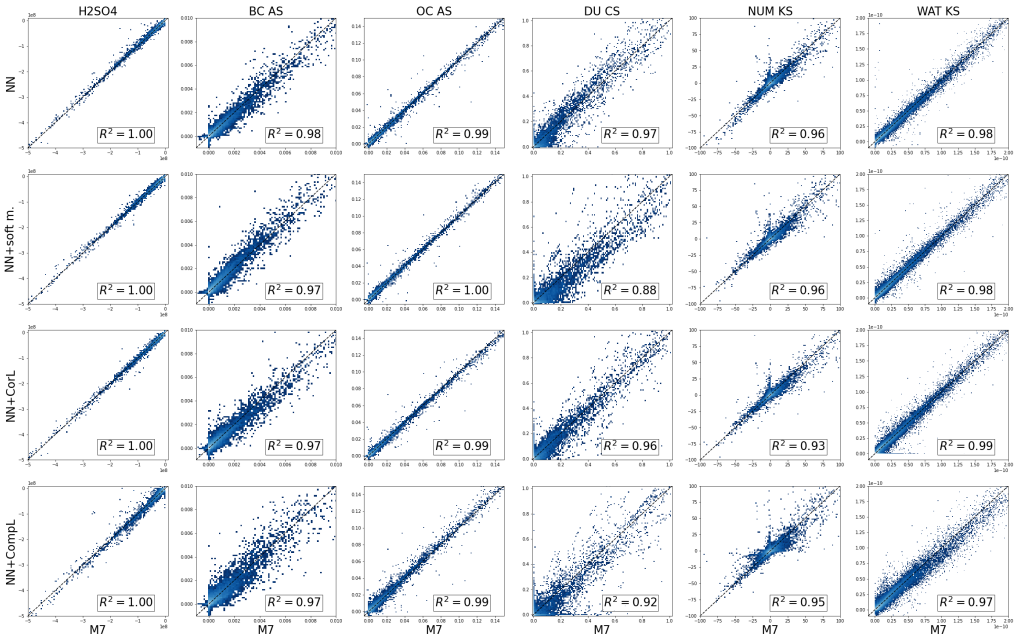

Figure 5.3: NN predictions vs. M7 original values on the test set, calculated for random variables per species/type. First row is the base NN, second soft mass-constrained NN, third correction-constrained NN and the last the completion-constrained NN. The brightness indicates the density of points in the scatter plot.

To run our aerosol emulator coupled to a global climate model, we need to move it from the original implementation in Python/Pytorch to an implementation in Fortran, the language used for almost all climate and weather models. We achieve this by transforming our trained weights through the Fortran-Keras Bridge library [Ott et al., 2020], which can then run the NN with pre-implemented layers in Fortran. Within the global climate model code, ICON-HAM in our case, we replace the original M7 function, `m7`, with a NN version, `neural_m7`. An overview of our end-to-end pipeline for NeuralM7 is shown in Figure 5.4. The first step was discussed in the previous section about offline experiments and results; here we go through steps 2 (Retraining), 3 (Fortran implementation), and 4 (ICON integration).

Table 5.2: Test scores for different constraining methods evaluating offline performance, to determine the most promising approaches to move forward to online experiments, Calculated on the held-out test set over 0.5M samples. The mean is taken over three runs. The best scores are highlighted in bold blue, second best in bold. MaB=Mass Bias, MaE=Mass Error in RMSE, NFrac= negative fraction, NMean= negative mean.

Model Constraint	R^2 ↑	RMSE ↓	Bias ↓	MaB ↓	MaE ↓	NFrac ↓	NMean ↓
None	**0.87**	0.27	$7.3e^{-4}$	$3.0e^{-5}$	$1.5e^{-3}$	0.20	$1.6e^{-3}$
soft neg.	0.68	0.46	$2.2e^{-2}$	$3.1e^{-3}$	$1.3e^{-2}$	0.15	$1.6e^{-3}$
soft mass	0.86	0.27	$1.3e^{-3}$	$3.1e^{-5}$	$9.7e^{-4}$	0.22	$1.4e^{-3}$
soft n+m	0.59	0.51	$3.8e^{-2}$	$1.7e^{-3}$	$2.1e^{-2}$	0.17	$4.2e^{-3}$
CorL off.	**0.87**	**0.26**	**$4.5e^{-4}$**	$2.1e^{-4}$	$2.5e^{-3}$	**0.00**	**0.00**
CorL on.	0.86	0.27	$7.7e^{-4}$	$1.2e^{-4}$	$4.3e^{-3}$	**0.00**	**0.00**
Compl. off.	0.85	0.27	$6.8e^{-4}$	**0.00**	**0.00**	0.21	$1.9e^{-3}$
Compl. on.	0.84	0.27	$8.2e^{-4}$	**0.00**	**0.00**	0.24	$2.3e^{-3}$
AddCL	0.17	0.77	$4.4e^{-3}$	**0.00**	**0.00**	0.32	$5.4e^{-2}$
MultCL	0.36	0.70	$6.9e^{-4}$	**0.00**	**0.00**	0.25	$2.2e^{-2}$
SmCL	0.35	0.71	$1.3e^{-3}$	**0.00**	**0.00**	0.27	$2.2e^{-2}$
CorL +soft m.	**0.87**	**0.26**	**$6.1e^{-4}$**	$2.5e^{-5}$	$3.2e^{-3}$	**0.00**	**0.00**
CompL +soft n.	0.80	0.29	$1.2e^{-2}$	**0.00**	**0.00**	0.11	$5.1e^{-4}$

Retraining

For online experiments, the training, validation, and testing split are no longer required; we will automatically encounter new samples when running the simulation. We experience that a training dataset that covers a larger part of the data distribution is essential for stability when running online. This motivates us to retrain on our entire dataset, including the validation and testing parts. Additionally, we later extend the dataset to include more locations. Our finally used dataset includes five years of simulation and spans more locations than the offline case, covering slightly less than one-third of the globe. A description of how the dataset is built exactly and how the subsampling is done can be found in Appendix, Section B.4.

The retrained Pytorch weights are then converted to Keras/Tensorflow

Table 5.3: Runtime comparison for the original M7 model and the NN emulator. NN pure GPU includes only the transformation of the variables and the prediction, whereas NN CPU-GPU-CPU also includes the time to transfer the data from the CPU to the GPU and back to the CPU. NN CPU is the case where the data and neural network stay on the CPU and the application of the NN is performed on the CPU.

Model	M7	NN pure GPU	NN CPU-GPU-CPU	NN CPU
↓ time (s)	5.781	0.000517	0.0897	2.042
↑ speed-up	-	11181.8	64.4	2.80

using the Open Neural Network Exchange (ONNX) ecosystem [Foundation, 2019] as an in-between step. This is done to accommodate the required format for the integral part of converting to Fortran using the Fortran-Keras-Bridge.

Implementation in Fortran

The implementation is mostly based on the Fortran-Keras-Bridge library that converts our weights to a Fortran compatible file and provides most required NN parts.

Fortran-Keras Bridge The Fortran-Keras Bridge (FKB) is a library that enables the implementation of NNs in Fortran. Here, different NN layer types are pre-implemented in Fortran. This covers simple layers, such as dense layers, batch normalisation, and different activation functions. The user has to provide the weights file from their trained NN to run the inference in Fortran using the FKB layers. The weights file is required to be in a Keras/Tensorflow format, then this is transformed into a txt file, specifying the layer types and including the weights. There is also the possibility to train directly in Fortran, but here we only use its inference features. One limitation of FKB is that it takes only a one-dimensional input and does not support batches, which decreases the speed compared to standard batch approaches

Our constraint layers are not included as pre-implemented layers in the FKB library. They can be easily implemented here as a separate, non-trainable layer at the end of the NN and are added to the Fortran code by us directly.

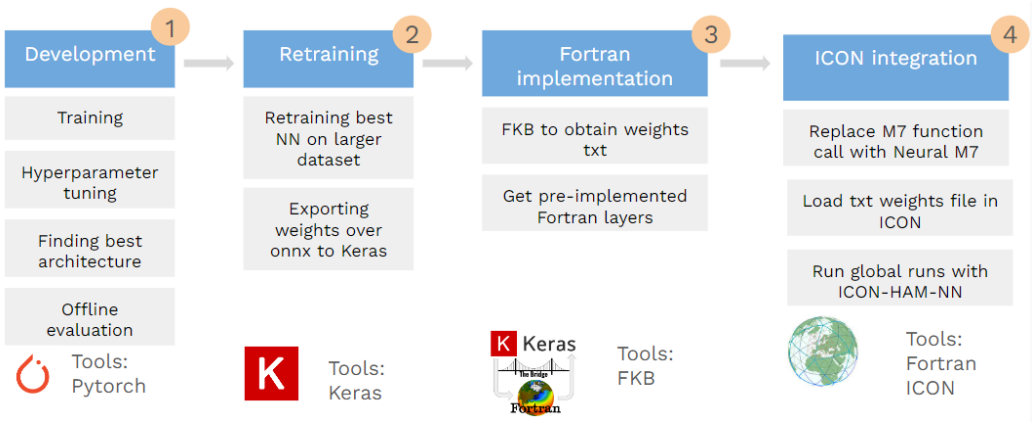

Figure 5.4: An overview of our NeuralM7 pipeline: The first phase consists of the development and offline testing in PyTorch. The second phase is retraining and conversion into Keras. The third phase applies the FKB library to obtain the Fortran implementation, and in the final fourth step, we include NeuralM7 in ICON-HAM..

NeuralM7 in ICON-HAM

The global climate-aerosol model ICON-HAM calls the function `m7` for each time step to update the aerosol microphysics properties. We introduce the new subroutine `neural_m7` that replaces the standard `m7`. The `neural_m7` function applies the following steps

1. Initializes all variables.
2. Standardizes and normalizes the inputs for the NN.
3. Applies the NN.
4. Applies a potential constraint layer.
5. Back-transforms outputs to the original scale.
6. Update aerosol masses, numbers and water content.

In addition to replacing M7, we need to extend the initialisation to load the NN's weights once and distribute copies of them among processes for parallel execution. The code exists as a branch within the ETH Zürich

ICON development Gitlab repository[5] and is available to ICON developers and researchers.

Runs We employ the ICON-HAM global aerosol-climate model as described in the previous data section (see Section 5.2). We run the model either for a year (starting in January 2000) or until the climate model crashes (for details on model crashes see Section B.5). We add a three-month spin-up period at the beginning, which is done using the original M7 module. For more details on the run setup, see in the appendix, Section B.3.

Testing To ensure correct results, we test the NN throughout the pipeline. A test case prediction is done first in Pytorch, secondly with the Keras weights, then with the offline Fortran implementation, and in the end with the weights loaded in ICON-HAM to ensure that at all stages the application of the NN gives the same outcome.

5.6 Online Results

In this section, we discuss the results achieved by performing our coupled runs of the NN emulator and ICON-HAM. We go through the aspects of stability and predictive accuracy and look at burdens and predicted variable values as well as aerosol properties. We compare the values after a year-long run including different versions of our emulator to a reference run. Finally, we discuss the runtime performance depending on the neural network size.

Stability

The first criterion for evaluating an emulator's performance is its stability. Even a very accurate surrogate is not of use if it does not result in a stable climate run when coupled. Table 5.4 shows the stability for all implemented NN architectures, as well as tables (see Tables 5.7 and 5.6) showing the stability depending on the size of the NN and the training set.

Simulation crashes, i.e., the occurrence of an error that leads to the simulation stopping, appear to be caused by some versions of our emulator. Here the error does not occur in the emulator code, but in later parts, such as the convection scheme, as a result of unrealistic predictions by the NN. More details can be found in the appendix (see Section B.5).

[5] `https://git.iac.ethz.ch/rherbe/icon-hammoz/-/tree/harder-neural-m7`, including 100 lines of Fortran code

Table 5.4: The stability of different NN setups, when coupled to ICON-HAM, measured in days until model crash. We run ICON for one year, after a three-month spin-up. Here, 365 days is the maximum that can be achieved. Best results are highlighted in bold blue.

run	Stab. (d)/365 ↑
None	0
Soft mass	**365**
CorL offl.	**365**
CorL onl.	19
CompL offl.	0
CompL onl.	0
CorL offl.+soft m.	**365**
CorL onl.+soft m.	26
CompL offl+soft m.	37

As shown in Table 5.4, three of the nine architectures achieve full stability. The initial NN, without any constraint included, does not achieve a single day of stable run. The completion layer does not help to provide a stable run despite its mass-conserving properties. While the mass loss term does not have a significant effect on offline performance, it stabilizes the NN performance for a full year in the online setting. Offline correction (enforcing mass positivity) and a combination of soft mass constraint and correction achieve full stability too.

Neural network size We experience a high sensitivity of stability towards the NN's size. Table 5.7 shows that smaller architectures tend to be less stable, especially when including fewer layers. Here, stability correlates with offline performance, which also increases with the neural network's size and depth. We observe that a further increase in depth or width of the NN does not aid stability or performance and can even harm it. Additionally, a bigger NN always increases computational costs.

Training Set Size The size and diversity of the training set significantly affect the stability of NeuralM7 as well. Table 5.6 shows that a training set is needed that spans more years and more locations to achieve a stable run. We use four different datasets. The first and smallest dataset consists of samples from only one year and from a fifth of the locations, the second dataset includes samples from an additional three, and the third from four

Table 5.5: R^2 scores (higher is better) for different emulators evaluating **online performance**, by considering aerosol properties and burdens. Best scores are highlighted in bold blue.

RUN	SOFT MASS	CORL OFFL.	CORL OFFL.+SOFT MASS
AOD 550NM	**0.89**	0.63	0.63
AOD 865NM	**0.88**	0.61	0.62
ABS. 550NM	**0.94**	0.84	0.82
AE. 550NM-865NM	0.79	0.75	**0.80**
BURDEN H2S04	0.90	**0.91**	**0.91**
BURDEN S04	0.80	**0.83**	**0.83**
BURDEN BC	0.93	**0.96**	0.93
BURDEN OC	0.94	**0.96**	0.93
BURDEN DU	**0.89**	0.59	0.62
BURDEN WAT	0.63	0.70	**0.75**

years, both still coming from the same location. The final fourth dataset samples from more locations, covering roughly one-third of the grid boxes and 5 years.

Table 5.6: Stability of online runs in days depending on the training set size. Here we use the soft mass-constrained architecture and train it on different data sets, samples from different spatial and temporal coverage. Bold blue highlights the best result.

DATA	SET1	SET2	SET3	SET4
YEARS	Y. 1	YRS. 1-4	YRS. 1-5	YRS. 1-5
SPATIAL COVERAGE	1/5	1/5	1/5	3/10
SET SIZE	0.5M	4.4M	5.5M	9M
↑ STAB. (D)/365	0	23	32	**365**

The proceeding accuracy analysis is limited to stable models only that provide a year of run data, i.e., the soft mass-constrained version (soft mass), the offline correction-constrained version (CorL offl.), and the combination of these two constraining methods (CorL offl. + soft mass).

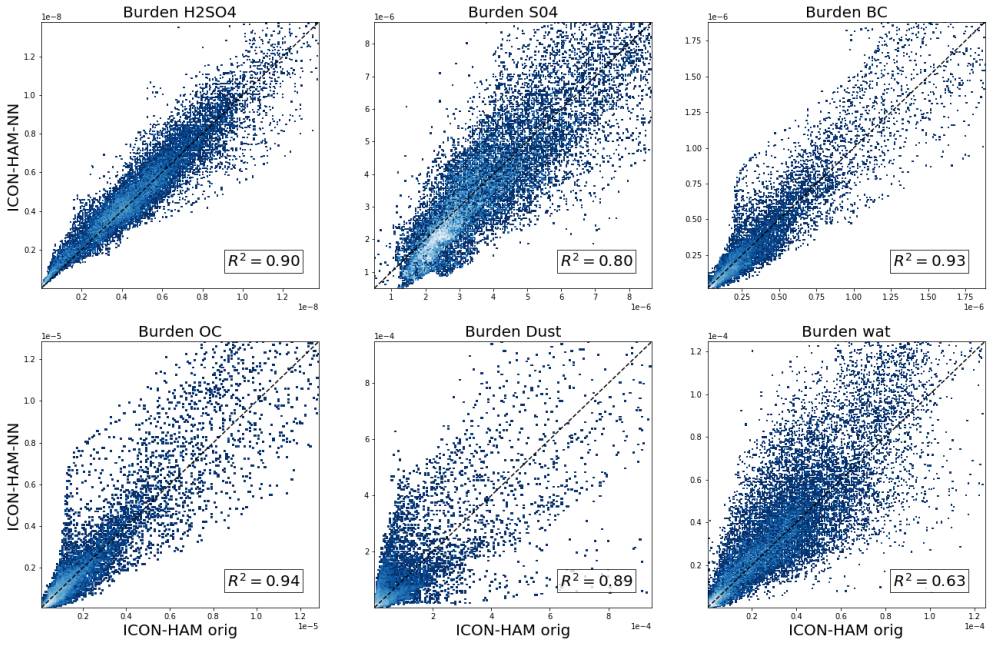

Figure 5.5: Aerosol burdens after 1 year run of ICON-HAM with our soft mass-constrained emulator compared to a reference run including the M7 original. Plots for the other two architectures can be found in the appendix. The brightness indicates the density of points in the scatter plot.

Additional Attempts to Stabilize Runs We try a variety of methods and ways to stabilize emulator runs, here are attempts that we implement to resolve the stability issues.

- Ensemble forecast: We implement a setup where different weight files are applied for inference and the mean of the prediction is then passed on to ICON. The different weights files can either come from the exact same architectural and hyperparameter setup and just differ based on different random initialisation, or include architectural differences. Ensembles are known to make predictions more stable, but in our case it does not prevent model crashes. A disadvantage of ensemble NNs is an increased computational demand.

- Separate NNs: Predicted variables can be divided in different subgroups, such as sulphate masses, black carbon masses, number concentration, etc.. We train individual NNs for each of those subsets and included them in ICON. Offline we can achieve slightly better performance than with one single NN. The idea for stability is that if one variable gets predicted off bounds, it does not directly impact other

variables. We still run into stability issues with this setting. This is to be expected as most M7 modes are internally mixed so all mass species in one mode undergo the same physical processes.

- Set hard upper-boundaries: We hard code upper boundaries for the predicted variables in Fortran, this then causes that in some grid boxes all the values saturate that boundary and stay there, still giving problems in later code parts.

- Interchanging physical prediction and NN: We conduct runs where a time step performed with NeuralM7 is followed by a time step performed by the original model. While this extends the time ICON-HAM runs stably, it did not resolve stability issues completely.

- Predicting full values: Instead of learning to predict the changes we train a NN to predict the full mass and number values at the next time step. Here we can easily force the positivity and set hard boundaries using a sigmoid as a last layer. The offline scores evaluated on the full values themselves achieve good scores, but when evaluated on changes they do ot achieve much predictive skill. In an online run, we achieve stable runs but the aerosol properties stay constant throughout the simulation, which makes predicting full values infeasible for our task.

Aerosol Properties We look at aerosol radiative properties that depend on their microphysics such as optical thickness, absorption optical depth, and Angstrom Exponent (AE) after one year of simulation, considering the monthly mean of all properties. Aerosol optical thickness, also called aerosol optical depth (AOD), measures the amount of light that gets redistributed through scattering and absorption due to the presence of aerosols on a vertical path through the atmosphere. AOD depends on the wavelength considered, here we include 550nm and 865nm. The absorption optical depth (Abs. OD) is the amount of light at a specific wavelength absorbed by aerosols. The Angstrom Exponent (AE) describes how AOD changes relative to the various wavelengths of light and is a valuable proxy of column-integrated aerosol size.

In Table 5.5 and Figure 5.6 we can see that all stable emulators are able to predict aerosol properties well. The soft mass-constrained neural network achieves very good results with R^2 values between 0.79 and 0.94.

Variables and Burdens Aerosol burdens are the column aggregated changes in aerosol masses or numbers. Taking into account values after one year of

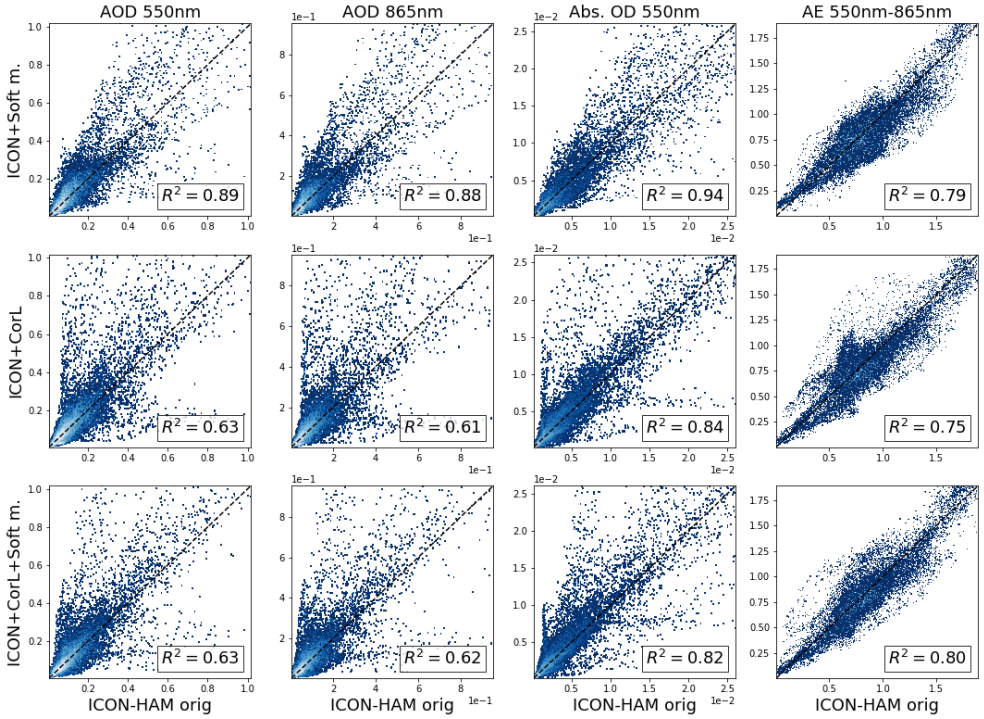

Figure 5.6: Aerosol properties, monthly mean, after a 1-year run of ICON-HAM with our three different stable architectures. The first row is the soft mass-constrained emulator compared to a reference run with the original M7, the second row is the emulator with CorL and the last combines CorL and soft mass-constraining. We show aerosol optical depth (AOD) at 550nm and 865nm, absorption optical depth (Abs. OD) at 550nm and the Angstrom Exponent (AE) from 550nm-865nm. The brightness indicates the density of points in the scatter plot.

running, we show the mass burdens for H2S04, SO4, BC, OC, dust, and water content in Figure 5.5 for the soft mass-constrained NN; the same plots for the CorL and CorL + soft mass networks can be found in the appendix (see Figures B.1 and B.2). Table 5.5 shows that we achieve very good or good results ranging from a R^2 value of 0.59 to 0.96 depending on the variable and model. For burdens, the correction NN performs better than the soft mass-constrained one, with the exception of dust burden.

Tables B.4, B.5, B.6 (see appendix) show the R^2 score for the monthly mean of every individual variable value after a year of simulation. Most

Table 5.7: Runtime comparison for the original M7 model and the NN emulators, when coupled with ICON, using four nodes and 128 processes per node. We report the time spend on `m7/neural_m7` for a month of a global run. We compare different sizes of NN, with the architectures' depth x width. Best scores are highlighted in bold blue.

Model	M7	2x16	2x32	2x64	1x256	2x128	2x256
#params	-	1.3k	3.1k	8.3k	16.4k	24.7k	82.2k
↓ time (s)	15	**11**	17	28	61	68	193
↑ stability (d)/365	365	2	2	4	2	14	**365**
↑ offl. R^2	1.0	0.66	0.70	0.72	0.39	0.79	**0.87**

of the 28 variables are well predicted, with R^2 reaching 0.70 to 0.96. The six weakest variables are coarse modes for black and organic carbon, water content accumulation mode as well as Aitken mode for sulphates and number concentration. The three different models perform all well on the same variables.

Runtime Comparison The runtime heavily depends on the number of parameters in the NN, which is determined by the number of layers and nodes per layer. The constraint layers do not noticeably influence the runtime. Table 5.7 shows the runtime of ICON-HAM-NeuralM7 compared to the original. With the current setup, a speed-up is achieved only with the smallest NN consisting of 16 nodes per hidden layer and two hidden layers. Our stable NN setup with two layers and 256 nodes per layer needs about ten times more time than the original. Compared to the offline runtime evaluation it is important to notice that here the batch size is one, only one sample is predicted by the NN at once. Here, tools are required to support parallelized batch execution of NN, or a setting where the NN can be run on a GPU to achieve its full potential. When considering the NN size, we observe a trade-off between stability and runtime, where bigger architectures perform better but are slower.

Chapter 6

Conclusion and Future Work

This chapter finalizes our work on the topic of physics-constrained DL for accelerating climate modeling. It is divided into a conclusion and an outlook section. The conclusion section briefly summarizes the results of this thesis and highlights our contributions within the field. The future work section describes how current limitations can be tackled in ongoing efforts and showcases interesting avenues for new research building upon this work.

6.1 Conclusion

There is a need for faster climate modeling to achieve more accurate predictions, make models easier to use, save energy while forecasting, or predict further into the future. The recent success of deep learning can be utilized for speeding up climate modeling both through replacing expensive model parts (emulation) and through running models at lower resolution and increasing the resolution afterward (downscaling). However, at the same time, we need to pay attention to physical constraints that are often violated when using deep learning.

This thesis introduces novel methodologies to incorporate generalized linear equality and inequality constraints into any deep learning architecture. Our framework covers both soft constraint methods by adding penalizers to the loss function and hard constraint methods by stacking a final constraint layer to the end of the NN. Our hard constraint layers guarantee a specified constraint to be satisfied, and we can showcase their successful application for downscaling and emulation. All of our constraining code is publicly available and usable for anyone. The constraints layers can be easily utilized for

any other problem; they are available as Pytorch modules.[1] [2]

In this work, we develop four different hard constraint layers for equality constraints and one hard constraint layer for inequality constraints. For equality constraints, this includes the readjustment layers AddCL, MultCL, and SmCL, and a completion layer, CompL. AddCL adds a small adjustment term depending on the input and the constraint violation, MultCL rescales the intermediate prediction by a factor depending on the violation, and SmCL is a softmax layer scaled by an additional factor depending on the constraints again. CompL works differently; it sets one variable per constraint subset depending on the other variable in a way that the constraints are satisfied. The correction layer, CorL, corrects the output so that inequality constraints, such as positivity, are satisfied. Constraint layers are usually included during training, to be considered by the optimizer. Though CompL and CorL can also be applied offline, at inference only. All these layers show exact enforcement of the desired properties, but depending on the task they affect the predictive performance differently.

For constrained downscaling, we show that our AddCL and SmCL readjustment layers perform well, across different deep learning architectures, upsampling factors, predicted quantities, and datasets. We demonstrate its ability to operate on both standard downscaling datasets and on data created by independent simulations with the WRF. Our constrained models are not only guaranteed to satisfy consistency such as mass conservation between LR and HR, but also increase predictive performance across metrics, such as RSME and SSIM, and across use cases. Compared to soft-constraining through the loss function, the hard-constraint methodology does not suffer from the common accuracy-constraints enforcement tradeoff and is easy to integrate without any tuning necessary. Our hard-constraining performance enhancement is not only limited to climate super-resolution but also noticeable in satellite imagery of the lunar surface. Within the climate context, our AddCL and SmCL constraint layers can help with common issues connected to deep learning applied to downscaling: it dampens the coastal effect, errors get lower in critical regions, and training is more stable. Hard-constraining can weaken performance if the enforced relationships are strongly violated in the true data (see NorESM data). Also, if a bias exists in the LR (or other input) it can be propagated to the HR prediction by constraining on the LR.

Apart from our major contribution to constrained downscaling, we also advance the field of deep learning for climate downscaling by introducing a novel neural network (NN) architecture to perform spatial and temporal

[1] `https://github.com/RolnickLab/constrained-downscaling`

[2] `https://github.com/paulaharder/aerosol-microphysics-emulation`

super-resolution simultaneously. It is the first model to tackle this problem and outperforms the CNN+interpolation baseline by combining an optical flow encoder-decoder frame interpolation architecture with a convolutional RNN. Moreover, we establish a new challenging dataset as a testbed for climate super-resolution, introducing the NorESM data as a downscaling problem[3].

When building an aerosol emulator, physical constraints appear naturally, making it an interesting second test case for constrained deep learning. The aerosol emulator shows slightly different behavior with respect to the constraints compared to downscaling. Hard constraint layers still ensure satisfaction of the constraint by construction, but here completion and correction are much more successful in keeping the original performance compared to readjustment layers. The online performance benefits from constraints, especially soft mass constraining and CorL that enable stable runs.

We are the first to build an emulator for the M7 aerosol microphysics model, predicting changes in aerosol masses and numbers. We achieve very good offline performance, with a R^2 score of 0.87, using a fully-connected, 2-layer NN that significantly outperforms linear regression random forests and gradient boosting.

Unlike many works in the field, we take an extra step towards deployment and implement and test our emulator in a global climate model, using the Fortran-Keras bridge library. We showcase stable coupled runs with ICON as well as an accurate prediction of aerosol properties, such as AOD with an R^2 of 0.89 with respect to the reference run after one year of simulation. One observation for online runs was that NNs could easily become unstable, predicting unrealistic or unphysical values, especially when encountering out-of-the-distribution samples. This made it necessary to extend our training dataset size several times until we reached stable runs. While increasing the training set size and its variety, we also need to increase the NN's size, running into a size-stability and with that a speedup-stability tradeoff. With the current library being used that only supports a single batch size, the NN is slower than the original, here further work needs to be invested to achieve the speed-up building on the promising offline runtimes.

In summary, this work advances the field of constrained deep learning methodologically by developing new layers. We showcase how downscaling and emulation can benefit from these constraints. The development of an aerosol microphysics emulator and its integration into ICON demonstrated both the promise and difficulties of neural network emulators.

[3] `https://drive.google.com/file/d/1D5tLE7cGcvh3dap-P3VOLEOK_7FqdChF/view?usp=sharing`

6.2 Future Work

This thesis results in a variety of exciting future avenues for research, from examples, like extending the constraint methods over new downscaling architectures, to accelerating the online emulator

Although our generalized linear formulation for constraints covers all the application cases we encountered and is relatively straightforward to enforce, it is interesting to extend the set of constraints that can be addressed using a hard constraining methodology. Here, future research can be done on incorporating non-linear constraints and finding appropriate application cases that benefit from that. Existing constraint layers such as SmCL can be enhanced by including a learnable parameter, and soft-constraining can be potentially advanced through the scheduling of the regularisation factor. More combinations of our introduced constraining methods could be explored to achieve inequality and equality constraint satisfaction simultaneously.

Our work focuses on the effect of constraints in different architectures, and less on pushing downscaling performance to its maximum. Here, climate super-resolution performance could be enhanced by including additional information such as by incorporating high-resolution topography. The constrained downscaling work can be extended to any NN architecture. As the field advances quickly, new methods come out regularly. The first works building on our ideas use a constrained Fourier Neural Operator for arbitrary resolution downscaling [Yang et al., 2023]. Additionally, for a perfect prognosis setup, the idea of hard inequality constraints via ReLU has been adapted from our work [González-Abad et al., 2023]. Diffusion-based super-resolution methods, that recently appeared as a popular probabilistic method in climate science, can also benefit from a hard constraint layer when applied to climate downscaling. Transformer-based architectures, state-of-the-art in SR for computer vision, could be adopted for deterministic downscaling and enhanced with constraint layers. Splitting up a DL downscaling task into bias correction and super-resolution parts can enable constraints to be more successful when LR and HR are too far apart (e.g. for our NorESM data).

An open question for constrained climate downscaling is a concrete application case that includes a downstream task. For post-processing purposes, the offline application of our method, our code is readily available. To deploy these constrained super-resolution methods online, the next step is to use Fortran-Python bridges [Ott et al., 2020] to include them in global climate model runs.

There are a variety of possibilities to improve the final online performance of our aerosol emulator. There are multiple options to improve stability and accuracy. One new pathway is an active learning approach, in which we can

utilize our developed aerosol box model (see Appendix B.1) to produce new output from whatever input we need. The speed of our emulator can be increased in various ways. As we established in our current setup, the inference time depends heavily on the number of nodes in the NN. Pruning methods can help to achieve a smaller architecture. Utilizing different libraries that support parallelized batch predictions for NNs can help to accelerate the emulator while remaining at the same size. Before employing an emulator such as NeuralM7 for climate forecasting, it is necessary to ensure that ICON-HAM-NeuralM7 reproduces historical climate and forcing of aerosols in a multi-year run. Given the expressive power of NNs but the limited ability to speed up already efficient small submodels in climate models, an exciting future project is emulating not only the aerosol microphysics but the entire aerosol model HAM.

This thesis contributes significantly to advancements in the integration of deep learning with climate modeling, presenting novel methodologies that ensure physical constraints are respected while accelerating simulation processes. Looking forward, our constraining methods can be beneficial for a variety of application tasks, and our online evaluation can aid future work with the challenging integration of deep learning within existing climate model setups.

Bibliography

B. A. Albrecht. Aerosols, cloud microphysics, and fractional cloudiness. *Science*, 245(4923):1227–1230, 1989. ISSN 0036-8075. doi: 10.1126/science.245.4923.1227.

T. Arcomano, I. Szunyogh, A. Wikner, B. R. Hunt, and E. Ott. A hybrid atmospheric model incorporating machine learning can capture dynamical processes not captured by its physics-based component. *Geophysical Research Letters*, 50, 2023.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *ArXiv*, abs/1701.07875, 2017.

G. A. R. Auger, C. D. Watson, and H. R. Kolar. The influence of weather forecast resolution on the circulation of lake george, ny. *Water Resources Research*, 57(10):e2020WR029552, 2021. doi: https://doi.org/10.1029/2020WR029552. e2020WR029552 2020WR029552.

J. Baño Medina, R. Manzanas, and J. M. Gutiérrez. Configuration and intercomparison of deep learning neural models for statistical downscaling. *Geoscientific Model Development*, 13(4):2109–2124, 2020. doi: 10.5194/gmd-13-2109-2020.

T. Bäck. *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

J. Baño-Medina, R. Manzanas, and J. M. Gutiérrez. On the suitability of deep convolutional neural networks for continental-wide downscaling of climate change projections. *Climate Dynamics*, 57:2941 – 2951, 2021.

N. Bellouin, J. Quaas, E. Gryspeerdt, S. Kinne, P. Stier, D. Watson-Parris, O. Boucher, K. S. Carslaw, M. Christensen, A.-L. Daniau, and e. a. Dufresne. Bounding global aerosol radiative forcing of climate change.

Reviews of Geophysics, 58(1):e2019RG000660, 2020. doi: https://doi.org/10.1029/2019RG000660. e2019RG000660 10.1029/2019RG000660.

Z. Ben-Bouallegue, M. C. A. Clare, L. Magnusson, E. Gascon, M. Maier-Gerber, M. Janousek, M. Rodwell, F. Pinault, J. S. Dramsch, S. T. K. Lang, B. Raoult, F. Rabier, M. Chevallier, I. Sandu, P. Dueben, M. Chantry, and F. Pappenberger. The rise of data-driven weather forecasting, 2023.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. In *Journal of Machine Learning Research*, volume 13, pages 281–305, 2012.

G. Bertoli, F. Ozdemir, S. Schemm, and F. Pérez-Cruz. Revisiting machine learning approaches for short- and longwave radiation inference in weather and climate models, part i: Offline performance, 2023.

T. Beucler, S. Rasp, M. Pritchard, and P. Gentine. Achieving conservation of energy in neural network emulators for climate modeling, 2019.

T. Beucler, I. Ebert-Uphoff, S. Rasp, M. S. Pritchard, and P. Gentine. Machine learning for clouds and climate, 2020a.

T. Beucler, M. Pritchard, P. Gentine, and S. Rasp. Towards physically-consistent, data-driven models of convection, 2020b.

T. Beucler, M. Pritchard, S. Rasp, J. Ott, P. Baldi, and P. Gentine. Enforcing analytic constraints in neural networks emulating physical systems. *Phys. Rev. Lett.*, 126:098302, Mar 2021. doi: 10.1103/PhysRevLett.126.098302.

K. Bi, L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian. Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast. *ArXiv*, abs/2211.02556, 2022.

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. S. Chatterji, A. S. Chen, K. A. Creel, J. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. D. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. F. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. S. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair,

A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. F. Nyarko, G. Ogut, L. J. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. H. Roohani, C. Ruiz, J. Ryan, C. R'e, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. P. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. A. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang. On the opportunities and risks of foundation models. *ArXiv*, abs/2108.07258, 2021.

N. D. Brenowitz and C. S. Bretherton. Prognostic validation of a neural network unified physics parameterization. *Geophysical Research Letters*, 45:6289 – 6298, 2018.

N. D. Brenowitz, B. Henn, J. McGibbon, S. K. Clark, A. Kwa, W. A. Perkins, O. Watt-Meyer, and C. S. Bretherton. Machine learning climate model dynamics: Offline versus online performance. *arXiv: Atmospheric and Oceanic Physics*, 2020.

C. Brochet, L. Raynaud, N. Thome, M. Plu, and C. Rambour. Multivariate emulation of kilometer-scale numerical weather predictions with generative adversarial networks: a proof-of-concept. *Artificial Intelligence for the Earth Systems*, 2023.

J. Broecker and L. A. Smith. *Increasing the Reliability of Reliability Diagrams*, volume 22. Weather and Forecasting, 2007.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.*, 34:18–42, 2016.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. J. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

S. R. Cachay, V. Ramesh, J. N. S. Cole, H. Barker, and D. Rolnick. ClimART: A benchmark dataset for emulating atmospheric radiative transfer in weather and climate models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

S. R. Cachay, B. Zhao, H. James, and R. Yu. Dyffusion: A dynamics-informed diffusion model for spatiotemporal forecasting. *ArXiv*, abs/2306.01984, 2023.

M. Chantry, S. Hatfield, P. Dueben, I. Polichtchouk, and T. N. Palmer. Machine learning emulation of gravity wave drag in numerical weather forecasting. *Journal of Advances in Modeling Earth Systems*, 13, 2021.

C. Chaudhuri and C. Robertson. Cligan: A structurally sensitive convolutional neural network model for statistical downscaling of precipitation from multi-model ensembles. *Water*, 2020.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

X. Chen, K. Feng, N. Liu, B. Ni, Y. Lu, Z. Tong, and Z. Liu. Rainnet: A large-scale imagery dataset and benchmark for spatial precipitation downscaling. In *Neural Information Processing Systems*, 2020.

X. Chen, X. Wang, J. Zhou, and C. Dong. Activating more pixels in image super-resolution transformer. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22367–22377, 2022.

F. Chevallier, F. Chéruy, N. A. Scott, and A. Chédin. A neural network approach for a fast and accurate computation of a longwave radiative budget. *Journal of Applied Meteorology*, 37:1385–1397, 1998.

K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

M. Chu and N. Thuerey. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics*, 36(4):1–14, jul 2017. doi: 10.1145/3072959.3073643.

C. J. Darken, J. T. Chang, and J. E. Moody. Learning rate schedules for faster stochastic gradient search. *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, pages 3–12, 1992.

C. O. de Burgh-Day and T. Leeuwenburg. Machine learning for numerical weather and climate modelling: a review. *Geoscientific Model Development*, 16(22):6433–6477, 2023. doi: 10.5194/gmd-16-6433-2023.

J. Delgado-Centeno, P. Harder, V. Bickel, B. Moseley, F. Kalaitzis, S. Ganju, and M. Olivares-Mendez. Superresolution of lunar satellite images for enhanced robotic traverse planning: Maximizing the value of existing data products for space robotics. *IEEE Robotics & Automation Magazine*, pages 2–14, 2023. doi: 10.1109/MRA.2023.3276267.

J. Delgano-Centeno, P. Harder, B. Moseley, V. Bickel, S. Ganju, F. Kalaitzis, and M. Olivares-Mendez. Single image super-resolution with uncertainty estimation for lunar satellite images. *NeurIPS Workshop ML for Physical Sciences*, 2021.

C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:295–307, 2014.

C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016. doi: 10.1109/TPAMI.2015.2439281.

P. Donti, D. Rolnick, and J. Z. Kolter. Dc3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021.

P. Dueben, M. G. Schultz, M. Chantry, D. J. Gagne, D. Hall, and A. McGovern. Challenges and benchmark datasets for machine learning in the atmospheric sciences: Definition, status and outlook. *Artificial Intelligence for the Earth Systems*, 2022.

P. D. Dueben and P. Bauer. Challenges and design choices for global weather and climate models based on machine learning. *Geoscientific Model Development*, 2018.

B. Fallah, C. Menz, E. Russo, P. Harder, P. Hoffmann, I. Didovets, and F. F. Hattermann. Climate model downscaling in central asia: a dynamical and a neural network approach. *Geoscientific Model Development Discussions*, 2023:1–27, 2023.

P. Forster, T. Storelvmo, K. Armour, W. Collins, J.-L. Dufresne, D. Frame, D. Lunt, T. Mauritsen, M. Palmer, M. Watanabe, M. Wild, and H. Zhang. *The Earth's Energy Budget, Climate Feedbacks, and Climate Sensitivity*, page 923–1054. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA, 2021. doi: 10.1017/9781009157896.009.

T. L. Foundation. Onnx, 2019. URL `https://onnx.ai/`. Accessed on 16th Jan, 2024.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

W. L. Gates, J. S. Boyle, C. Covey, C. G. Dease, C. M. Doutriaux, R. S. Drach, M. Fiorino, P. J. Gleckler, J. J. Hnilo, S. M. Marlais, T. J. Phillips, G. Potter, B. D. Santer, K. R. Sperber, K. E. Taylor, and D. N. Williams. An overview of the results of the atmospheric model intercomparison project (amip i). *Bulletin of the American Meteorological Society*, 80:29–55, 1999.

A. Geiss and J. C. Hardin. Strictly enforcing invertibility and conservation in cnn-based super resolution for scientific datasets. *Artificial Intelligence for the Earth Systems*, 2(1):e210012, 2023. doi: https://doi.org/10.1175/AIES-D-21-0012.1.

A. Geiss, S. Silva, and J. Hardin. Downscaling atmospheric chemistry simulations with physically consistent deep learning. *Geoscientific Model Development Discussions*, 2022:1–26, 2022. doi: 10.5194/gmd-2022-76.

A. Geiss, P. Ma, B. Singh, and J. C. Hardin. Emulating aerosol optics with randomly generated neural networks. *Geoscientific Model Development*, 2023.

P. Gentine, M. Pritchard, S. Rasp, G. Reinaudi, and G. Yacalis. Could machine learning break the convection parameterization deadlock? *Geophysical Research Letters*, 45(11):5742–5751, 2018a. doi: https://doi.org/10.1029/2018GL078202.

P. Gentine, M. S. Pritchard, S. Rasp, G. Reinaudi, and G. Yacalis. Could machine learning break the convection parameterization deadlock? *Geophysical Research Letters*, 45:5742 – 5751, 2018b.

A. Gettelman, D. J. Gagne, C.-C. Chen, M. W. Christensen, Z. J. Lebo, H. Morrison, and G. Gantos. Machine learning the warm rain process. *Journal of Advances in Modeling Earth Systems*, 13(2):e2020MS002268, 2021. doi: https://doi.org/10.1029/2020MS002268. e2020MS002268 2020MS002268.

R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, 2006.

J. González-Abad, Á. Hernández-García, P. Harder, D. Rolnick, and J. M. Gutiérrez. Multi-variable hard physical constraints for climate model downscaling. In *Proceedings of the AAAI Symposium Series*, volume 2, pages 62–67, 2023.

J. González-Abad, Álex Hernández-García, P. Harder, D. Rolnick, and J. M. Gutiérrez. Multi-variable hard physical constraints for climate model downscaling, 2023.

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

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

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63:139 – 144, 2014b.

B. Groenke, L. Madaus, and C. Monteleoni. Climalign: Unsupervised statistical downscaling of climate variables via normalizing flows. In *Proceedings of the 10th International Conference on Climate Informatics*, CI2020, page 60–66, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450388481. doi: 10.1145/3429309.3429318.

M. Hameed, F. Yang, S. U. Bazai, M. I. Ghafoor, A. Alshehri, I. Khan, S. Ullah, M. Baryalai, F. H. Jaskani, and M. Andualem. Convolutional autoencoder-based deep learning approach for aerosol emission detection using lidar dataset. *J. Sensors*, 2022:1–17, 2022.

P. Harder, D. Watson-Parris, D. Strassel, N. Gauger, P. Stier, and J. Keuper. Emulating aerosol microphysics with a machine learning. In *ICML 2021 Workshop on Tackling Climate Change with Machine Learning*, 2021.

P. Harder, D. Watson-Parris, P. Stier, D. Strassel, N. R. Gauger, and J. Keuper. Physics-informed learning of aerosol microphysics. *Environmental Data Science*, 1:e20, 2022a. doi: 10.1017/eds.2022.22.

P. Harder, Q. Yang, V. Ramesh, P. Sattigeri, A. Hernandez-Garcia, C. D. Watson, D. Szwarcman, and D. Rolnick. Generating physically-consistent high-resolution climate data with hard-constrained neural networks. In *NeurIPS 2022 Workshop on Tackling Climate Change with Machine Learning*, 2022b.

P. Harder, A. Hernandez-Garcia, V. Ramesh, Q. Yang, P. Sattegeri, D. Szwarcman, C. Watson, and D. Rolnick. Hard-constrained deep learning for climate downscaling. *Journal of Machine Learning Research*, 24 (365):1–40, 2023a.

P. Harder, A. Kwa, A. Perkins, and C. S. Bretherton. Reservoir computing for sea surface temperature prediction in earth system digital twins. *AGU23*, 2023b.

N. Harilal, M. Singh, and U. Bhatia. Augmented convolutional lstms for generation of high-resolution climate change projections. *IEEE Access*, 9: 25208–25218, 2021. doi: 10.1109/ACCESS.2021.3057500.

L. Harris, A. T. T. McRae, M. Chantry, P. D. Dueben, and T. N. Palmer. A generative deep learning approach to stochastic downscaling of precipitation forecasts. *Journal of Advances in Modeling Earth Systems*, 14, 2022.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

H. Hersbach, B. Bell, P. Berrisford, S. Hirahara, A. Horányi, J. Muñoz-Sabater, J. Nicolas, C. Peubey, R. Radu, D. Schepers, A. Simmons, C. Soci, S. Abdalla, X. Abellan, G. Balsamo, P. Bechtold, G. Biavati, J. Bidlot, M. Bonavita, G. De Chiara, P. Dahlgren, D. Dee, M. Diamantakis, R. Dragani, J. Flemming, R. Forbes, M. Fuentes, A. Geer, L. Haimberger, S. Healy, R. J. Hogan, E. Hólm, M. Janisková, S. Keeley, P. Laloyaux, P. Lopez, C. Lupu, G. Radnoti, P. de Rosnay, I. Rozum, F. Vamborg, S. Villaume, and J.-N. Thépaut. The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020. doi: https://doi.org/10.1002/qj.3803.

P. Hess, M. Drüke, S. Petri, F. M. Strnad, and N. Boers. Physically constrained generative adversarial networks for improving precipitation fields from earth system models. *Nature Machine Intelligence*, 4, 2022.

G. Hinton. Neural networks for machine learning, 2012. Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude.

T. K. Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

K. Hornik, M. B. Stinchcombe, and H. L. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

J.-B. Huang, A. Singh, and N. Ahuja. Single image super-resolution from transformed self-exemplars. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5197–5206, 2015.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

IPCC. *Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA, 2013. ISBN ISBN 978-1-107-66182-0. doi: 10.1017/CBO9781107415324.

P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2016.

C. M. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, K. Kashinath, M. Mustafa, H. A. Tchelepi, P. Marcus, Prabhat, and A. Anandkumar. Meshfreeflownet: A physics-constrained deep continuous space-time super-resolution framework. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

J. Kaltenborn, C. E. E. Lange, V. Ramesh, P. Brouillard, Y. Gurwicz, C. Nagda, J. Runge, P. Nowack, and D. Rolnick. Climateset: A large-scale climate model dataset for machine learning. *ArXiv*, abs/2311.03721, 2023.

K. Kashinath, M. Mustafa, A. Albert, J.-L. Wu, C. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, R. Wang, A. Chattopadhyay, A. Singh, A. Manepalli, D. B. Chirila, R. Yu, R. Walters, B. White, H. Xiao, H. A. Tchelepi, P. S. Marcus, A. Anandkumar, P. Hassanzadeh, and Prabhat. Physics-informed machine learning: case studies for weather and climate modelling. *Philosophical Transactions of the Royal Society A*, 379, 2021.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

R. Kriegmair, Y. Ruckstuhl, S. Rasp, and G. C. Craig. Using neural networks to improve simulations in the gray zone. *Nonlinear Processes in Geophysics*, 2021.

R. Kurinchi-Vendhan, B. Lütjens, R. Gupta, L. Werner, and D. Newman. Wisosuper: Benchmarking super-resolution methods on wind and solar data, 2021.

R. R. Lam, A. Sanchez-Gonzalez, M. Willson, P. Wirnsberger, M. Fortunato, A. Pritzel, S. V. Ravuri, T. Ewalds, F. Alet, Z. Eaton-Rosen, W. Hu, A. Merose, S. Hoyer, G. Holland, J. Stott, O. Vinyals, S. Mohamed, and P. W. Battaglia. Graphcast: Learning skillful medium-range global weather forecasting. *ArXiv*, abs/2212.12794, 2022.

M. Langguth, P. Harder, I. Schicker, A. Patnala, S. Lehner, K. Mayer, and M. Dabernig. A benchmark dataset for meteorological downscaling. In *ICLR 2024 Workshop on Tackling Climate Change with Machine Learning*, 2024.

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

C. Ledig, L. Theis, F. Huszár, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114, 2016a.

C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2016b.

C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

J. Leinonen, D. Nerini, and A. Berne. Stochastic super-resolution for downscaling time-evolving atmospheric fields with a generative adversarial network. *IEEE Transactions on Geoscience and Remote Sensing*, 59(9):7211–7223, 2021. doi: 10.1109/TGRS.2020.3032790.

C. Lessig, I. Luise, B. Gong, M. Langguth, S. Stadtler, and M. G. Schultz. Atmorep: A stochastic model of atmosphere dynamics using large scale representation learning. *ArXiv*, abs/2308.13280, 2023.

R. Lguensat, M. Sun, R. Fablet, E. Mason, P. Tandeo, and G. Chen. Eddynet: A deep neural network for pixel-wise classification of oceanic eddies. *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 1764–1767, 2017.

J. Liang, J. Cao, G. Sun, K. Zhang, L. V. Gool, and R. Timofte. Swinir: Image restoration using swin transformer. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 1833–1844, 2021.

B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee. Enhanced deep residual networks for single image super-resolution. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1132–1140, 2017. doi: 10.1109/CVPRW.2017.151.

Y. Liu, E. Racah, Prabhat, J. Correa, A. Khosrowshahi, D. A. Lavers, K. E. Kunkel, M. F. Wehner, and W. D. Collins. Application of deep convolutional neural networks for detecting extreme weather in climate datasets. *ArXiv*, abs/1605.01156, 2016.

Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala. Video frame synthesis using deep voxel flow. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4473–4481, 2017. doi: 10.1109/ICCV.2017.478.

E. N. Lorenz. Predictability of weather and climate: Predictability – a problem partly solved, 2006.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017.

X. Lu, J. Wang, Y. Yan, L. Zhou, and W. Ma. Estimating hourly pm2.5 concentrations using himawari-8 aod and a dbscan-modified deep learning model over the yrdua, china. *Atmospheric Pollution Research*, 2020.

A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *Proc. icml*, 30(1):3, 2013.

D. Maraun and M. Widmann. *Statistical Downscaling and Bias Correction for Climate Research*. Cambridge University Press, 2018. doi: 10.1017/9781107588783.

M. Mardani, N. Brenowitz, Y. Cohen, J. Pathak, C.-Y. Chen, C.-C. Liu, A. Vahdat, K. Kashinath, J. Kautz, and M. S. Pritchard. Generative residual diffusion modeling for km-scale atmospheric downscaling. *ArXiv*, abs/2309.15214, 2023.

M. Martens, D. Izzo, A. Krvzivc, and D. Cox. Super-resolution of proba-v images using convolutional neural networks. *Astrodynamics*, 3:387 – 402, 2019.

D. R. Martin, C. C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, 2:416–423 vol.2, 2001.

D. Meyer, S. Grimmond, P. D. Dueben, R. J. Hogan, and M. van Reeuwijk. Machine learning emulation of urban land surface processes. *Journal of Advances in Modeling Earth Systems*, 14, 2021.

J. Millar, P. Harder, L. Freischem, P. Weiss, and P. Stier. Towards downscaling global aod with machine learning. In *ICLR 2024 Workshop on Tackling Climate Change with Machine Learning*, 2024.

M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv: Learning*, 2014a.

M. Mirza and S. Osindero. Conditional generative adversarial nets, 2014b.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020.

A. A. Nair and F. Yu. Using machine learning to derive cloud condensation nuclei number concentrations from commonly available measurements. *Atmospheric Chemistry and Physics*, 20(21):12853–12869, 2020. doi: 10.5194/acp-20-12853-2020.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. *In Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

N. M. Nawi, W. H. Atomi, and M. Rehman. The effect of data pre-processing on optimized training of artificial neural networks. *Procedia Technology*, 11:32–39, 2013. ISSN 2212-0173. doi: https://doi.org/10.1016/j.protcy.2013.12.159. 4th International Conference on Electrical Engineering and Informatics, ICEEI 2013.

Y. Nesterov. A method for solving the convex programming problem with convergence rate o(1/k$\hat{2}$). *Dokl. Akad. Nauk SSSR*, 269:543–547, 1983.

D. T. Ngo, O. Aouedi, K. Piamrat, T. Hassan, and P. R. Parvédy. Empowering digital twin for future networks with graph neural networks: Overview, enabling technologies, challenges, and opportunities. *Future Internet*, 15: 377, 2023.

T. Nguyen, J. Brandstetter, A. Kapoor, J. K. Gupta, and A. Grover. Climax: A foundation model for weather and climate. In *International Conference on Machine Learning*, 2023a.

T. Nguyen, J. Jewik, H. Bansal, P. Sharma, and A. Grover. Climatelearn: Benchmarking machine learning for weather and climate modeling. *ArXiv*, abs/2307.01909, 2023b.

P. A. O'Gorman and J. G. Dwyer. Using machine learning to parameterize moist convection: Potential for modeling of climate, climate change, and extreme events. *Journal of Advances in Modeling Earth Systems*, 10(10): 2548–2563, 2018. doi: https://doi.org/10.1029/2018MS001351.

O. Okonkwo, R. Patel, R. D. Gudi, and P. Biswas. A machine learning based approach to solve the aerosol dynamics coagulation model. *Aerosol Science and Technology*, 57:1098 – 1116, 2023.

J. Ott, M. Pritchard, N. Best, E. Linstead, M. Curcic, and P. Baldi. A fortran-keras deep learning bridge for scientific computing. *arXiv preprint arXiv:2004.10652*, 2020.

B. Pang, J. Yue, G. Zhao, Z. Xu, et al. Statistical downscaling of temperature with the random forest model. *Advances in Meteorology*, 2017, 2017.

J. Pathak, S. Subramanian, P. Z. Harrington, S. Raja, A. Chattopadhyay, M. Mardani, T. Kurth, D. Hall, Z.-Y. Li, K. Azizzadenesheli, P. Hassanzadeh, K. Kashinath, and A. Anandkumar. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *ArXiv*, abs/2202.11214, 2022.

A. Prasad, P. Harder, Q. Yang, P. Sattegeri, D. Szwarcman, C. Watson, and D. Rolnick. Evaluating the transferability potential of deep learning models for climate downscaling. *ICML Workshop Machine Learning for Earth System Modeling*, 2024.

A. Quiquet, D. M. Roche, C. Dumas, and D. Paillard. Online dynamical downscaling of temperature and precipitation within the *i*loveclim model (version 1.1). *Geoscientific Model Development*, 11(1):453–466, 2018. doi: 10.5194/gmd-11-453-2018.

E. Racah, C. Beckham, T. Maharaj, Prabhat, and C. J. Pal. Semi-supervised detection of extreme weather events in large climate datasets. *ArXiv*, abs/1612.02095, 2016.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378: 686–707, 2019.

A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *ArXiv*, abs/2204.06125, 2022.

N. Rampal, S. Hobeichi, P. B. Gibson, J. Baño-Medina, G. Abramowitz, T. Beucler, J. González-Abad, W. Chapman, P. Harder, and J. M. Gutiérrez. Enhancing regional climate downscaling through advances in machine learning. *Artificial Intelligence for the Earth Systems*, 2024. doi: 10.1175/AIES-D-23-0066.1.

S. Rasp, M. S. Pritchard, and P. Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences of the United States of America*, 115:9684 – 9689, 2018a.

S. Rasp, M. S. Pritchard, and P. Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, 2018b. ISSN 0027-8424. doi: 10.1073/pnas.1810286115.

M. Reichstein, G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, and Prabhat. Deep learning and process understanding for data-driven earth system science. *Nature*, 2(1), 2019. doi: https://doi.org/10.1038/s41586-019-0912-1.

N. M. Roberts and H. W. Lean. Scale-selective verification of rainfall accumulations from high-resolution forecasts of convective events. *Monthly Weather Review*, 136(1):78–97, 2008.

J. L. Rodgers and W. A. Nicewander. *Thirteen Ways to Look at the Correlation Coefficient*, volume 42. American Statistical Association, 1988.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *ArXiv*, abs/1505.04597, 2015.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

M. Salzmann, S. Ferrachat, C. Tully, S. Münch, D. Watson-Parris, D. Neubauer, C. Siegenthaler-Le Drian, S. Rast, B. Heinold, T. Crueger, R. Brokopf, J. Mülmenstädt, J. Quaas, H. Wan, K. Zhang, U. Lohmann, P. Stier, and I. Tegen. The global atmosphere-aerosol model icon-a-ham2.3–initial model evaluation and effects of radiation balance tuning on aerosol optical thickness. *Journal of Advances in Modeling Earth Systems*, 14(4):e2021MS002699, 2022. doi: https://doi.org/10.1029/2021MS002699. e2021MS002699 2021MS002699.

J. S. Schreck, C. Becker, D. J. Gagne, K. Lawrence, S. Wang, C. Mouchel-Vallon, J. Choi, and A. Hodzic. Neural network emulation of the formation of organic aerosols based on the explicit gecko-a chemistry model. *Journal of Advances in Modeling Earth Systems*, 14, 2022.

M. G. Schultz, C. Betancourt, B. Gong, F. Kleinert, M. Langguth, L. H. Leufen, A. Mozaffari, and S. Stadtler. Can deep learning beat numerical weather prediction? *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 379, 2021.

A. Serifi, T. Günther, and N. Ban. Spatio-temporal downscaling of climate data using convolutional and error-predicting neural networks. *Frontiers in Climate*, 3, 2021. ISSN 2624-9553. doi: 10.3389/fclim.2021.656479.

Y. Sha, D. J. Gagne, G. West, and R. Stull. Deep-learning-based gridded downscaling of surface meteorological variables in complex terrain. part i: Daily maximum and minimum 2-m temperature. *Journal of Applied Meteorology and Climatology*, 2020.

E. Sharifi, B. Saghafian, and R. Steinacker. Downscaling satellite precipitation estimates with multiple linear regression, artificial neural networks, and spline interpolation techniques. *Journal of Geophysical Research: Atmospheres*, 124:789 – 805, 2019.

H. Shi. Best-first decision tree learning, 2007.

X. Shi, Z. Chen, H. Wang, D. Y. Yeung, W.-K. Wong, and W. chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Neural Information Processing Systems*, 2015.

S. J. Silva, P.-L. Ma, J. C. Hardin, and D. Rothenberg. Physically regularized machine learning emulators of aerosol activation. *Geoscientific Model Development Discussions*, 2020:1–19, 2020a. doi: 10.5194/gmd-2020-393.

S. J. Silva, P. L. Ma, J. C. Hardin, and D. Rothenberg. Physically regularized machine learning emulators of aerosol activation. *Geoscientific Model Development*, 2020b.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

J. N. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *ArXiv*, abs/1503.03585, 2015.

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

K. Stengel, A. Glaws, D. Hettinger, and R. N. King. Adversarial super-resolution of climatological wind and solar data. *Proceedings of the National Academy of Sciences*, 117(29):16805–16815, 2020. doi: 10.1073/pnas.1918964117.

P. Stier, J. Feichter, S. Kinne, S. Kloster, E. Vignati, J. Wilson, L. N. Ganzeveld, I. Tegen, M. Werner, Y. Balkanski, M. Schulz, O. Boucher, A. Minikin, and A. Petzold. The aerosol-climate model echam5-ham. *Atmospheric Chemistry and Physics*, 5:1125–1156, 2004.

P. O. Sturm and A. S. Wexler. A mass- and energy-conserving framework for using machine learning to speed computations: a photochemistry example. *Geoscientific Model Development*, 2020.

P. O. Sturm and A. S. Wexler. Conservation laws in a neural network architecture: Enforcing the atom balance of a julia-based photochemical model (v0.2.0). *Geoscientific Model Development*, 2021.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

S. Twomey. Pollution and the planetary albedo. *Atmospheric Environment (1967)*, 8(12):1251–1256, 1974. ISSN 0004-6981. doi: https://doi.org/10.1016/0004-6981(74)90004-3.

T. Vandal, E. Kodra, S. Ganguly, A. Michaelis, R. Nemani, and A. R. Ganguly. Deepsd: Generating high resolution climate change projections through single image super-resolution. *Association for Computing Machinery*, page 1663–1672, 2017. doi: 10.1145/3097983.3098004.

J. Vicent, L. Martino, J. Verrelst, and G. Camps-Valls. Multifidelity gaussian process emulation for atmospheric radiative transfer models. *IEEE Transactions on Geoscience and Remote Sensing*, 61:1–10, 2023.

E. Vignati, J. Wilson, and P. Stier. M7: An efficient size-resolved aerosol microphysics module for large-scale aerosol transport models. *Journal of Geophysical Research: Atmospheres*, 109(D22), 2004. doi: https://doi.org/10.1029/2003JD004485.

Z. Y. Wan, R. Baptista, Y.-F. Chen, J. R. Anderson, A. Boral, F. Sha, and L. Zepeda-N'unez. Debias coarsely, sample conditionally: Statistical downscaling through optimal transport and probabilistic diffusion models. *ArXiv*, abs/2305.15618, 2023.

F. Wang and D. Tian. On deep learning-based bias correction and downscaling of multiple climate models simulations. *Climate Dynamics*, 59:3451 – 3468, 2022.

J. Wang, Z. Liu, I. Foster, W. Chang, R. Kettimuthu, and V. R. Kotamarthi. Fast and accurate learned multiresolution dynamical downscaling for precipitation. *Geoscientific Model Development*, 14(10):6355–6372, 2021a. doi: 10.5194/gmd-14-6355-2021.

J. Wang, Z. Liu, I. Foster, W. Chang, R. Kettimuthu, and V. R. Kotamarthi. Fast and accurate learned multiresolution dynamical downscaling for precipitation. *Geoscientific Model Development*, 14(10):6355–6372, 2021b. doi: 10.5194/gmd-14-6355-2021.

X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, C. C. Loy, Y. Qiao, and X. Tang. Esrgan: Enhanced super-resolution generative adversarial networks, 2018.

Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *Conference Record of the Thirty-Seventh*

Asilomar Conference on Signals, Systems and Computers, 2003, volume 2, pages 1398–1402 Vol.2. IEEE, 2003.

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

C. D. Watson, C. Wang, T. Lynar, and K. Weldemariam. Investigating two super-resolution methods for downscaling precipitation: ESRGAN and CAR, 2020.

D. Watson-Parris, Y. Rao, D. J. L. Oliviè, Ø. Seland, P. J. Nowack, G. Camps-Valls, P. Stier, S. Bouabid, M. Dewey, E. Fons, J. Gonzalez, P. Harder, K. Jeggle, J. Lenhardt, P. Manshausen, M. Novitasari, L. Ricard, and C. E. Roesch. Climatebench v1.0: A benchmark for data-driven climate projections. *Journal of Advances in Modeling Earth Systems*, 14, 2022.

O. Watt-Meyer, G. Dresdner, J. McGibbon, S. K. Clark, B. Henn, J. Duncan, N. Brenowitz, K. Kashinath, M. S. Pritchard, B. Bonev, M. E. Peters, and C. Bretherton. Ace: A fast, skillful learned global atmospheric model for climate prediction. *ArXiv*, abs/2310.02074, 2023.

J. Wider, J. Kruse, N. Weitzel, J. C. Bühler, U. Köthe, and K. Rehfeld. Towards learned emulation of interannual water isotopologue variations in general circulation models. *Environmental Data Science*, 2, 2023.

C. E. Winkler, P. Harder, and D. Rolnick. Climate variable downscaling with conditional normalizing flows. In *NeurIPS 2023 Workshop on Tackling Climate Change with Machine Learning*, 2023.

F. Yang, H. Yang, J. Fu, H. Lu, and B. Guo. Learning texture transformer network for image super-resolution. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5790–5799, 2020. doi: 10.1109/CVPR42600.2020.00583.

Q. Yang, P. Harder, V. Ramesh, A. Hernandez-Garcia, D. Szwarcman, P. Sattigeri, C. D. Watson, and D. Rolnick. Fourier neural operators for arbitrary resolution climate data downscaling. In *ICLR 2023 Workshop on Tackling Climate Change with Machine Learning*, 2023.

S. Yu, W. M. Hannah, L. Peng, M. A. Bhouri, R. Gupta, J. Lin, B. Lutjens, J. C. Will, T. Beucler, B. E. Harrop, B. R. Hillman, A. M. Jenney,

S. L. Ferretti, N. Liu, A. Anandkumar, N. Brenowitz, V. Eyring, P. Gentine, S. Mandt, J. Pathak, C. Vondrick, R. Yu, L. Zanna, R. P. Abernathey, F. Ahmed, D. C. Bader, P. Baldi, E. A. Barnes, G. Behrens, C. S. Bretherton, J. J. M. Busecke, P. M. Caldwell, W. K. Chuang, Y. Han, Y. Huang, F. Iglesias-Suarez, S. R. Jantre, K. Kashinath, M. Khairoutdinov, T. Kurth, N. J. Lutsko, P. Ma, G. Mooers, J. D. Neelin, D. A. Randall, S. Shamekh, A. Subramaniam, M. A. Taylor, N. M. Urban, J. Yuval, G. J. Zhang, T. Zheng, and M. S. Pritchard. Climsim: An open large-scale dataset for training high-resolution physics emulators in hybrid multi-scale climate simulators. *ArXiv*, abs/2306.08754, 2023.

J. Yuval, P. A. O'Gorman, and C. N. Hill. Use of neural networks for stable, accurate and physically consistent parameterization of subgrid atmospheric processes with good performance at reduced precision. *Geophysical Research Letters*, 48, 2020.

G. Zängl, D. Reinert, P. Rípodas, and M. A. Baldauf. The icon (icosahedral non-hydrostatic) modelling framework of dwd and mpi-m: Description of the non-hydrostatic dynamical core. *Quarterly Journal of the Royal Meteorological Society*, 141, 2015.

L. Zanna and T. Bolton. Data-driven equation discovery of ocean mesoscale closures. *Geophysical Research Letters*, 47(17):e2020GL088376, 2020. doi: https://doi.org/10.1029/2020GL088376.

L. Zanna and T. Bolton. *Deep Learning of Unresolved Turbulent Ocean Processes in Climate Models*, chapter 20, pages 298–306. John Wiley & Sons, Ltd, 2021. ISBN 9781119646181. doi: https://doi.org/10.1002/9781119646181.ch20.

R. Zeyde, M. Elad, and M. Protter. On single image scale-up using sparse-representations. In J.-D. Boissonnat, P. Chenin, A. Cohen, C. Gout, T. Lyche, M.-L. Mazure, and L. Schumaker, editors, *Curves and Surfaces*, pages 711–730, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

D. Zhang, F. Huang, S. Liu, X. Wang, and Z. Jin. Swinfir: Revisiting the swinir with fast fourier convolution and improved training for image super-resolution. *ArXiv*, abs/2208.11247, 2022.

Appendix A

Appendix Downscaling

A.1 Tuning Soft Mass-Constraining

Here, we investigate the influence of the factor α on the soft mass-constraining method in more detail. α is used as follows

$$\text{Loss} = (1 - \alpha) \cdot \text{MSE} + \alpha \cdot \text{Constraint violation}. \tag{A.1}$$

Table A.1 shows how the increase of α improves the mass conservation but only up to a value between 0.014 and 0.017. At the same time, it shows that the predictive skill decreases with the increase of α significantly.

Table A.1: Metrics calculated over 10,000 validation samples. The best scores are highlighted in bold blue, the second best in bold black.

Data	Alpha	RMSE ↓	MAE ↓	MS-SSIM ↑	Mass viol. ↓	#Neg per mil. ↓
W2	0.0001	**0.241**	**0.102**	99.95	0.021	1.21
W2	0.001	**0.237**	**0.100**	**99.96**	0.022	**0.12**
W2	0.01	0.247	0.103	**99.96**	0.022	1.39
W2	0.1	0.252	0.104	99.95	0.023	**0.41**
W2	0.9	0.268	0.110	99.95	0.020	16.83
W2	0.99	0.297	0.133	99.94	**0.014**	31.01
W2	0.999	0.477	0.261	99.84	**0.016**	600.96
W2	0.9999	0.706	0.433	99.71	0.017	3867.90
W2	1	2.618	1.814	94.22	0.017	960.42

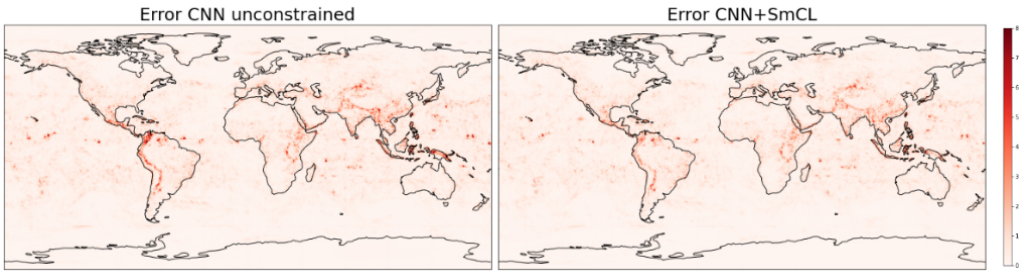

Figure A.1: The errors of the global predictions for unconstrained and constrained (SmCL) CNNs, when compared to the ground truth. The CNN is applied per 32x32 patch and then put together for a global predictions at a random time step. Used here is the TCW4 dataset. We can observe how the stronger errors in coastal and mountainous regions for the unconstrained predictions are dampened by soft-max constraining.

A.2 Additional Scores

We look at additional scores for our water content dataset. We investigate the mean bias (mean over the difference for each pixel value of prediction and truth), the peak signal-to-noise ratio (PSNR), the structural similarity

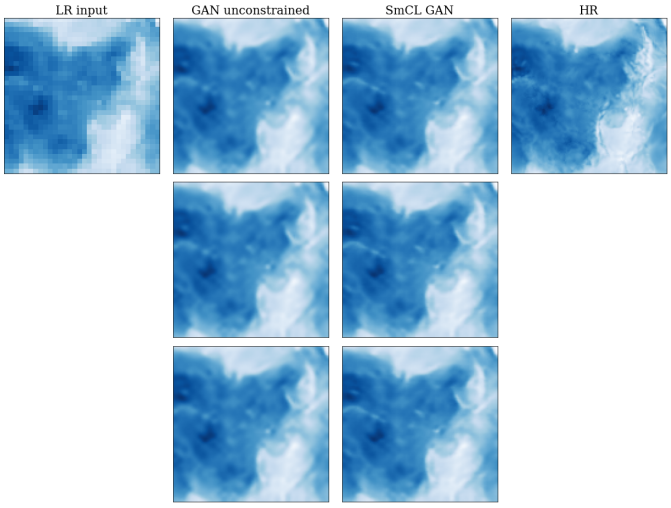

Figure A.2: A random sample for the GAN predictions, showing 3 different outputs from the ensemble, comparing constrained and unconstrained GANs.

index measure, the Pearson correlation (Corr), and the negative mean (the

average magnitude of predicted negative values, the average is calculated over all predicted values, including positive, that are set to zero to calculate the negative mean). These metrics show a similar trend then the metrics shown in Chapter 4: all of them are improved by adding constraints (soft m.) in our architecture. Without or with soft mass-constraining there are small biases appearing in the predictions, but hard constraining removes those biases. PSNR is a function of the MSE and therefore shows the same trend as it.

Table A.2: Metrics for different constraining methods applied to the SR CNN applied on the OOD water content dataset, calculated over 10,000 test samples. The mean is taken over 3 runs. The best scores are highlighted in bold blue, second best in bold black.

Data	Model Constraint	RMSE ↓	MAE ↓	MS-SSIM ↑	Mass viol. ↓	# Neg per mil. ↓
OOD	Enlarge	1.274	0.711	97.60	**0.000**	**0**
OOD	Bicubic	0.792	0.397	98.63	0.167	0.55
OOD	CNN	0.661	0.327	99.39	0.059	4.93
OOD	AddCL	**0.575**	**0.287**	**99.50**	**0.000**	1.65
OOD	MultCL	0.591	0.294	99.47	**0.000**	**0**
OOD	SmCL	**0.579**	**0.289**	**99.49**	**0.000**	**0**

SSIM and correlation give very similar results, AddCL and SmCL showing the best scores. Overall we can see that soft mass-constraining leads to the most significantly negative predictions, which would cause issues in the context of climate models and predictions.

In Table A.4 we report the Fractional Skills Score (FSS) for window sizes 2,4 and 8 for 95th and 99th percentiles, observing a slight improvement with enforcing constraints.

Table A.3: More metrics for different constraining methods applied to an SR CNN, calculated over 10,000 test samples. The best scores are highlighted in bold blue, second best in bold.

Data	Model Constraint	Mean bias ↓	PSNR ↑	SSIM ↑	Corr ↑	Neg mean ↓
W2	Enlarge	**0.000**	45.36	98.65	99.75	**0.000**
W2	Bicubic	**0.000**	51.46	99.71	99.95	**0.000**
W2	CNN	−0.003	53.62	99.82	99.97	0.002
W2	soft m.	−0.002	52.07	99.74	99.94	0.192
W2	AddCL	**0.000**	**54.91**	**99.85**	**99.98**	0.002
W2	MultCL	**0.000**	54.65	99.84	99.97	**0.000**
W2	SmCL	**0.000**	**54.95**	**99.85**	**99.98**	**0.000**
W4	Enlarge	**0.000**	39.43	94.91	98.98	**0.000**
W4	Bicubic	**0.000**	43.55	98.29	99.63	0.000
W4	CNN	−0.015	45.26	98.70	99.74	0.001
W4	soft m.	−0.001	43.55	98.15	99.59	0.546
W4	AddCL	**0.000**	**46.35**	**98.89**	**99.80**	0.001
W4	MultCL	**0.000**	45.98	98.83	99.78	**0.000**
W4	SmCL	**0.000**	**46.31**	**98.88**	**99.79**	**0.000**
W8	Enlarge	**0.000**	34.84	89.08	96.95	**0.000**
W8	Bicubic	+0.0001	37.77	95.40	98.50	0.006
W8	CNN	−0.0148	38.96	95.93	98.82	0.012
W8	soft m.	−0.0071	37.32	94.37	98.22	0.656
W8	AddCL	**0.000**	**39.56**	**96.23**	**98.96**	0.011
W8	MultCL	**0.000**	39.13	95.99	98.87	**0.000**
W8	SmCL	**0.000**	**39.55**	**96.21**	**98.96**	**0.000**
W16	Enlarge	**0.000**	30.92	85.20	92.19	**0.000**
W16	Bicubic	+0.0090	32.91	91.99	95.15	0.063
W16	CNN	−0.0091	33.83	92.48	95.94	0.006
W16	soft m.	+0.0115	32.70	90.45	94.63	4.233
W16	AddCL	**0.000**	**34.14**	**92.67**	**96.20**	0.581
W16	MultCL	**0.000**	33.98	92.54	96.07	**0.000**
W16	SmCL	**0.000**	**34.13**	**92.68**	**96.19**	**0.000**

Table A.4: Fractional Skill Score (FSS, higher is better) for different constraining methods and SR CNN applied on the ERA5 water content data, calculated over 10,000 test samples. We look at window sizes 2, 4 and 8 and the 95th and 99th percentiles. The best scores are highlighted in bold blue, second best in bold black.

Data	Model	95perc.			99perc.		
	Constraint	2	4	8	2	4	8
W4	Enlarge	0.970	0.989	0.997	0.935	0.974	0.991
W4	Bicubic	0.971	0.987	0.994	0.935	0.969	0.986
W4	CNN	0.978	0.992	0.997	0.950	0.979	0.993
W4	soft m.	0.971	0.989	0.997	0.935	0.974	0.991
W4	AddCL	**0.981**	**0.993**	**0.998**	**0.956**	**0.983**	**0.994**
W4	MultCL	0.979	0.992	0.998	0.951	0.980	0.993
W4	SmCL	**0.981**	**0.993**	**0.998**	**0.955**	**0.983**	**0.994**

Table A.5: The variance among super-pixels for different constraining methods and SR CNN applied on the ERA5 water content data, calculated over 10,000 test samples.

Data	Model	Constraint	Variance
W4	Enlarge	none	0.00
W4	Bicubic	none	0.85
W4	CNN	none	1.22
W4	CNN	soft m.	0.96
W4	CNN	AddCL	1.32
W4	CNN	MultCL	1.24
W4	CNN	SmCL	1.34
W4	HR	none	1.65

Table A.6: Metrics for different constraining methods applied to an SR CNN, calculated over 10,000 test samples of the water content data. The mean is taken over 3 runs. The best scores are highlighted in bold blue, second best in bold.

Data	Model Constraint	RMSE ↓	MAE ↓	MS-SSIM ↑	Mass viol. ↓	#Neg per mil.↓
W2	Enlarge	0.422	0.361	99.61	**0.000**	**0**
W2	Bicubic	0.322	0.137	99.90	0.066	0.25
W2	CNN	0.251	0.105	99.95	0.026	1.40
W2	soft m.	0.301	0.137	99.23	0.016	104.65
W2	AddCL	**0.216**	**0.092**	**99.96**	**0.000**	1.31
W2	MultCL	0.223	**0.094**	**99.96**	**0.000**	**0**
W2	SmCL	**0.215**	**0.094**	**99.96**	**0.000**	**0**
W4	Enlarge	1.286	0.717	97.60	**0.000**	**0**
W4	Bicubic	0.800	0.401	99.12	0.169	0.53
W4	CNN	0.657	0.326	99.40	0.058	2.41
W4	soft m.	0.801	0.410	99.15	0.023	581.54
W4	AddCL	**0.580**	**0.290**	**99.50**	**0.000**	1.42
W4	MultCL	0.606	0.300	99.47	**0.000**	**0**
W4	SmCL	**0.582**	**0.291**	**99.49**	**0.000**	**0**
W8	Enlarge	2.181	1.294	92.39	**0.000**	**0**
W8	Bicubic	1.557	0.900	96.49	0.318	6.56
W8	CNN	1.358	0.782	97.15	0.109	15.48
W8	soft m.	1.640	0.965	96.06	0.029	103,702
W8	AddCL	**1.267**	**0.733**	**97.41**	**0.000**	632.32
W8	MultCL	1.331	**0.733**	97.22	**0.000**	0.10
W8	SmCL	**1.268**	0.734	**97.40**	**0.000**	**0**
W16	Enlarge	3.425	2.159	85.55	**0.000**	**0**
W16	Bicubic	2.723	1.730	91.72	0.510	53.67
W16	CNN	2.450	1.545	92.68	0.203	4.15
W16	soft m.	2.794	1.776	90.74	0.036	2250.77
W16	AddCL	**2.364**	**1.491**	**92.96**	**0.000**	457.34
W16	MultCL	2.409	1.518	92.77	**0.000**	0.17
W16	SmCL	**2.368**	**1.492**	**92.95**	**0.000**	**0**

Table A.7: Metrics for different constraining methods applied to the SR CNN, calculated over the test set for water vapor, liquid water, and temperature. The mean is taken over 3 runs. For Q_L, RMSE, MAE, and Constr. violation are scaled by a factor of 10^3 for readability. The best scores are highlighted in bold blue, second best in bold black.

Data	Var.	Model Constr.	RMSE ↓	MAE ↓	MS-SSIM ↑	Constr. viol. ↓
MEn	Q_v	Enlarge	0.474	0.262	94.74	**0.000**
MEn	Q_v	Bicubic	0.326	0.182	97.12	0.07
MEn	Q_v	CNN	0.260	0.141	98.14	0.02
MEn	Q_v	AddCL	**0.250**	**0.133**	**98.28**	**0.00**
MEn	Q_v	MultCL	**0.250**	**0.133**	**98.28**	**0.00**
MEn	Q_v	SmCL	**0.248**	**0.132**	**98.30**	**0.00**
MEn	Q_L	Enlarge	0.0217	0.00862	98.34	**0.00000**
MEn	Q_L	Bicubic	0.0186	0.00765	98.96	0.00236
MEn	Q_L	CNN	0.0157	0.00617	99.15	0.00067
MEn	Q_L	AddCL	**0.0155**	**0.00588**	**99.18**	**0.00000**
MEn	Q_L	MultCL	0.0166	0.00647	99.06	**0.00000**
MEn	Q_L	SmCL	**0.0155**	0.00585	**99.17**	**0.00000**
MEn	T	Enlarge	0.470	0.288	99.03	**0.0**
MEn	T	Bicubic	0.281	0.156	99.67	159.1
MEn	T	CNN	0.459	0.287	99.03	139.7
MEn	T	AddCL	0.276	0.160	99.67	**0.0**
MEn	T	MultCL	**0.270**	**0.155**	**99.69**	**0.0**
MEn	T	SmCL	**0.272**	**0.155**	**99.68**	**0.0**

Appendix B

Appendix Aerosol Emulation

B.1 Aerosol Boxmodel

The aerosol box model is a tool we built to easily generate input-output pairs for emulator development. It extracts the M7 module from the ECHAM-HAM model, so it can quickly run without the rest of the climate model. This can be used in the future for active learning approaches.

B.2 Modeled Variables

Table B.2 shows all the variables used for our emulator. Overall we consider 39 quantities; the first eleven shown in the table are only input variables and are not changed by M7/our emulator. The masses and concentrations of different aerosol species are both input and output for the model. Additionally, we output the water content of different aerosol modes.

B.3 ICON Setup Details

The AMIP run is based on the non-hydrostatic atmosphere and ECHAM physics. It is initialized from the Integrated Forecast System (IFS) analysis files and uses transient boundary conditions for sea surface temperature (SST) and sea ice, spectral solar irradiation, well-mixed greenhouse gases CO2, CH4, N2O, CFCs, O3 concentration, optical properties of tropospheric aerosols and optical properties of stratospheric volcanic aerosols. ICON-A uses a vertical hybrid sigma height coordinate with 47 levels. The emission scenario used is the Representative Concentration Pathway (RCP) 4.5.

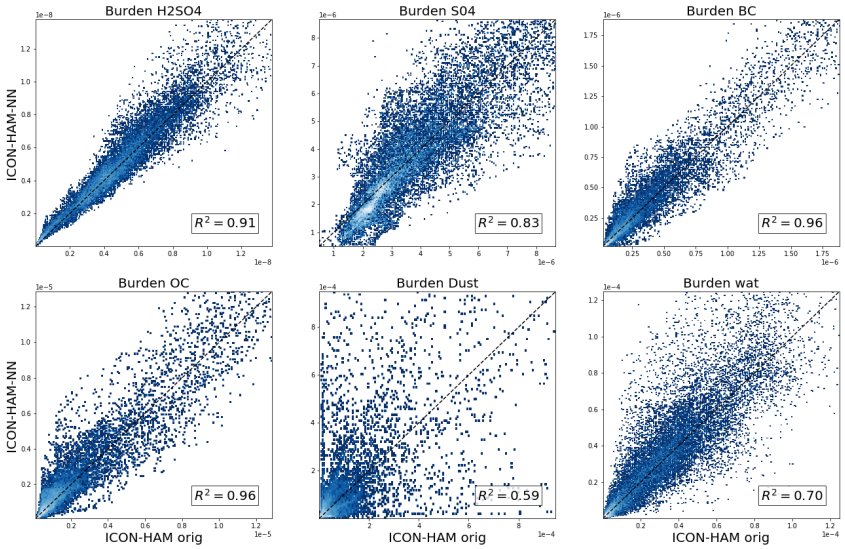

Figure B.1: Aerosol burdens after 1 year run of ICON-HAM with our correction-constrained emulator compared to a reference run including the M7 original.

B.4 Data Subsampling

The objective for the dataset is to have enough and very diverse data, that covers many different cases, so application to new samples does not go too far out of the training distribution. We also want the dataset to still have a manageable size, around a couple of gigabyte to enable fast and flexible experiments.

For both offline and online cases we run ICON as described for 5 years, from 2000-2004. Here we then subsample across times. We sample from the first day of each month, alternating from midnight to noon UTC. We always use all pressure levels.

Whereas the temporal subsampling is the same for both cases, the spatial subsampling then differs for offline and online training datasets. For practical reasons, the spatial subsampling is done across processes. For parallelisation, the global gridboxes are divided across processes, in our case 1024 processes for 8 cores with 128 processes each. The offline dataset then consists of every fifth process's data, taking every process with a number that ends with "0"

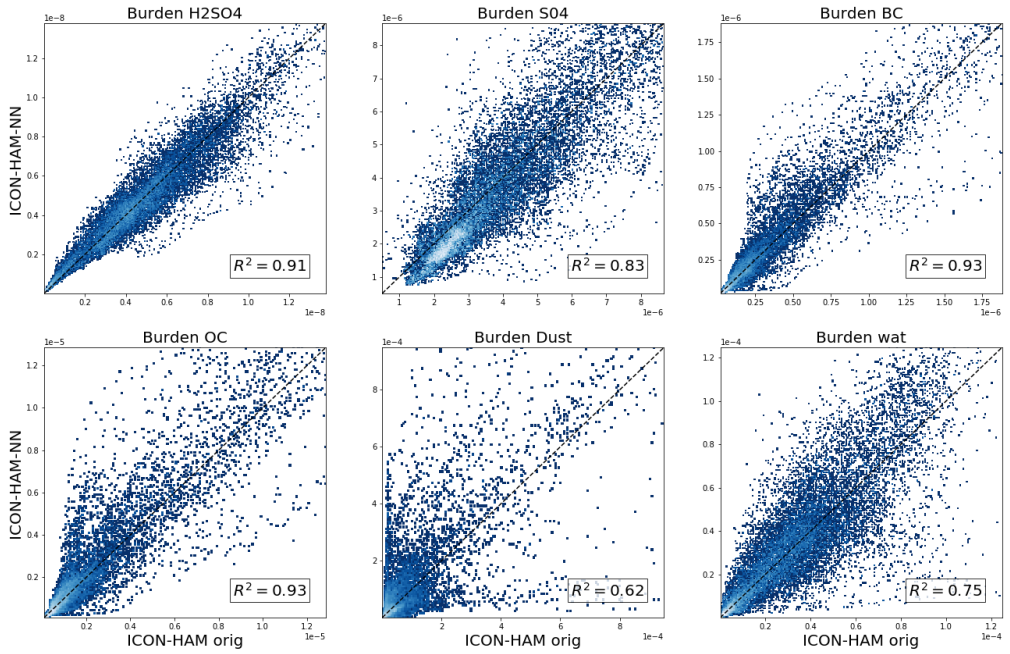

Figure B.2: Aerosol burdens after 1 year run of ICON-HAM with our soft mass-constrained plus CorL emulator compared to a reference run including the M7 original.

or "5". The online dataset includes one additional process's data per ten processes, i.e. includes data from 3/10 of the processes/locations.

We also employed or emulator on data from all locations, i.e. from all processes, but we observed a worsening of the online performance here. This might be because if a more complex or bigger dataset it might be necessary to adjust the neural network size as well. Keeping our 2-layer and 256 nodes architecture, a smaller dataset is more beneficial.

This subsampling scheme could be improved in future work, optimizing it for more diversity with fewer data samples and not depending on different processes.

B.5 ICON Crashes

Here, we explain what happens when our simulation "crashes". With 'crash,' we refer to the case when an error in the simulation occurs that is severe enough to stop the simulation. We only observe crashes, when our NN em-

ulator is included, not in the original M7 versions. This gives a strong indication that the crashes are caused by our emulator. The errors that appear, never appear inside our NN code part, but in code parts that follow later.

We observe two kinds of errors:

1. A lookup table overflow error: "FATAL ERROR in cuini: lookup table overflow".

2. A lookup table overflow error: "FATAL ERROR in cuadjtq (1): lookup table overflow"

The errors occur both due to a look-up table overflow in the convection scheme. These errors occur possibly due to unrealistic temperature or humidity fields, that are caused by unphysical predictions of the NN, when encountering out-of-distribution samples.

We find that the first error appears when we do not strictly ensure the positivity of aerosol numbers and masses. The second error appears when we do not ensure mass conservation. It is important to notice that our constraining methods applied in the aerosol case only enforce either positivity or conservation of mass and not both at the same time. This is why the instabilities can still occur.

We are able to find three different setups where no crashes are observed: including the correction layer Corl, which guarantees positive masses and numbers, including a soft mass constraint, which enables to model to run stably. So far, we have only tested the stability of a one-year run, but it is likely that a multi-year run would also remain stable.

Variable	Unit	Input	Output
Pressure	Pa	√	
Temperature	K	√	
Rel. Humidity	-	√	
Geop. height	m	√	
layer thick.	m	√	
grid vol.	m^{-3}	√	
Ionisation Rate	-	√	
cloud cover	-	√	
Boundary layer	-	√	
Forest fraction	-	√	
H2SO4 prod. rate	$cm^{-3}s^{-1}$	√	
H2SO4 mass	$\mu g\ m^{-3}$	√	√
SO4 ns mass	$molec.\ m^{-3}$	√	√
SO4 ks mass	$molec.\ m^{-3}$	√	√
SO4 as mass	$molec.\ m^{-3}$	√	√
SO4 cs mass	$molec.\ m^{-3}$	√	√
bc ks mass	$\mu g\ m^{-3}$	√	√
bc as mass	$\mu g\ m^{-3}$	√	√
bc cs mass	$\mu g\ m^{-3}$	√	√
bc ki mass	$\mu g\ m^{-3}$	√	√
oc ks mass	$\mu g\ m^{-3}$	√	√
oc as mass	$\mu g\ m^{-3}$	√	√
oc cs mass	$\mu g\ m^{-3}$	√	√
oc ki mass	$\mu g\ m^{-3}$	√	√
du as mass	$\mu g\ m^{-3}$	√	√
du cs mass	$\mu g\ m^{-3}$	√	√
du ai mass	$\mu g\ m^{-3}$	√	√
du ci mass	$\mu g\ m^{-3}$	√	√
ns concentration	cm^{-3}	√	√
ks concentration	cm^{-3}	√	√
as concentration	cm^{-3}	√	√
cs concentration	cm^{-3}	√	√
ki concentration	cm^{-3}	√	√
ai concentration	cm^{-3}	√	√
ci concentration	cm^{-3}	√	√
ns water	$kg\ m^{-3}$		√
ks water	$kg\ m^{-3}$		√
as water	$kg\ m^{-3}$		√
cs water	$kg\ m^{-3}$		√

Table B.1: R^2 scores (higher is better) on the test set for each variable for the **offline case**, for our base NN, soft mass constrained, correction constrained and completion constrained variants. Best scores are highlighted in bold blue.

	SO4					Black Carbon			
Variable	H2SO4	NS	KS	AS	CS	KS	AS	CS	CI
base	**1.00**	**0.87**	0.95	**0.95**	0.28	**0.67**	**0.98**	**-0.03**	**0.97**
soft m.	**1.00**	0.76	**0.96**	**0.95**	**0.30**	0.52	0.97	**-0.03**	**0.97**
CorL	**1.00**	0.79	0.95	**0.95**	0.28	0.51	0.97	-0.07	**0.97**
CompL	**1.00**	0.86	0.87	**0.95**	0.24	0.44	0.97	-0.22	**0.97**

Table B.2: Same as the table above for additional variables.

	Organic Carbon				Dust			
Variable	KS	AS	CS	CI	AS	CS	AI	CI
base	**1.00**	0.99	0.77	0.97	**0.95**	**0.97**	0.95	**0.97**
soft m.	**1.00**	**1.00**	**0.85**	**0.98**	0.93	0.88	0.95	0.88
CorL	**1.00**	0.99	0.83	0.97	**0.95**	0.96	**0.96**	0.96
CompL	**1.00**	0.99	0.88	**0.98**	0.51	0.92	0.86	0.93

Table B.3: Same as table above for additional variables.

	Number particles							Water content			
Variable	NS	KS	AS	CS	KI	AI	CI	NS	KS	AS	CS
base	0.77	**0.96**	**0.98**	0.94	**0.98**	0.95	0.94	**0.98**	0.98	0.88	**0.85**
soft m.	0.77	**0.96**	0.97	**0.97**	0.97	0.95	**0.97**	**0.98**	0.98	0.88	0.83
CorL	**0.80**	0.93	0.97	0.95	0.97	**0.96**	0.96	**0.98**	**0.99**	**0.91**	0.84
CompL	0.72	0.95	0.96	0.89	0.97	0.86	0.88	**0.98**	0.97	0.89	0.78

Table B.4: R^2 scores (higher is better) for tracer values after a one-year run, comparing ICON-HAM-NeuralM7 and original ICON-HAM runs. We consider all three stable versions of NeuralM7, the soft mass-constrained one (soft m.), the offline correction one (corr. offl.) and the model constrained with a combination (sm+corr). Best scores are highlighted in bold blue.

	SO4					Black Carbon			
Variable	H2SO4	NS	KS	AS	CS	KS	AS	CS	CI
soft m.	0.86	0.21	0.06	0.75	0.83	0.85	0.88	-0.33	0.94
corr. offl.	**0.89**	0.75	0.25	0.80	**0.85**	**0.88**	**0.92**	**0.48**	0.94
sm+corr	0.86	**0.80**	**0.37**	**0.80**	0.83	0.87	0.89	0.21	**0.95**

Table B.5: Same as table above for additional variables.

	Organic Carbon				Dust			
Variable	KS	AS	CS	CI	AS	CS	AI	CI
soft m.	0.91	0.91	**0.38**	**0.95**	0.86	0.89	**0.85**	**0.87**
corr offl.	**0.92**	**0.94**	-3.15	**0.95**	**0.89**	**0.91**	0.10	0.24
sm + corr.	**0.92**	0.91	-2.26	**0.95**	**0.89**	0.89	0.15	0.33

Table B.6: Same as table above for additional variables.

	Number particles							Water content			
Variable	NS	KS	AS	CS	KI	AI	CI	NS	KS	AS	CS
soft m.	0.72	-1.68	0.84	0.86	**0.96**	**0.84**	**0.87**	0.09	0.67	0.25	0.66
corr offl.	0.88	**-1.19**	**0.86**	**0.91**	**0.96**	0.10	0.31	0.61	**0.75**	**0.41**	0.71
sm + corr.	**0.91**	-1.75	0.84	0.89	**0.96**	0.16	0.36	**0.70**	0.73	0.37	**0.76**

Appendix C

Publication List

Peer-reviewed publications

Physics-Constrained Deep Learning for Climate Modeling

1. P. Harder, A. Hernandez-Garcia, V. Ramesh, Q. Yang, P. Sattegeri, D. Szwarcman, C. Watson, and D. Rolnick. Hard-constrained deep learning for climate downscaling. *Journal of Machine Learning Research*, 24(365):1–40, 2023a

2. P. Harder, Q. Yang, V. Ramesh, P. Sattigeri, A. Hernandez-Garcia, C. D. Watson, D. Szwarcman, and D. Rolnick. Generating physically-consistent high-resolution climate data with hard-constrained neural networks. In *NeurIPS 2022 Workshop on Tackling Climate Change with Machine Learning*, 2022b

3. P. Harder, D. Watson-Parris, P. Stier, D. Strassel, N. R. Gauger, and J. Keuper. Physics-informed learning of aerosol microphysics. *Environmental Data Science*, 1:e20, 2022a. doi: 10.1017/eds.2022.22

4. J. González-Abad, Á. Hernández-García, P. Harder, D. Rolnick, and J. M. Gutiérrez. Multi-variable hard physical constraints for climate model downscaling. In *Proceedings of the AAAI Symposium Series*, volume 2, pages 62–67, 2023

Downscaling/Super-Resolution

5. Q. Yang, P. Harder, V. Ramesh, A. Hernandez-Garcia, D. Szwarcman, P. Sattigeri, C. D. Watson, and D. Rolnick. Fourier neural operators for

arbitrary resolution climate data downscaling. In *ICLR 2023 Workshop on Tackling Climate Change with Machine Learning*, 2023

6. J. Delgado-Centeno, P. Harder, V. Bickel, B. Moseley, F. Kalaitzis, S. Ganju, and M. Olivares-Mendez. Superresolution of lunar satellite images for enhanced robotic traverse planning: Maximizing the value of existing data products for space robotics. *IEEE Robotics & Automation Magazine*, pages 2–14, 2023. doi: 10.1109/MRA.2023.3276267

7. J. Delgano-Centeno, P. Harder, B. Moseley, V. Bickel, S. Ganju, F. Kalaitzis, and M. Olivares-Mendez. Single image super-resolution with uncertainty estimation for lunar satellite images. *NeurIPS Workshop ML for Physical Sciences*, 2021

8. C. E. Winkler, P. Harder, and D. Rolnick. Climate variable downscaling with conditional normalizing flows. In *NeurIPS 2023 Workshop on Tackling Climate Change with Machine Learning*, 2023

9. J. Millar, P. Harder, L. Freischem, P. Weiss, and P. Stier. Towards downscaling global aod with machine learning. In *ICLR 2024 Workshop on Tackling Climate Change with Machine Learning*, 2024

10. M. Langguth, P. Harder, I. Schicker, A. Patnala, S. Lehner, K. Mayer, and M. Dabernig. A benchmark dataset for meteorological downscaling. In *ICLR 2024 Workshop on Tackling Climate Change with Machine Learning*, 2024

11. N. Rampal, S. Hobeichi, P. B. Gibson, J. Baño-Medina, G. Abramowitz, T. Beucler, J. González-Abad, W. Chapman, P. Harder, and J. M. Gutiérrez. Enhancing regional climate downscaling through advances in machine learning. *Artificial Intelligence for the Earth Systems*, 2024. doi: 10.1175/AIES-D-23-0066.1

12. A. Prasad, P. Harder, Q. Yang, P. Sattegeri, D. Szwarcman, C. Watson, and D. Rolnick. Evaluating the transferability potential of deep learning models for climate downscaling. *ICML Workshop Machine Learning for Earth System Modeling*, 2024

13. B. Fallah, C. Menz, E. Russo, P. Harder, P. Hoffmann, I. Didovets, and F. F. Hattermann. Climate model downscaling in central asia: a dynamical and a neural network approach. *Geoscientific Model Development Discussions*, 2023:1–27, 2023

DL for Emulation

14. P. Harder, D. Watson-Parris, D. Strassel, N. Gauger, P. Stier, and J. Keuper. Emulating aerosol microphysics with a machine learning. In *ICML 2021 Workshop on Tackling Climate Change with Machine Learning*, 2021

15. D. Watson-Parris, Y. Rao, D. J. L. Oliviè, Ø. Seland, P. J. Nowack, G. Camps-Valls, P. Stier, S. Bouabid, M. Dewey, E. Fons, J. Gonzalez, P. Harder, K. Jeggle, J. Lenhardt, P. Manshausen, M. Novitasari, L. Ricard, and C. E. Roesch. Climatebench v1.0: A benchmark for data-driven climate projections. *Journal of Advances in Modeling Earth Systems*, 14, 2022

Appendix D

CV

Paula Harder

Education

University of Kaiserslautern, DE	*July 2020 - Nov 2024*
Ph.D. Computer Science	
University of Oxford, Oxford, UK	*Oct 2021 - Dec 2021*
Student visitor	
University of Tübingen, Tübingen, DE	*Oct 2017 - Sep 2019*
M.S. Mathematics	
University of Tübingen, Tübingen, DE	*Oct 2014 - Sep 2017*
B.S. Mathematics	
Waldorf School, Weimar, DE	*Sep 2001 - Jul 2014*
High School Diploma (Abitur)	

Research/Work Experience

Fraunhofer ITWM, Kaiserslautern	*Jul 2020 - Dec. 2024*
Researcher Deep Learning Group	
Allen Institute AI2, Seattle, USA	*Jun 2023 - Sep 2023*
Internship Climate Modeling Team	
Mila Quebec AI Institute, Montreal, Canada	*Oct 2022 - Dec 2022*
Research Intern	*Jan 2022 - May 2022*
University of Oxford, Oxford, UK	*May 2022 - Oct 2022*
Visiting Researcher Climate Process Team	

Frontier Development Lab ESA, remote Team Lead	*Jun 2022 - Aug 2022*
Frontier Development Lab NASA, remote Machine Learning Scientist	*Jun 2021 - Aug 2021*
TWT Science and Innovation, Stuttgart Junior Development Engineer	*Nov 2019 - May 2020*
University of Tübingen, Tübingen, DE Teaching Assistant, Numerical Analysis	*Oct 2018 - Mar 2019*
DigSILENT, Gomaringen, DE Research Intern	*Jul 2018 - Oct 2018*
DKRZ, Hamburg, DE Student Research Assistant	*Feb 2018 - Apr 2018*

Awards and Scholarships

Travel Award, Aerosol Conference IAMA	*2023*
E-fellows scholarship	*2022*
Poster Award, Climate Informatics Conference	*2022*
Winner 3rd NOAA AI Workshop Hackathon, Climate Emulation	*2021*
Winner Climate Crisis AI Hackathon, AI Artist Challenge	*2021*
Winner AI for Climate Hackathon, Forest Fire Challenge	*2021*
Fraunhofer Doctoral Scholarship	*2020*
Winner AI Chess Competition	*2019*
Germany Scholarship (Deutschlandstipendium)	*2018*
Germany Scholarship (Deutschlandstipendium)	*2015*