Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for naturvitenskap
Institutt for fysikk

Jacob Wulff Wold

GAN Based Super-Resolution of Near-Surface 3D Atmospheric Wind Flow With Physics Informed Loss Function

Masteroppgave i Fysikk og matematikk
Veileder: Jon Andreas Støvneng
Medveileder: Adil Rasheed og Jan-Tore Horn
Juli 2023

NTNU
Kunnskap for en bedre verden

Jacob Wulff Wold

GAN Based Super-Resolution of Near-Surface 3D Atmospheric Wind Flow With Physics Informed Loss Function

Masteroppgave i Fysikk og matematikk
Veileder: Jon Andreas Støvneng
Medveileder: Adil Rasheed og Jan-Tore Horn
Juli 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for naturvitenskap
Institutt for fysikk

NTNU
Kunnskap for en bedre verden

Abstract

To meet the growing demand for green power generation, wind power development is projected to increase much in the coming years. Computationally cheap high resolution wind data can help wind developers increase production and give better production estimates, thereby reducing both the cost of wind power and the risk associated with wind farm development.

To help accomplish this, here a fully convolutional 3D GAN with a wind gradient based loss function is trained to super-resolve microscale near-surface atmospheric wind flow. Applying an ESRGAN generator architecture with a terrain feature extractor and a feature dropout layer, the model super-resolves the wind field to 4x4 increased horizontal resolution with average length of the error wind vector 0.24 m/s, and physically reasonable-looking 3D wind flow.

The model is applied to a generated dataset with unevenly spaced vertical coordinates. Various approaches for addressing the irregular grid are tested, and it is found that interpolating the wind field with regular vertical spacing relative to the ground, and adding terrain as input to the model performs best.

Based on the physics of atmospheric wind flow, multiple loss functions comparing specific parts of the wind gradient of the generated and the high resolution wind field are introduced and tested. A loss function focusing on, in descending order, differences in the $\frac{\partial}{\partial x}\frac{\partial}{\partial y}$ derivatives, horizontal divergence $\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}$ and divergence, mediated by an average absolute error pixel loss was found to increase performance greatly. Adversarial learning is dropped, as it at best only slightly improved on pure content loss.

The best model is shown to do significantly better than interpolation, even approximating turbulent wind patterns reasonably well. The model is shown to be flexible, also able to produce 8x8 and 16x16 resolution increase. Further work is needed to apply the model in wind farm development or operation, tailoring the model to the relevant scale, terrain and datasets. All in all, the model should be considered a proof-of-concept for CNN driven super-resolution of 3D microscale atmospheric wind flow.

Sammendrag

I møte med økende behov for grønn energi er det froventet med stor økning i vindkraftutbygging de neste årene. Billig høyoppløslig vinddata kan hjelpe vindutviklere å optimere produksjon og redusere risiko ved vindutbygging.

For å bidra til dette, brukes en konvulær GAN modell her til å øke oppløsningen på mikroskala atmosfæriske vindfelt. Ved å anvende en ESRGAN generator-arkitektur med terrenganalyse og dropout-lag gir modellen en 4x4 økning i horisontal oppløsning på vinddata med 0.24 m/s gjennomsnittlig vektorlengde på feilen, og fysisk rimelige genererte 3D vindfelt.

Modellen brukes på et numerisk generert datasett med irregulære vertikale koordinater. Forskjellige tilnærminger for å håndtere dette koordinatsystemet blir testet. Det best fungerende oppsettet interpolerer dataen så den blir regulær i forhold til bakkenivå, og inkluderer z-koordinater som input til modellen.

Basert på relevant fysikk introduseres og testes flere tapfunksjoner som sammenligner spesifikke deler av vindgradienten til det genererte og høyoppløselige vindfeiltet. Den beste tapsfunksjonen fokuserer mest på, i synkende rekkefølge, forskjeller i $\frac{\partial}{\partial x}\frac{\partial}{\partial y}$-deriverte, horisontal divergens $\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}$, absoluttfeil og divergens. Adversarial læring blir til slutt droppet, da det maks bidro til en liten forbedring.

Den beste modellen presterer betydelig bedre enn interpolering, og anslår til og med turbulente vindmønstre rimelig bra. Modellen er også fleksibel, den kan også brukes til 8x8 og 16x16 økning i horisontal oppløsning. Mer må gjøres før modellen kan anvendes i utvikling eller drift av vindparker, da modellen må tilpasses relevant skala, terreng og datasett. Totalt sett er resultatene lovende.

Preface

Venturing from my project in philosophy of physics to work with something as applied as this Master's thesis was a transition. It was refreshing to get grounded with some very applied machine learning and coding from trying to get an overview of physics as a whole. However, the area was also new to me, so there was a lot to learn and not always anyone to ask for help. Working with 3D data and a generative adversarial also brings a lot of complications to the data processing, model design, training and plotting. So it has been frustrating, but I've learned a lot. A nice side of working with 3D data is that when you actually mange to generate and plot wind field you can actually look at the results in 3D, which is very rewarding.

I want to thank

- Adil Rasheed for helping me get started and giving feedback when the deadline was approaching.
- Mandar Tabib from SINTEF for providing some useful advice and and links
- Thomas Nakken Larsen for providing his code and thoughts about the project
- Jon Andreas Støvneng for stepping in when needed
- Jan-Tore Horn from Vind Technologies for being a valuable connection to the real world, and for pleasant meetings throughout the semester
- Most of all, my friend Lars Bentsen, who has been a great help, and always been available for discussion or helping me understand the machine learning frameworks.

Jan-Tore Horn has co-supervised this thesis, and Vind Technologies can utilise the results of the work. Lastly, this thesis has been prepared as part of NorthWind (Norwegian Research Centre on Wind Energy) co-financed by the Research Council of Norway, industry and research partners. Read more at www.northwindresearch.no.

Table of Contents

List of Tables

List of Figures

Abbreviations

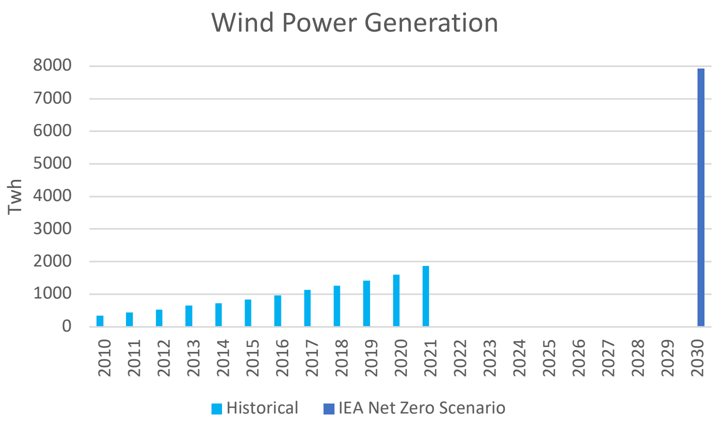
List of all abbreviations:

- **ASHA** Asynchronous Successive Halving Algorithm
- **CFD** Computational Fluid Dynamics
- **GAN** Generative Adversial Network
- **IEA** The International Energy Agency
- **HARMONIE** Hirlam Aladin Regional Mesoscale Operational Numerical prediction in Europe (CFD model)
- **HR** High resolution
- **LCOWE** Levelised Cost of Wind Energy
- **LR** Low resolution
- **PSNR** Peak signal to noise ratio
- **RANS** Reynolds Averaged Navier-Stokes Stokes Equations (CFD model)
- **RDB** Residual Dense Block
- **RRDB** Residual in Residual Dense Block
- **SIMRA** Semi Implicit Method for Reynolds Averaged Navier
- **SR** Super-resolution

Chapter 1

Introduction

Wind energy is an increasingly important power source. The International Energy Agency (IEA) estimates that wind power generation increased by a record 273 TWh [1] (up 17 percent) in 2021. Yet their 2050 net zero carbon emission scenario has 7900 TWh of wind electricity generation in 2030, and to reach this we will need to more than double the annual wind power additions relative to the record production increase in 2021.



Source: IEA, [1] Licence: CC BY 4.0

Figure 1.1: Yearly wind power generation, and estimated needed production in 2030 in IEA's Net Zero Emissions by 2050 Scenario

Further increases in wind power depend on reductions in the levelised cost of wind energy (LCOWE), effective cost per unit electricity from wind power. One way to achieve this is to optimise the turbine layout with respect to the wind field. The ability to optimise the layout depends on the quality of accessible wind flow data, and our ability to estimate how placing turbines would influence the wind flow described by that data. Better layout optimisation would help reduce LCOWE and give immediate production gains for new projects by allowing wind developers to squeeze more power generation out of each turbine. And, perhaps more important, better production estimates reduce the uncertainty, and thereby risk of developing a wind farm.

1.1 Wind Flow Modelling and Downscaling

To model wind flow, computational fluid dynamics (CFD) simulations is state-of-the-art. However, on the scale of wind farms these simulations are so computationally demanding that they are often impractical or impossible to use in practice. More data-driven approaches can therefore be utilised as a computationally cheaper way of predicting wind flow. This has lead to a lot of research into both purely data driven and more physics guided methods for estimating wind flow of wind parks [2]. That research is mostly focused on predicting wake effects of wind turbines, as these can significantly reduce the effectiveness of turbine layouts. These techniques are dependent on good input data for the wind flow in the area.

Decent low resolution data is usually easily available, with databases like ERA5 [3]. These are useful, as they provide general wind statistics for a wind farm location. However, onshore the shape of the terrain significantly influences the wind flow, and the data is too coarse to capture the local topographical adaptation of the wind field. The higher resolution of wind data one has, the better one can find the locally most attractive places for wind turbines, and the better one can estimate the wake effects between the turbines. Therefore, methods for what is usually referred to as *downscaling*, to produce approximate high resolution data from low resolution wind data, are applied to (among other applications like weather forecasting) improve the production estimates of wind developers.

Atmospheric wind flow is hard to model exactly, and have scale dependent dynamics. This makes downscaling an interesting engineering challenge of combining our knowledge of the physics with a diverse range of mathematical and computational tools effectively. Different assumptions are made on different scales, and different methods might be effective for different assumption regimes, and according to what features of the wind flow one most wants to model correctly. Dynamical downscaling [4] [5] involves applying a high resolution numerical model on a small part of the area of interest, and then apply the relations found between the low resolution and high resolution data in that area to approximate high resolution wind flow outside of the high resolution domain. Empirical-statistical downscaling [6] [7] [8] [9] does not involve simulating a high-resolution region, instead using statistical models to infer high resolution predictands from low resolution predictors from a dataset of known predictor-predictand paris.

Recently, these methods have been more often combined [10] or replaced with machine learning methods, especially deep convolutional neural networks (CNNs) [11] [12], employing techniques often developed for image segmentation. Convolutional generative adversarial networks (GANs) have also proven effective for images, especially for *super* *resolving* (SR) low resolution images to higher resolution. Such techniques have also made their way into downscaling wind data. Miralles et al. (2022) [13] succesfully downscaled historical 2D wind data in from a 25km x 25km to 1.1km x 1.1km grid using an adversarial approach aided by detailed topographical data. Stengel et al. (2020) [14] employed a two step convolutional GAN to downscale 2D wind data all the way from 100km x 100km to 2km x 2km with good looking results. They did this by training one network on generating a medium resolution dataset from a low resolution dataset and another on generating the high resolution data from the medium resolution data.

In this thesis a fully convolutional GAN is employed to downscale wind data, but working with 3D wind flow, and on a smaller scale than the above, from 800m x 800m to 200m x 200m in the horizontal plane.

1.2 Project Description

The project builds on previous work by Thomas Nakken Larsen 2020 [15] and Tran et al. 2020 [16] on a perceptually driven GAN for super resolving a 2D slice of the wind field. Their idea was to convert wind vectors to RGB colors and utilise a well-known image super resolving architechture involving a pretrained image feature extractor. Larsen concluded that this was not a good idea, as perceptually interesting features of a picture are not necessarily the same as the physically important features of a wind field. This project instead aims to replace the perceptual approach with a more physically guided loss function to train the network to approximate the actual physics of 3D atmospheric wind flow.

More specifically, Three points will be investigated:

1. Methods for applying a 3D convolutional neural network to unevenly spaced atmospheric data. (**Experiment 1**)
2. Physically guided loss functions to help the model approximate the physics of 3D atmospheric flow. (**Experiment 2** and **Experiment 2.5**)
3. Design, performance and usefulness of the best performing model (**Chapter 5**).

Chapter 2.1 will go through the physics of atmospheric wind flow, deriving the equations that are used to generate the dataset that will be used, highlighting key properties with respect to the machine learning techniques that will be employed. Then the most relevant machine learning methods and GAN architectures are covered in Chapter 2.2. Chapter 3 covers the dataset, data processing, model architecture, loss function and experimental setups. Chapter 4 presents the results of the experiments, and discussion of these results. Chapter 5 evaluates the best performing model. Finally, Chapter 6 sums up and concludes the thesis.

Chapter 2

Theory

2.1 Physics of Atmospheric Wind Flow

To understand the data that will be used, and how to evaluate the model's approximation of wind flow physics, I derive the equations that are used to generate the data.

General fluid flow is governed by mass (2.1), momentum (2.2) and energy (2.3) conservation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{2.1}$$

$$\rho \frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = \nabla \cdot \tau + \mathbf{f}, \tag{2.2}$$

$$\rho \frac{D}{Dt}\left(e + \frac{1}{2}\mathbf{u}^2\right) = \mathbf{f} \cdot \mathbf{u} + \nabla \cdot (\tau \cdot \mathbf{u}) - \nabla \cdot \mathbf{q} \tag{2.3}$$

where $\mathbf{u}$ is the velocity field, t is time, ρ is mass density, τ is the stress tensor of the fluid, e is internal energy of the fluid, $\mathbf{q}$ is the heat flux, $\mathbf{f}$ is the sum of body forces on an infinitesimal volume of the fluid and $\frac{\mathrm{D}}{\mathrm{D}t} = \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla$, the material derivative. For a Newtonian, isotropic fluid, with no other forces than gravity (neglecting the coriolis force for atmospheric flow), we can rewrite τ and get the Navier-Stokes momentum equation

$$\frac{\mathrm{D}(\rho \mathbf{u})}{\mathrm{D}t} = -\nabla p + \nabla \cdot \left(\mu \left[\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathrm{T}} - \frac{2}{3}(\nabla \cdot \mathbf{u})\mathbf{I}\right]\right) + \rho \mathbf{g}. \tag{2.4}$$

Here, the stress tensor has been separated into a volumetric contribution coming from the pressure p gradient and a deviatoric part coming from the velocity gradient. If we dot the above equation with $\mathbf{u}$, subtract it from (2.3) and use the same assumptions as before (more details in [17]), we can rewrite (2.3) to

$$\rho \frac{De}{Dt} = -p(\nabla \cdot \mathbf{u}) + \phi - \nabla \cdot \mathbf{q}, \tag{2.5}$$

with ϕ as the viscous component of the deformation work rate:

$$\phi = \mu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathrm{T}} \right) : \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathrm{T}} \right) - \frac{2}{3}\mu(\nabla \cdot \mathbf{u})^2 \tag{2.6}$$

Here, : is the double dot product.

2.1.1 The Boussinesq Approximation

Now, applying the Boussinesq approximation [18], we assume that density variations are small, $\rho = \rho_0 + \delta\rho, \delta\rho \ll \rho_0, \rho_0 = const$, so that these variations only affect the buoyancy term ρg of the equation, while in the other terms we approximate $\rho \approx \rho_0$. Applying the approximation to (2.1) and (2.4) we get

$$\nabla \mathbf{u} = 0 \tag{2.7}$$

$$\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = -\frac{1}{\rho_0}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g}\frac{\rho}{\rho_0}, \tag{2.8}$$

with $\nu = \frac{\mu}{\rho_0}$. Neglecting ϕ (usually very small), assuming an ideal gas, setting $de = C_V dT$ (T is temperature and C_V is heat capacity at constant volume), inserting (2.1) for $\nabla \cdot \mathbf{u}$ and using the boussinesq approximation, we can simplify equation (2.5) to

$$\rho C_p \frac{DT}{Dt} = \nabla \cdot (\kappa \nabla T), \tag{2.9}$$

with $\mathbf{q} = \kappa \nabla T$, κ as the thermal conductivity of the fluid and C_p as the heat capacity at constant pressure.

2.1.2 The Anelastic Approximation

In the anelastic approximation, instead of only assuming small absolute deviations in density, one assumes small deviations in ρ, p, and potential temperature θ with regards to a homoentropic hydrostatically balanced reference state. Also assuming an ideal gas, we have

$$\frac{\partial p_0}{\partial z} = \rho_0(z) g \tag{2.10}$$

$$\rho = \rho_0(z) + \delta\rho \tag{2.11}$$

$$p = p_0(z) + \delta p \tag{2.12}$$

$$\theta = \theta_0 + \delta\theta \tag{2.13}$$

$$\theta = T \left(\frac{p_{ref}}{p} \right)^{\frac{R}{C_p}} = \frac{Rp}{\rho} \left(\frac{p_{ref}}{p} \right)^{\frac{R}{C_p}}, \tag{2.14}$$

where p_{ref} is some reference pressure. Taylor expanding (2.14) around $\theta_0(p_0, \rho_0)$ to first order we get

$$\theta = \theta_0 \left(1 + \left[\frac{(1-\kappa)\delta p}{p_0} - \frac{\delta\rho}{\rho_0}\right]\right). \tag{2.15}$$

Using these relations to rewrite (2.4) we get

$$\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = -\nabla\left(\frac{\delta p}{\rho_0}\right) + \nu\nabla^2\mathbf{u} + \mathbf{g}\frac{\delta\theta}{\theta_0} \tag{2.16}$$

2.1.3 Reynolds Averaged Navier-Stokes equations

Looking for a steady state solution of the wind field, we perform a Reynolds decomposition of the velocity field into an average and a time dependent component $\mathbf{u}(\mathbf{x}, t) = \overline{\mathbf{u}}(\mathbf{x}) + \mathbf{u}_f(\mathbf{x}, t)$. Inserting this into (2.1) and (2.4), then time averaging we get the Reynolds Averaged Navier-Stokes (RANS) equations:

$$\frac{\partial\overline{\rho}}{\partial t} + \nabla\cdot(\overline{\rho\mathbf{u}}) = 0 \tag{2.17}$$

$$\frac{\mathrm{D}(\overline{\rho\mathbf{u}})}{\mathrm{D}t} = -\nabla\overline{p} + \nabla\cdot\left(\mu\left[\nabla\overline{\mathbf{u}} + (\nabla\overline{\mathbf{u}})^{\mathrm{T}} - \frac{2}{3}(\nabla\cdot\overline{\mathbf{u}})\mathbf{I}\right] - \overline{\rho\mathbf{u}_f\otimes\mathbf{u}_f}\right) + \overline{\rho}\mathbf{g}, \tag{2.18}$$

We now have an equation with only the time averaged quantities, except for the $\nabla(\overline{\rho\mathbf{u}_f\otimes\mathbf{u}_f})$ term. This term is highly non-linear, and is seen to act as a kind of stress on the fluid, called the Reynolds stress. A turbulence closure model is needed to make the RANS equations solvable.

2.1.4 Eddy Viscosity and the k–ε Turbulence Model

Turbulence is notoriously hard to model exactly, but several methods have proven useful for approximating turbulent flow. The standard way of doing this for RANS models is to model the turbulence as a modification of the viscosity of the fluid, called eddy viscosity. [19]

$$-\overline{\rho\mathbf{u}_f\otimes\mathbf{u}_f} = \mu_t\left[\nabla\overline{\mathbf{u}} + (\nabla\overline{\mathbf{u}})^{\mathrm{T}} - \frac{2}{3}(\nabla\cdot\overline{\mathbf{u}})\mathbf{I}\right] + \frac{2}{3}\overline{\rho}k\mathbf{I} \tag{2.19}$$

$$k = \frac{1}{2}\overline{\mathbf{u}_f\cdot\mathbf{u}_f} = \frac{1}{2}\,\mathrm{Tr}(\overline{\mathbf{u}_f\otimes\mathbf{u}_f}) \tag{2.20}$$

We see that this makes the Reynolds stress serve as a viscosity term with eddy viscosity μ_t:

$$\frac{\mathrm{D}(\overline{\rho\mathbf{u}})}{\mathrm{D}t} = -\nabla p' + \nabla\cdot\left((\mu+\mu_t)\left[\nabla\overline{\mathbf{u}} + (\nabla\overline{\mathbf{u}})^{\mathrm{T}} - \frac{2}{3}(\nabla\cdot\overline{\mathbf{u}})\mathbf{I}\right]\right) + \overline{\rho}\mathbf{g}, \tag{2.21}$$

with p' absorbing the isotropic part of the Reynolds stress, $p' = \overline{p} - \frac{2}{3}\overline{\rho}k\mathbf{I}$. We can similarly use the eddy viscosity to approximate the effective thermal diffusivity [20]:

$$\frac{\mathrm{D}(\overline{\rho\theta})}{\mathrm{D}t} = \nabla\left(\left(\frac{\kappa}{C_p} + \frac{\mu_t}{\sigma_\theta}\right)\nabla\overline{\theta}\right), \tag{2.22}$$

with σ_θ as the turbulent Prandtl number for heat, usually set between 0.7 and 0.9. So in the same way we model the turbulence as an effective viscosity increase of the average wind flow, we model turbulence as an effective diffusivity increase for the average potential temperature.

The eddy viscosity depends on wind flow, so we use the $k{-}\varepsilon$ [19] model to fully close the equations:

$$\frac{\mathrm{D}(\overline{\rho}\mathrm{k})}{\mathrm{D}t} = \nabla\left(\left(\mu + \frac{\mu_t}{\sigma_k}\right)\nabla\mathrm{k}\right) + \overline{\rho}P_k + \overline{\rho}P_b - \overline{\rho}\varepsilon \tag{2.23}$$

$$\frac{\mathrm{D}(\overline{\rho}\varepsilon)}{\mathrm{D}t} = \nabla\left(\left(\mu + \frac{\mu_t}{\sigma_\varepsilon}\right)\nabla\varepsilon\right) + \frac{\varepsilon\overline{\rho}}{\mathrm{k}}\left(\mathrm{C}_1P_k + \mathrm{C}_3P_b\right) - \mathrm{C}_2\overline{\rho}\frac{\varepsilon^2}{\mathrm{k}} \tag{2.24}$$

$$\nu_t = \frac{\mu_t}{\overline{\rho}} = \mathrm{C}_\mu\frac{\mathrm{k}^2}{\varepsilon} \tag{2.25}$$

Here, k is still time averaged turbulent kinetic energy [(2.20)], ε is the time averaged turbulent kinetic energy dissipation rate, C_μ, C_1, C_2, C_3, σ_k and σ_ε are empirical constants, usually set to 0.09, 1.92, 1.43, 1.0, 1.0 and 1.3, respectively. P_k and P_b are turbulent kinetic energy sources from respectively shear velocity and buoyancy [21]:

$$P_k = \nu_t\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\frac{\partial u_i}{\partial x_j} \tag{2.26}$$

$$P_b = -\frac{g}{\theta}\frac{\nu_t}{\sigma_T}\frac{\partial\theta}{\partial z}, \tag{2.27}$$

with $x_1, x_2, x_3 = x, y, z$ and $(u_1, u_2, u_3) = \overline{\mathbf{u}}$.

The $k{-}\varepsilon$ model has a sound empirical basis but we can also make sense of it intuitively. Our goal was to model the $\nabla(\overline{\rho\mathbf{u}_f \otimes \mathbf{u}_f})$ term in the RANS momentum equation. As it is given by the average fluctuations from the average flow in an area, it represents average chaotic (turbulent) wind flow in that area. Instead of dealing with the 3D details of this chaotic flow we define a scalar μ_t that measure the *level of chaoticity*, and aim to set the value of this scalar so that it affects the steady state wind flow the way viscosity affect general fluid flow. Viscosity represents a fluid's resistance to deformation, so we model the turbulence as viscosity of the steady flow, as the chaotic patterns will oppose the steady state wind flow similarly to how viscosity opposes normal flow.

Equations (2.23) and (2.24) introduced the turbulent kinetic energy k and the turbulence kinetic energy dissipation rate ε to estimate strength of that viscous effect. Equation (2.23) tells us that turbulent kinetic energy is induced by shear velocities (P_k) in the average flow and buoyancy (P_b), and removed by ε. Equation (2.24) tells us that P_k and P_b also produces ε, but scaled with $\frac{\varepsilon}{k}$, so that it is produced quickly for small k, but less quickly when k is bigger. Large kinetic energy also inhibits the sink term (proportional to $\frac{\varepsilon^2}{k}$), so that ε remains sizeable as long as k is sizable, though with the ε^2 ensuring that ε doesn't become too large. What we end up with is a distribution of average turbulence intensity and dissipation rate in space. We use this to model the strength of the eddy viscosity as $\mu_t \propto k\frac{k}{\varepsilon}$. That the introduced virtual viscosity increases with turbulent kinetic energy is intuitive, the $\frac{k}{\varepsilon}$ can be thought of as a measure of the resistance of the turbulence to being dissipated.

2.1.5 Resulting Equations and Key Takeaways

Usually, for atmospheric wind flow $\mu \ll \mu_t$, $\frac{\kappa}{C_p} \ll \frac{\mu_t}{\sigma_\theta}$, so that the molecular viscosity and molecular diffusivity can be neglected. Doing that and applying the anelastic approximation to the $k-\varepsilon$ equations (2.21)-(2.24), we can combine everything to this set of equations:

$$\nabla\overline{\mathbf{u}} = 0 \tag{2.28}$$

$$(\overline{\mathbf{u}} \cdot \nabla)\overline{\mathbf{u}} = -\nabla\left(\frac{\overline{\delta p'}}{\rho_0}\right) + \nu_t\nabla^2\overline{\mathbf{u}} + \mathbf{g}\frac{\overline{\delta\theta}}{\theta_0}. \tag{2.29}$$

$$\frac{\mathrm{D}k}{\mathrm{D}t} = \nabla\left(\frac{\nu_t}{\sigma_k}\nabla k\right) + P_k + P_b - \varepsilon \tag{2.30}$$

$$\frac{\mathrm{D}\overline{\theta}}{\mathrm{D}t} = \nabla\left(\frac{\nu_t}{\sigma_\theta}\nabla\overline{\theta}\right) \tag{2.31}$$

$$\frac{\mathrm{D}\varepsilon}{\mathrm{D}t} = \nabla\left(\frac{\nu_t}{\sigma_\varepsilon}\nabla\varepsilon\right) + \frac{\varepsilon}{k}\left(\mathrm{C}_1 P_k + \mathrm{C}_3 P_b\right) - \mathrm{C}_2\frac{\varepsilon^2}{\mathrm{k}}, \tag{2.32}$$

The above set of equations can be solved numerically to find stable steady state solutions of atmospheric wind flow. Before moving on, I highlight some relevant characteristics of these equations:

- **The gradient of the wind field is highly constrained.** The wind field should be divergence free, conserve momentum, and accurately model turbulent effects. All of this creates strong constraints on the wind gradient. Also, large gradients signal an area of interest, as the features of the wind field that are hard to model are coupled to larger wind gradients.
- **Terrain shape is a major influencer of the wind field.** The divergence free criterion (2.28) means that when the wind is blowing in a certain direction, hills in the terrain will act like sources for the wind field, pushing the wind in other directions, and dips in the terrain act like sinks, pulling the wind flow toward them. The $k-\varepsilon$ model tells us that this pushing and pulling will induce turbulence effects, significantly complicating the wind flow close to the ground.

- **Differences in height above ground level means different dynamics for the wind field.** Equation (2.10) tells us that pressure is decreasing quickly with increasing height. Also, a normal boundary condition applied in CFD simulations is that the wind velocity is zero at ground level, and has approximate wind profile $u_z = \frac{u_*}{\kappa}\left[\ln\left(\frac{z-d}{z_0}\right)\right]$ [22], with u_*, κ, d, z_0 as shear velocity, Karman's constant, ground level height and surface roughness length (length of typical blockers like trees) respectively. The point is that wind speed increases quickly with elevation close to the ground. That also further emphasises the the significance of terrain shape, as varying ground altitude moves this wind profile and the no-slip boundary. All in all, it should be clear that the horizontal coordinates and the vertical coordinates cannot be treated the same.

These three points will prove relevant for designing the data-driven model tested in this thesis.

2.2 Relevant Machine Learning Methods

Machine learning is a broad term, used for a variety of methods involving some kind of algorithmic learning from data. Here, I give a summary of the main machine learning techniques applied in this thesis. As these were covered extensively in Thomas Larsen's thesis [15], and are generally well known, this chapter and Chapter 2.3 will not be very detailed, instead focusing on key attributes of the most relevant techniques.

Machine learning is usually divided into supervised and unsupervised learning. Supervised learning is driven by labeled input-output ($\mathbf{x}-\mathbf{y}$) datasets. The goal is to teach the model to reliably predict the labeled output $\mathbf{y}$ from the input $\mathbf{x}$, so the trained model can produce the desired output for unlabeled inputs. In unsupervised learning one does not have labeled data, but instead the goal is for the algorithm to discover patterns in the data.

2.2.1 Artificial Neural Networks and Backpropagation

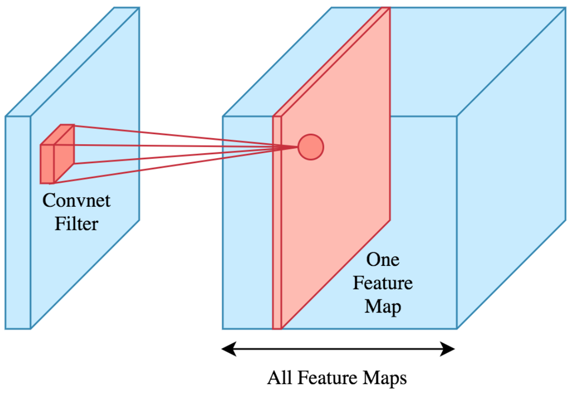
An artificial neural network (ANN) is a set of matrices, bias and activation functions that transforms input to output. An ANN is dividided into layers, with the ith layer being a combination of a weight matrix $\mathbf{W}_i$, a bias vector $\mathbf{b}_i$ and an (element wise) activation function f_i to transform input vector $\mathbf{x}_i$ to output vector $\mathbf{x}_{i+1}$:

$$\mathbf{x}_{i+1} = f_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i) \tag{2.33}$$

The activation function is often chosen as a simple, but non-linear function. This non-linearity enables ANNs to approximate *any* function [23]. An input can be transformed through many layers before producing the final output, which is then evaluated by a loss function. Then we can calculate the gradient of this loss function for all the weights and biases of the entire model, by tracing the output back through the network and see how each weight contributed to the resulting loss. This is known as backpropagation. With the gradient in hand one can then modify each weight and bias slightly in the direction of less loss. Multilayered ANNs are often referred to as "deep learning". Such models can quickly become very large and computationally demanding, so many techniques have been developed to make the models more effective.

2.2.2 Convolutional Neural Networks

Instead of allowing all the parts of the input vector to affect each output of a layer like in a *fully connected layer* [(2.33)], a convolutional neural network (CNN) applies convolutional filters of a certain size *across* the data to capture key features of each local region, meaning that one convolutional layer applies the same small filter across the entire spatial dimension of the data. This makes sense for spatial data, as all points in space are a priori equivalent, and are naturally most influenced by their immediate surroundings. Relative



Source: `https://brilliant.org/wiki/convolutional-neural-network/`

Figure 2.1: Convolutional Neural Network. The figure demonstrates how the output of a convolutional layer is only affected by its immediate surroundings, and how one often extracts a large amount of features when using CNNs.

to fully connected layers, convolutional layers are extremely cheap in terms of number of parameters. One can scale the data as much as one wants in the spatial dimension without increasing the number of weights – the same filter is applied across all the data. Instead, one can extract a large number of *features*, as demonstrated in Figure 2.1, increasing the depth of the data with feature maps aiming to capture different key properties of the input. Deep CNNs have been shown to be very effective in image segmentation, with initial convolutional filters capturing features like edges or textures, and later layers in the network being able to identify sophisticated traits like tumors or facial expressions.

2.2.3 Residual Learning

In residual learning, instead of transforming the input completely in each layer, an incremental modification is added to the input each time:

$$\mathrm{x_{i+1}} = \mathrm{x_i} + \mathrm{M}(\mathrm{x_i}) \tag{2.34}$$

Here $\mathbf{M}$ can represent any combination of layers or techniques that outputs data with the same dimensions as $\mathbf{x_i}$. This means that each $\mathbf{M}$ can be optimised for this slight modification, rather than simultaneously figuring out how to both preserve key parts of the original data and modifying it in a full transformation. The technique can be combined in several nested blocks to strengthen this effect further, as each layer can then exploit the information of each previous layer in the block and the original input. Figure 2.2 demonstrates resiudal learning visually.

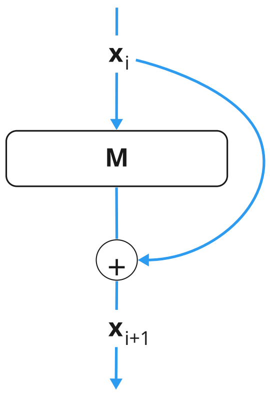


Figure 2.2: Residual Block. A model $\mathbf{M}$ is trained to optimize a modification of $\mathbf{x_i}$, $\mathbf{x_{i+1}} = \mathbf{x_i} + \mathbf{M}(\mathbf{x_i})$ instead of $\mathbf{x_{i+1}} = \mathbf{M}(\mathbf{x_i})$. $\mathbf{M}$ can represent any combination of layers or techniques that outputs data with the same dimensions as $\mathbf{x_i}$

2.2.4 Dropout Layers

A dropout layer regularises a network by randomly zeroing out some of parts of the output of hidden layers during training. This enhances generalization, and prevents overfitting the data, as the different parts of the network are prevented from relying too much on combining each others output, instead having to each contribute more robustly to optimise the loss function. In CNNs this is typically done on a feature level, so each feature has a set probability of being dropped, as shown in Figure 2.3.

Dropout has proven to be an effective regularisation technique, and Kong et al. [24] found that a dropout layer before the last convolutional layer in a super-resolving residual network improved performance and generalisability of the network. Applying both dropout and batch normalization in the same network can produce negative cross-effects [25], but if dropout is applied after the last batch normalisation in the network, this should not be a problem.

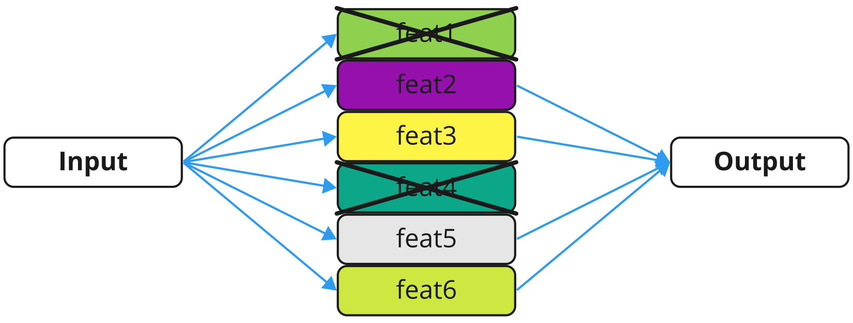


Figure 2.3: Feature dropout layer. Entire feature channels are randomly zeroed out with a set probability.

2.2.5 Gradient Clipping

Gradient clipping means clipping the gradient so that it cannot be larger than a certain threshold. One way of doing this is clipping the total norm of the gradient, so that it never exceeds a set threshold C:

$$\mathbf{g} = \begin{cases} C\frac{\mathbf{g}}{||\mathbf{g}||}, & \text{if } ||\mathbf{g}|| > C \\ \mathbf{g}, & \text{otherwise}, \end{cases} \tag{2.35}$$

with $\mathbf{g}$ as the calculated gradient of the model. Gradient clipping can be beneficial if the loss function has very steep valleys or peaks in parameter space, ensuring that the model doesn't take to take a too large optimisation step during training. A large step can move the model away from a promising area in parameter space, and in the worst case send the model spiralling away from less loss, with the model overstepping the beneficial area more and more with each iteration.

2.3 Generative Adversial Networks

A generative adversial network (GAN) is an unsupervised generative model. A GAN model consists of a generator network **G** and a discriminator network **D** that interact as shown in Figure 2.4.

The generator tries to generate samples $\mathbf{f} = \mathbf{G}(\mathbf{z})$ that are indistinguishable from samples $\mathbf{t}$ of the true distribution $\mathbf{T}$, while the discriminator tries to distinguish the generated samples from true samples. As the generator becomes better, the job of the discriminator gets harder, while as the discriminator gets better, the job of fooling it becomes harder for the generator. So by including the output of the discriminator in the loss function of the generator, both networks can help each other get better. The discriminator outputs a a

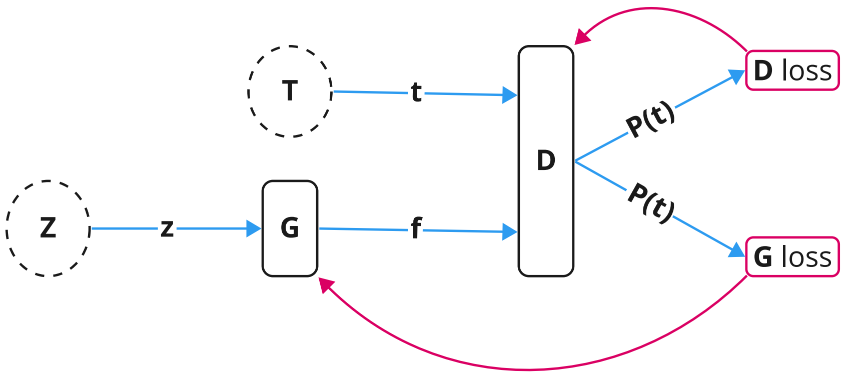


Figure 2.4: A standard GAN model. **T** is a distribution of true samples **t** that the generator **G** is trying to approximate given a random input **z** from its input space **Z**. The discriminator **D** is trained to distinguish samples **t** of **T** from samples **f** = **G**(**z**). The accuracy of **D**'s evaluation **P(t)** is used to train both **G** and **D**.

number between 1 and 0 representing certainty of the input being a true sample (1.0) or a false sample (0.0). Then we can define the loss functions of both networks:

$$L_D^{GAN} = \log\left(\mathbf{D}(\mathbf{t})\right) - \log\left(1 - \mathbf{D}(\mathbf{G}(\mathbf{z}))\right) \tag{2.36}$$

$$L_G^{GAN} = \log\left(1 - \mathbf{D}(\mathbf{G}(\mathbf{z}))\right). \tag{2.37}$$

G is penalised for the discriminator labeling its generated data as false, while **D** is penalised both for not correctly identifying true samples and for being fooled by the generator.

2.3.1 Relativistic Average GANs

Since its introduction in 2014 [26], multiple adjustments of the original GAN model have been introduced. The relativistic average GAN (RaGAN) model was introduced by Jolicoeur-Martineau in 2018 [27]:

$$L_D^{RaGAN} = -\Big\langle \log\left(\bar{\mathbf{D}}(\mathbf{t})\right)\Big\rangle - \Big\langle \log\left(1 - \bar{\mathbf{D}}(\mathbf{f})\right)\Big\rangle \tag{2.38}$$

$$L_G^{RaGAN} = -\Big\langle \log\left(\bar{\mathbf{D}}(\mathbf{f})\right)\Big\rangle - \Big\langle \log\left(1 - \bar{\mathbf{D}}(\mathbf{t})\right)\Big\rangle, \tag{2.39}$$

with

$$\begin{aligned} \bar{\mathbf{D}}(\mathbf{t}) &= \sigma\Big(C\left(\mathbf{t}\right) - \Big\langle C(\mathbf{f})\Big\rangle\Big) \\ \bar{\mathbf{D}}(\mathbf{f}) &= \sigma\Big(C\left(\mathbf{f}\right) - \Big\langle C(\mathbf{t})\Big\rangle\Big). \end{aligned} \tag{2.40}$$

Here σ is the activation function of the discriminator, $C(x)$ is the discriminator output before the activation, $\mathbf{D}(x) = \sigma\big(C(x)\big)$, and the $\langle\rangle$ represents expected value, in practice meaning the batch average value. Now, instead of punishing the discriminator for absolute classification error, it is rewarded for its classification of true samples *relative* to how it classifies false samples. Similarly for the generator. This exploits the knowledge that half of the samples presented to the discriminator are fake. Whereas in the original GAN a perfectly calibrated model would classify all samples as real, a perfectly calibrated RaGAN would classify half of all samples as fake, and half as real. Both the generator and the discriminator are pushed to specifically target the recurring differences between the real and generated data, rather than just the result of each single sample.

2.3.2 GAN Failures and Training Techniques

Using a model with two adversarial networks complicates model training. A common failure mode is called *mode collapse*, in which the generator cycles through specific types of samples from the true distribution, never mapping out the entire space of real samples. Convergence failures can also occur in which the generator and discriminator get stuck in a changing pattern instead of finding a stable equilibrium.

Another failure type is vanishing generator gradients. Initially, it is easy to distinguish real and generated data. The discriminator may therefore be very confident in classifying the generated samples as false. This leads to the generator not receiving meaningful feedback from the discriminator. The generator's gradient vanishes, since small changes in the generator cannot help meaningfully towards fooling the discriminator. The problem is addressed by intentionally confusing the discriminator in early training, by for example smoothing or randomly flipping labels of **t** and **f**. One can also add instance noise to the generated and real samples. These techniques, GAN algorithms and failure modes, are covered in more detail in Larsen's thesis [15].

2.3.3 Single Image Super-Resolution GANs

A Single Image Super-Resolution GAN (SISRGAN) is a GAN network trained to approximate a high-resolution image $\mathbf{I}_{\mathrm{HR}}$ from a single low resolution image $\mathbf{I}_{\mathrm{LR}}$ by generating a *super-resolved* image $\mathbf{I}_{\mathrm{SR}}$. Training of such a network is illustrated in figure 2.5.

SISRGANs using deep residual networks have proven very effective at super resolving images. Of the most effective and well-known of these architectures is the Enhanced Super-Resolution GAN (ESRGAN), introduced by Wang et. al in 2018 [28]. Vesterkjær's 2019 [29] implementation of this architecture can be seen in figure 2.6.

The "enhanced" part refers to ESRGAN's inspiration from Super-Resolution GAN (SRGAN) [30]. In SRGAN, the generator consists of a number of residual dense blocks (RDBs) of Conv-BatchNorm-PRelu-Conv-BatchNorm, and the same discriminator as in Figure 2.6. In figure 2.6 we see how this idea has been taken further, nesting RDBs into what the authors named residual in residual dense blocks (RRDBs) and scale the outputs by a factor $\alpha = 0.2$ (called residual scaling, first used by Szegedy et al. [31]).

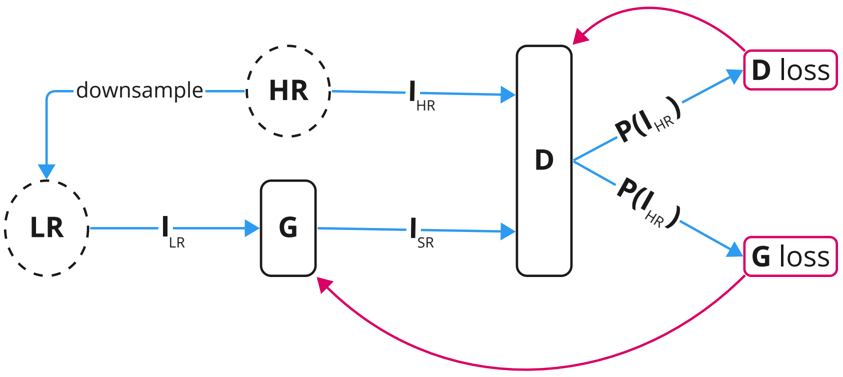


Figure 2.5: A SISRGAN model. $\mathbf{I}_{\text{HR}}$ is downsampled to $\mathbf{I}_{\text{LR}}$, which is used as input for **G** to produce $\mathbf{I}_{\text{SR}}$. **D** tries to distuinguish $\mathbf{I}_{\text{HR}}$ and $\mathbf{I}_{\text{SR}}$, and **G** and **D** are penalized according to the accuracy of **D**s classification.

SRGAN combines the (slightly modified from (2.36)-(2.37)) adversarial loss with a perceptual loss and a pixel loss:

$$L_G^{pix} = MSE(I_{\text{HR}}, \mathbf{G}(I_{\text{LR}})) \tag{2.41}$$

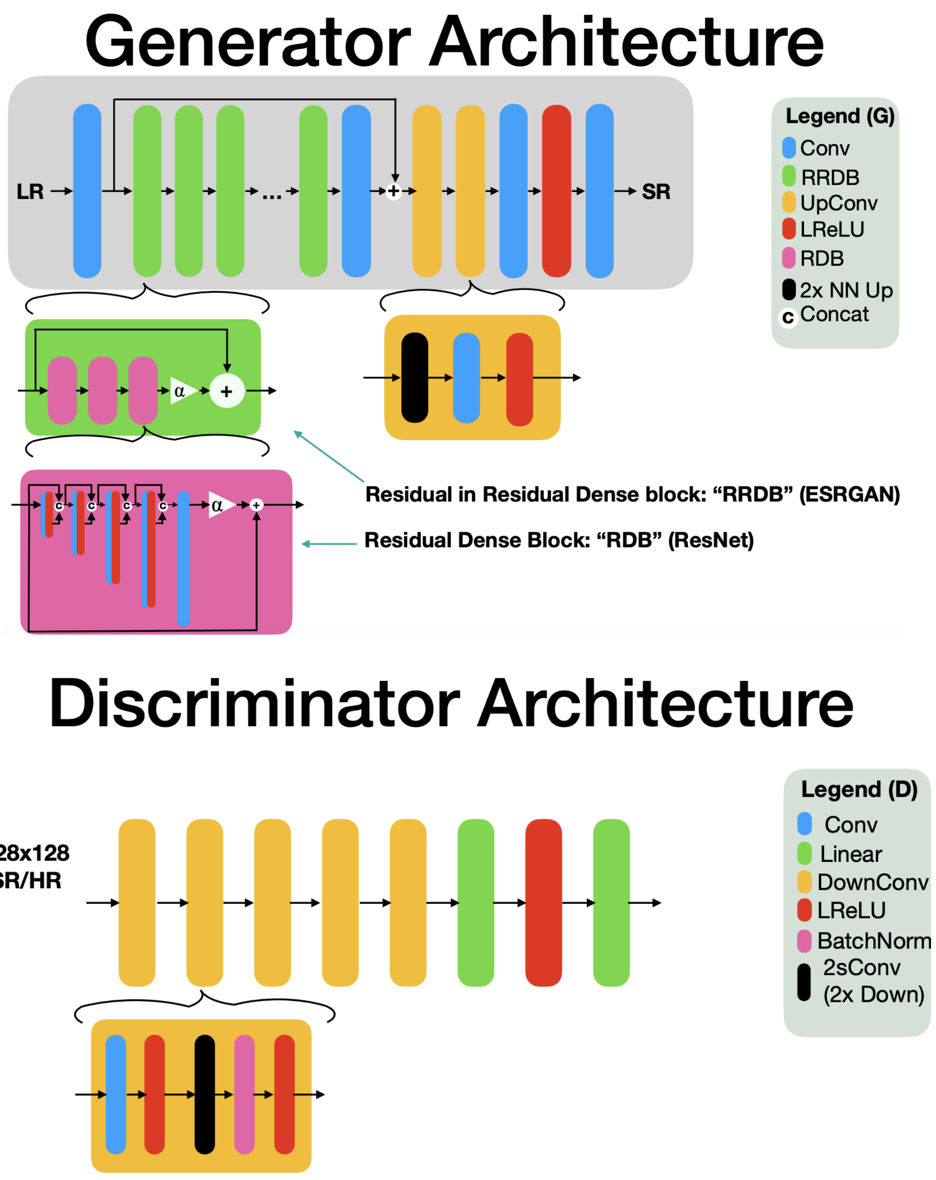
$$L_G^{perceptual_{ij}} = MSE(\phi_{i,j}(I_{\text{HR}}), \phi_{i,j}(\mathbf{G}(I_{\text{LR}})) \tag{2.42}$$

$$L_G = \eta_1 L_G^{pix} + \eta_2 L_G^{perceptual_{ij}} - \eta_3 \log(\mathbf{D}(\mathbf{G}(\mathbf{z}))) \tag{2.43}$$

$$L_D = L_D^{GAN} \tag{2.44}$$

Here, MSE is mean square error, η_1, η_2, η_3 are constants and $\phi_{i,j}$ is the feature map obtained by the j-th convolution after activation, before the i-th maxpooling layer within the pretrained VGG19 [32] network. The perceptual loss therefore penalises the generator according to how differently the generated SR images and the HR images activate features in a network *trained to detect important perceptual features*. The SRGAN generator is first trained on only L_G^{pix}. Then this pretrained model is trained further using L_G with, after some testing, $\eta_1, \eta_2, \eta_3 =$ 0.0, 0.06 (effectively as a result of feature scaling), 0.001 and $\phi_{i,j} = \phi_{5,4}$.

In ESRGAN the perceptual loss is modified to use the feature output from the pretrained image classifier before activation instead of after, thereby gaining more granular feature discrepancies. ESRGAN uses RaGAN losses instead of traditional adversarial loss, and $\eta_1, \eta_2, \eta_3 = 0.01, 1.0, 0.005$. ESRGAN also uses mean square error for pretraining the generator, but average absolute error instead of MSE for L_G^{pix} when used in (2.43). Having the pretrained MSE generator and the trained perceptual GAN generator, they use weight interpolation between the two generators, allowing choice of how much one wants to weigh pure content error versus perceptual features. Finally, they remove batch normalization from the generator, seeing that this increased performance further.



Source: Vesterkjær 2019 [29]

Figure 2.6: ESRGAN architecture as implemented by Vesterkjær 2019 [29]. Conv means 2D convolutional layer, 2sConv is a convolutional layer with stride 2 (halves the dimension of the data), BatchNorm is 2D batch normalisation, LReLU stands for the Leaky ReLu activation function (with negative slope 0.2 in ESRGAN), 2x NN Up means nearest neighbour upscaling with scale factor 2, Concat means concatenating the data along the feature dimension, the + symbol means add, the α triangle means multiply by α (= 0.2 in ESRGAN) and Linear means fully connected layer.

Chapter 3

Method

In Larsen's [15] thesis, the ESRGAN architecture in Figure 2.6 was tested for super-resolving a 2D slice of the wind field. However, in order to use the perceptual loss, each wind component was normalised independently to positive values to simulate RGB pixels. In training, the model failed to minimise the perceptual loss enough for the pixel loss or adversarial loss to have much effect.

The pretrained image classifier used in ESRGAN is trained to find central *perceptual* features like textures, edges and specific object shapes, and Larsen concluded that this translates poorly into central features of the RGB-translated wind field. Also, the normalisation scheme changed the orientation of the wind vectors, not preserving their spatial information. Instead, here the perceptual loss is dropped, and the loss function is modified to capture important *physical* features. Also, as emphasised in Chapter 2.1.5, the terrain shape is the main influencer of the local wind field, so the terrain should not be ignored by the model, but incorporated somehow. Furthermore, not being constrained by the image format, the new model aims to learn complex terrain adaptation of 3D wind flow. Below I describe in more detail the dataset, architecture, loss functions and code of the model.

3.1 Dataset

3.1.1 HARMONIE-SIMRA

The data that will be used is generated by HARMONIE (Hirlam Aladin Regional Mesoscale Operational Numerical prediction in Europe) - SIMRA (Semi Implicit Method for Reynolds Averaged Navier Stokes Equations), a multiscale numerical simulation model, desribed in Rasheed et al. 2017 [21].

Rasheed et al. describes the HARMONIE model as "a nonhydrostatic model, in which the dynamical core is based on a two-time level semi-implicit semi-Lagrangian discretization of the fully elastic equations, using a hybrid coordinate system in the vertical direction ([33])". Instead of using a separate equation for ε like in the k-ε model (Chapter 2.1.4), ε is calculated using these relations:

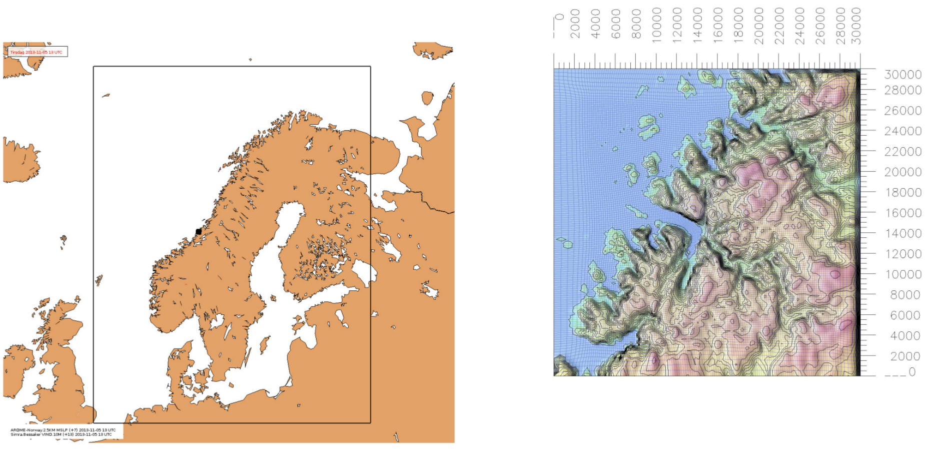
$$\varepsilon = \frac{\left(\sqrt{C_\mu} k\right)^{3/2}}{\ell_t} \tag{3.1}$$

$$\ell_t \approx \frac{\min(\kappa z, 200m)}{1 + 5Ri}, \tag{3.2}$$

$$Ri = \frac{(g/\theta)\partial\theta/\partial z}{(\partial u/\partial z)^2} \approx -\frac{P_b}{P_k} \tag{3.3}$$

with $\kappa = 0.4$, and k still being the average turbulent kinetic energy.

The 750 × 960 × 65 mesh of the HARMONIE model covers a 1875km × 2400km × 26km area, giving it a resolution of 2,5km x 2,5km in the horizontal domain. The simulated HARMONIE data is interpolated and used as input for the SIMRA model, which solves the anelastiq RANS equations derived in Chapter 2.1, (2.28)-(2.32), numerically on a 200x200x40 mesh covering 30km×30km×2.5km with higher resolution close to the wind farm. The actual geographical domains can be seen in Figure 3.1.



Source: Rasheed et al. [21]

Figure 3.1: Area covered by the HARMONIE-SIMRA model, HARMONIE to the left, and SIMRA to the right.

The hourly generated SIMRA data on a 200m x 200m x 40 resolution is available at `https://thredds.met.no/thredds/catalog/opwind/catalog.html`. This is the dataset that will be used as ground truth in this thesis.

3.1.2 Properties of the Data

When running CFD simulations, one designs a mesh that is much more detailed close to objects that complicate the wind flow, like terrain, than in areas of expected simple flow. The actual horizontal meshing used in SIMRA can be seen in Figure 3.1. The simulated data has been adapted to a regular 200m x 200m grid, but the data still has uneven vertical spacing, with dense terrain following layers close to the ground and gradually more flat horizontal layers as we move higher up. This pattern can be seen clearly in Figure 3.2.

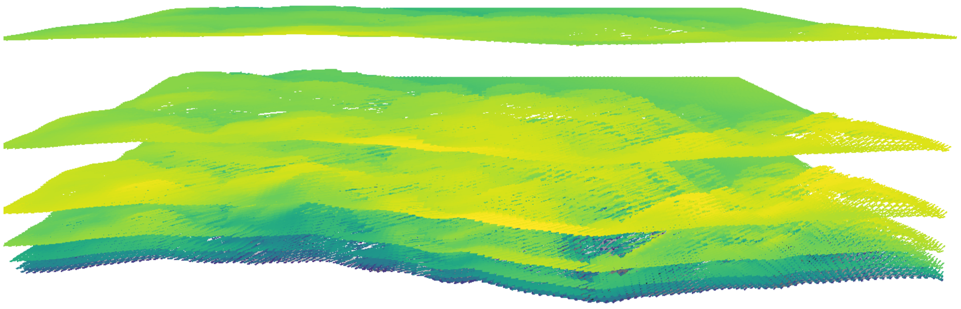


Figure 3.2: Shape of the dataset. Showing every fifth layer of data in the vertical coordinate. The z-coordinate has been multiplied by five to highlight the shape. Yellow marks high wind speed, blue marks low wind speed.

Zooming in on a 32x32x10 slice of the dataset, and not scaling the vertical axis, we can see the wind field in more detail in Figure 3.3. We see how the terrain affects the wind, pushing higher wind speed uphill, and less, even reversed wind speed downhill. Figure 3.3 also demonstrates how dense the horizontal layers are close to the ground, a normal property of simulated data, to help model the complexities of near-surface wind flow.

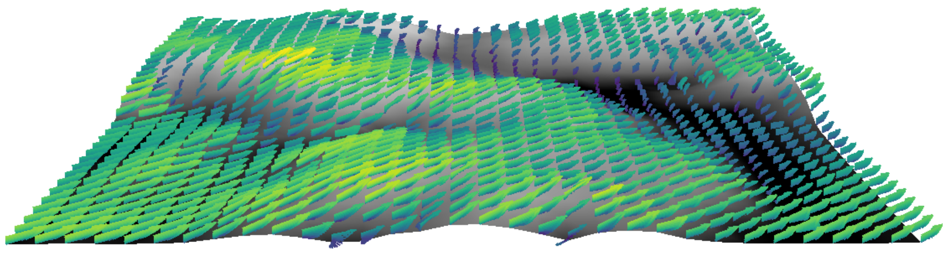
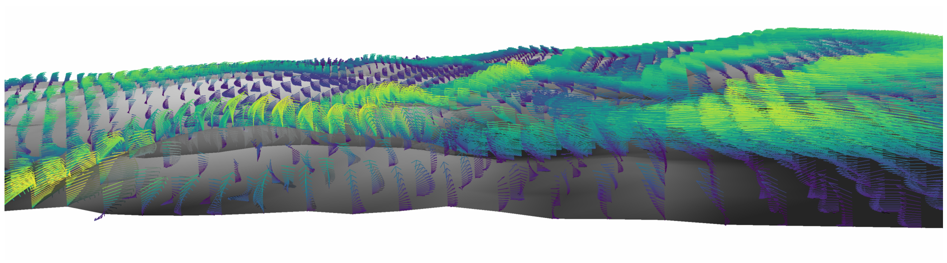


Figure 3.3: Zoomed in unscaled wind field. Showing the 10 bottommost layers in a 32x32 slice of the data area. Yellow marks high wind speed, blue marks low wind speed.

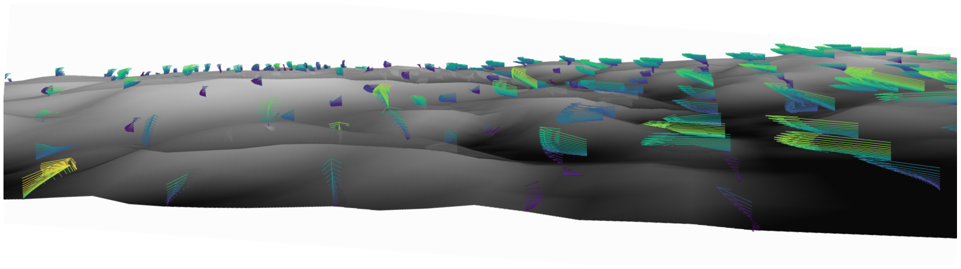
For our purposes, the uneveness in the vertical spacing is an advantage and a challenge at the same time. Higher information density in areas that are hard to approximate is beneficial for accurate modelling, and when designing wind farms one is interested in the wind speed relatively close to the ground, where turbines can harvest the wind energy.

At the same time, it poses a challenge for applying CNNs, as they, in applying the same convolutional filter across the entire spatial dimension, assume that the same relations hold between neighbouring points in one part of the data as others. If they don't, the CNNs have to compromise between features that are useful in different parts of space. Hence **Experiment 1**, where different approaches to handling the vertical spacing is tested and compared.

Zooming further, in Figure 3.4, we can see some of the turbulence effects that makes it hard to model near-surface wind fields in complex terrain. We can see how dips and bumps in the landscape induce swirls and chaotic wind patterns, as described in Chapter 2.1.5. From the figure we also get a picture of the difficulty of super-resolving the wind field as intended, as a perfect generator would have to infer the full swirling pattern of Figure 3.4a from the data we see in Figure 3.4b. Comparing such areas in the generated and real data therefore provides a key indicator of how well the model has learned to model the physics of atmospheric wind flow.



(a) High resolution wind field



(b) Low resolution wind field

Figure 3.4: Turbulent Terrain effects. Showing the 16 bottommost layers in a chaotic region of the wind field.

These patterns also demonstrate an advantage of working with full 3D data rather than 2D slices, as these effects are hard to pick up from one 2D slice. Recall that we introduced two new equations to model how turbulence is induced and dissipates in space. It is a highly 3D phenomena, as the turbulence propagates and interacts in space. Therefore, to accurately model a 2D slice without overfitting a certain terrain, the model would have to incorporate some spatial understanding of how these effects behave in 3D space. Working in 3D, we can guide this spatial learning properly.

3.1.3 Data Augmentation and Processing

The data processing, augmentation and loading can be summarised in seven steps:

1. Download available data from `https://thredds.met.no/thredds/catalog/opwind/catalog.html`
2. Extract the selected spatial part of the data, stripping off the edges, slicing the chosen horisontal layers and checking for any invalid values. (136x135x41 $\rightarrow$ 128x128x[chosen number of horisontal layers], set to 128x128x10 for the experiments in this thesis)
3. Save each data sample to a file and extremal values of the data to another file (for deciding normalization factors of a selected training set).
4. Split the data into train, validation and test sets with, respectively, ratios 0.8, 0.1, 0.1, and save normalization factors from only the training dataset (to avoid peeking). The split is done along the time axis, so that the validation and test sets contain data from other time periods, but still in the same geographical domain. The training data is always shuffled during training.
5. Create customised datasets and dataloaders that when providing a sample loads a single file, includes the specified input channels, normalises the data, downsamples to create the low resolution data and returns LR data, HR data and z-coordinates of the data. The downsampling is done simply by keeping every "scale"th point along each of the horizontal axes (LR = HR[::scale, ::scale, :]), with scale=4 for all results in this thesis except those in Chapter 5.2.3. The z-coordinates are not downsampled, as the model utilises high resolution terrain information (see Chapter 3.2). The HR-data is always just the wind field, but the LR-data may include low resolution z-coordinates, low resolution pressure, low resolution terrain or low resolution height above ground (See **Experiment 1**).
6. If enabled, the customized dataset interpolates (or loads from saved interpolated data) along the z-axis as specified in Algorithm 1.
7. If enabled, the customized dataset applies the following data augmentation techniques during training and validation:
   - **Horizontal slicing:** Change the HR data to a random horizontal 64x64x[chosen number of horizontal layers] slice of the 128x128x[chosen number of horizontal layers]. LR data is adjusted correspondingly. When horizontal slicing is enabled, with scale 4 and chosen number of horizontal layers 10, during training LR is of dimension 16x16x10, and HR and z-coordinates is of dimension 64x64x10.
   - **Rotation:** Rotate a random amount (randint(0,4)*90°) around the z-axis.
   - **Flipping:** Flip the data across the x or y-axis with 50% probability.

If all the data augmentation steps are taken, as is the case for all the discussed experiments except in Chapter 5.2.3, the amount of total possible data samples is increased by a factor of $64 * 64 * 4 * 2 * 2 = 65536$. Of course, only the rotations and flipping produces "new" wind flow, but the slicing exposes the model to more variations in the terrain and what datapoints are sampled as input to the model, as these are selected relative to the slicing area.

The slicing also causes a disparity in how much different parts of the data is utilised in training, as a pixel in the horizontal center of the dataset has a $\left(\frac{63}{64}\right)^2$ chance of being included in a slice, while a pixel in the corner has a $\left(\frac{1}{64}\right)^2$ chance of being included. The disparity is ameliorated somewhat by sampling from a β-distribution $Beta(\alpha, \beta)$ with $\alpha = \beta < 1$, so that it is more likely to slice closer to the edge. Figure 3.5 compares the resulting 2D distributions to show the skewness in utilisation of the dataset.

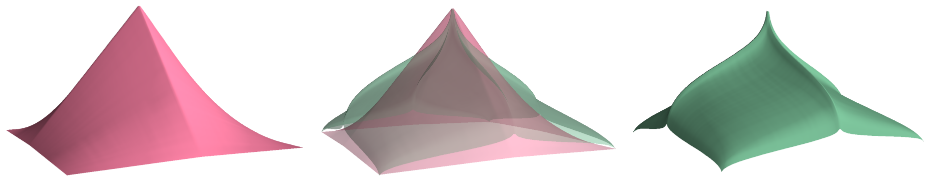


Figure 3.5: 2D Sampling Distribution due to Data Augmentation. Comparing uniform and β-distribution sampling of a 64x64 slice of a 128x128 grid. The surfaces show how often a pixel is included in a sample using the respective sampling techniques.

Though a single sample from the custom dataset only uses 25% of the (non-augmented) data, validation is still done by iterating through the validation dataloader one time. When testing, the data augmentaion is turned of, testing on the entire area without rotating or flipping.

After all experiments were run, a bug in the data augmentation process was discovered. When the wind field was rotated or flipped the orientation of the wind vectors remained the same. This is problematic for learning the 3D terrain adaptation of the wind field, as the rotated fields no longer contain the correct directional relations between the wind vectors and the terrain. However, for the model's part, what it learns is to make sure it is symmetric under a certain linear transformation. Due to the bug, this learned symmetry is for a different linear transformation than intended. With the bug, the model has to learn to rotate and flip the vectors to find the useful relations to the terrain, but at the same time, when the orientation of the vectors is kept, it is easier for the model to treat the rotated fields equivalently. In total, we should expect this bug to make it make it slightly easier or slightly harder for the model to correctly solve the problem it is given, depending on the usefulness of the terrain information, and risk of overfitting the terrain is higher.

If the following parameters are included in training, they are normalized according to these equations:

$$u_i^{norm} = \frac{u_i}{\max(\text{abs}(\text{concat}(u_1, u_2, u_3))} \tag{3.4}$$

$$x^{norm} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.5}$$

$$z_{alt}^{norm} = \frac{z_{alt}}{\max(z_{alt})} \tag{3.6}$$

Here, u_is are the wind components, $z_{alt} = z - terrain$ is altitude above ground, and x can represent pressure, z-coordinates or terrain, as these are all normalized according to (3.5). This way the wind components vary between -1 and 1 and the other coordinates between 0 and 1. The choice of z_{alt} normalization (3.6) is done to maintain the relationship between differences in the data relative to the altitude above ground.

When interpolation is enabled the vertical coordinates are interpolated as described in **Algorithm 1**.

Algorithm 1 Interpolate z-coordinates

```
Input: 3D array wind comp or pressure $u$, 3D array altitude above ground $z_{alt}$
Output: $u$ interpolated
$z\_1D \leftarrow \text{linspace}(\text{mean}(z_{alt}[:][:][0]), \text{mean}(z_{alt}[:][:][-1], num = \text{len}(z_{alt}[0][0][:]))$
for $x_i$ in $x$ do
    for $y_j$ in $y$ do
        $u[x_i][y_j][:] \leftarrow \text{interp}(z\_1D, z_{alt}[x_i][y_j][:], u[x_i][y_j][:])$
    end for
end for
return $u$
```

The linspace method makes an evenly spaced array between two values, and interp means simple linear interpolation. The main point is that the coordinates are interpolated only in the z-direction, and onto a regular grid relative to the ground, not regular in absolute terms. So the grid is still irregular, but it is regular with respect to the ground.

3.2 Model Architecture

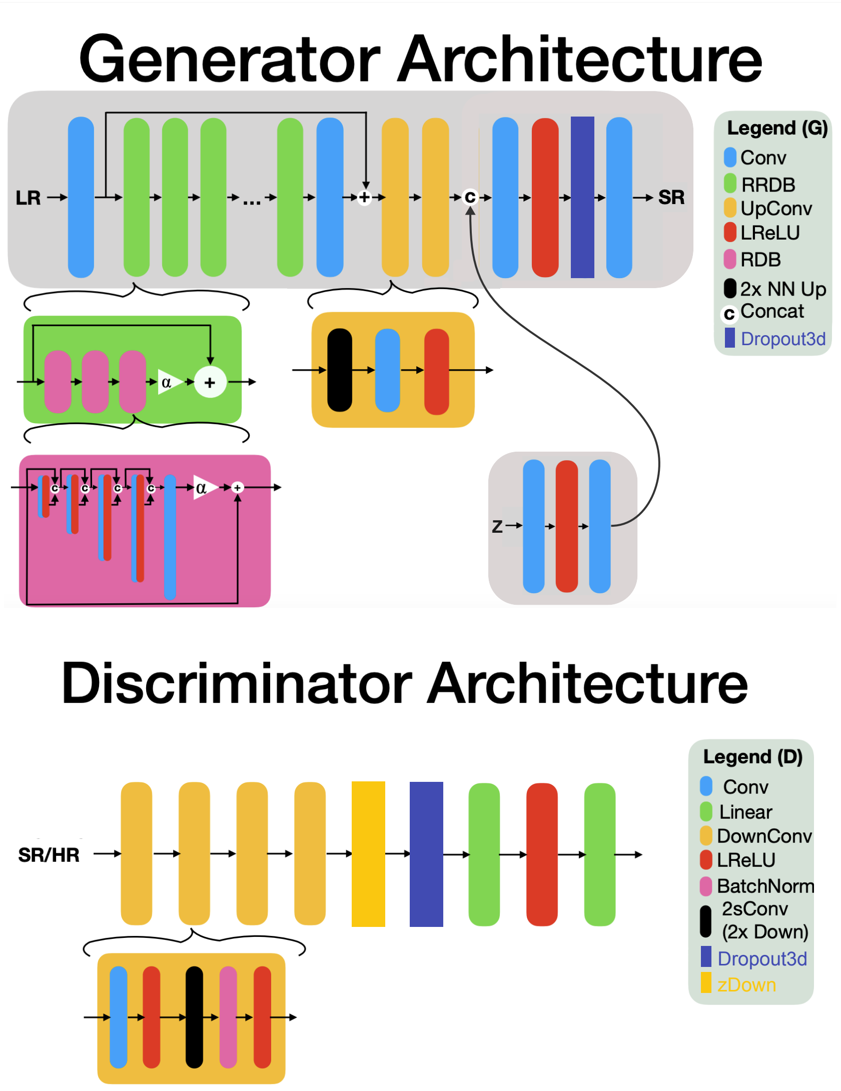
The employed model keeps the ESRGAN architecture shown in Figure 2.6. However some changes have been made:

- The network has been adapted to handle 3D input with a variable number of horizontal layers, moslty by changing from 2D to 3D convolutional layers. The first four "DownConv" blocks (see figure 2.6) in the discriminator maintains the size of the z-dimension, the last layer also halves the z-dimension. When horisontal data slicing is enabled, the last block does not halve the xy-dimensions.
- A feature dropout layer with probability 0.1 has been added before the last convolutional layer of the generator as in [24], and a feature dropout layer with probability 0.2 has been added after the feature extraction ("DownConv") part, but before the classifier part of the discriminator. This is intended to help regularise the models,

making the generator more robust and making it harder for the discriminator to target specific too specific traits of the generated data.

- A Conv-LReLu-Conv 3D terrain feature extractor has been added to the generator. It takes in the z-coordinates of the *high resolution* data and concatenates the features extracted with the extracted features of the rest of the network after the "UpConv" (see figure 2.6) blocks. This is intended to exploit that one usually has high resolution data of the terrain available regardless of the resolution of the wind data.

The resulting architecture can be seen in Figure 3.6.



Source: Modified from Vesterkjær 2019 [29]

Figure 3.6: The modified ESRGAN architecture employed in this thesis. In addition to changing from 2D to 3D data with a variable number of horizontal slices, a feature dropout layer (Dropout3d) has been added to each of the models, and a terrain feature extractor had been added the generator, in order to exploit high resolution terrain information. Conv means 3D convolutional layer, 2sConv is a convolutional layer with stride 2 in the horizontal directions and stride 1 in the vertical direction, zDown is like a DownConv block but with vertical stride 2 and horisontal stride 2 if slice data augmentation is enabled and 1 otherwise. BatchNorm means 3D batch normalisation, LReLU stands for Leaky ReLu activation function (with negative slope 0.2), 2x NN Up means nearest neighbour upscaling with a horizontal scale factor 2, Concat means concatenating the data along the feature dimension, the + symbol means add, the α triangle means multiply by α (= 0.2) and Linear means fully connected layer.

3.3 Loss Function

Three types of loss make up the total loss function: adversarial, pixel and wind gradient based losses. The adversarial losses are the GAN losses discussed in Capter 2.3, and, here as in ESRGAN, the RaGAN losses are used and average absolute error is used as the pixel loss. The gradient based losses are based on the gradient of the wind field, not to be confused with the gradient of the total loss function that is used to train the model. Four different gradient losses are introduced, giving us a total of seven distinct losses:

$$L_G^{pix} = AVG\big(|I_{\mathrm{HR}} - I_{\mathrm{SR}}|\big) \tag{3.7}$$

$$L_G^{\nabla xy} = MSE\Big(\big((\nabla I_{\mathrm{HR}})_{\partial x\partial y}\big)', \big((\nabla I_{\mathrm{SR}})_{\partial x\partial y}\big)'\Big) \tag{3.8}$$

$$L_G^{\nabla z} = MSE\Big(\big((\nabla I_{\mathrm{HR}})_{\partial z}\big)', \big((\nabla I_{\mathrm{SR}})_{\partial z}\big)'\Big) \tag{3.9}$$

$$L_G^{div} = MSE\Big((\nabla \cdot I_{\mathrm{HR}})', (\nabla \cdot I_{\mathrm{SR}})'\Big) \tag{3.10}$$

$$L_G^{div_{xy}} = MSE\Big(\big((\nabla \cdot I_{\mathrm{HR}})_{\partial x\partial y}\big)', \big((\nabla \cdot I_{\mathrm{SR}})_{\partial x\partial y}\big)'\Big) \tag{3.11}$$

$$L_G^{adversarial} = L_G^{RaGAN} \tag{3.12}$$

$$L_D^{adversarial} = L_D^{RaGAN}, \tag{3.13}$$

using the terminology of Chapter 2.3.3, with I_{HR}, I_{LR} and $I_{\mathrm{SR}} = \mathbf{G}(I_{\mathrm{LR}})$ referring to the high resolution wind field, low resolution wind field and generated (super-resolved) wind field. AVG means average component wise value, and MSE means mean squared error. In the notation above, $\nabla I_{\partial x\partial y} = (\frac{\partial u_x}{\partial x}, \frac{\partial u_y}{\partial x}, \frac{\partial u_z}{\partial x}, \frac{\partial u_x}{\partial y}, \frac{\partial u_y}{\partial y}, \frac{\partial u_z}{\partial y})$ and $(\nabla \cdot I)_{xy} = \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y}$, with u_i being the component of the wind field in the i-direction. Similarly for $\nabla I_{\partial z}$, and $(\nabla \cdot I)$ is the divergence of the wind field. The $'$ in the gradient losses represents that they are normalized according to (3.14).

$$\big(I_{j\mathrm{R}}^{\nabla}\big)' = \frac{I_{j\mathrm{R}}^{\nabla}}{\max\Bigg(\max\big(\mathrm{abs}(I_{\mathrm{HR}}^{\nabla})\big), \frac{\max\big(\mathrm{abs}(I_{\mathrm{SR}}^{\nabla})\big)}{100}\Bigg)} \tag{3.14}$$

Here, $\big(I_{j\mathrm{R}}^{\nabla}\big)'$ represents any of the components appearing in equations (3.8)-(3.11), and $I_{j\mathrm{R}}^{\nabla}$, I_{HR}^{∇} and I_{SR}^{∇} their corresponding unnormalised values.

If the loss was scaled with a normalization factor $\max\big(\mathrm{abs}(\mathrm{concat}(I_{\mathrm{HR}}^{\nabla}, I_{\mathrm{SR}}^{\nabla}))\big)$ the generator would be rewarded for I_{SR}^{∇} containing one very large value, while when scaled according to the I_{HR}^{∇} values, it is penalized severely for having very large or very small gradient values relative to the scale of the actual gradients. However, this could also lead to exploding losses, so if the difference is very large it is scaled according to a fraction of the maximum I_{SR}^{∇} value.

As emphasized in 2.1.5, the vertical and the horizontal directions are not equivalent when it comes to atmospheric flow. The $\frac{\partial}{\partial z}$ derivatives contains physically different information from the $\frac{\partial}{\partial x}\frac{\partial}{\partial y}$ derivatives. They are also different in that we are super-resolving the wind field only in the horizontal direction, not vertically. Furthermore, with wind speed increasing quickly with elevation close to the ground, the $\frac{\partial}{\partial z}$ derivatives could dwarf the $\frac{\partial}{\partial x}\frac{\partial}{\partial y}$ derivatives if combined. To specifically focus the model on mass conservation, the divergence losses (3.11)-(3.10) are introduced. With the disparity between the vertical and the horizontal directions in mind we define a a horizontal divergence loss (3.11) in addition to the 3D divergence loss.

Comparing the HR and SR components rather than using physically informed penalties directly on the SR data brings many advantages. For example, the continuity equation tells us that the wind field (in the boussinesq approximation) should be divergence free. This can't hold perfectly for discrete simulated data, so one would have to find a proper level of error tolerance when penalising divergence. To complicate further, a reasonable tolerance level may vary spatially, for example if a bump in the terrain and the discreteness of the data causes local divergence. It is also unclear how one would penalise the generated data according to the momentum equation (2.29) without generating pressure or potential temperature.

Penalising the *differences* between the gradient of the SR and HR data solves all of these problems, automatically adjusting the loss according to how well we can expect the the equations to hold and what patterns in the generated data that should be penalized. It is also what makes it physically reasonable to penalise horizontal divergence even though mass isn't conserved in a horizontal slice, because it might hold approximately for large spatial parts of the data, and comparing tells us which ones. The downside of this technique is that physical imprecisions in the simulated HR data will transfer to the SR data, but, of course, the entire project is based on the assumption that the simulated data is sufficiently precise.

These losses combine to the total loss function for the generator:

$$L_G^{tot} = \eta_1 L_G^{pix} + \eta_2 L_G^{\nabla xy} + \eta_3 L_G^{\nabla z} + \eta_4 L_G^{div} + \eta_5 L_G^{div_{xy}} + \eta_6 L_G^{adversarial} \tag{3.15}$$

Like before, η_i are weight constants. The various losses gives us flexibility to test what we should emphasise most to teach the model to approximate the physics of atmospheric wind flow. η_6 is set to 0.005 (as in ESRGAN) initially, combinations of η_{1-5} are tested in **Experiment 2-2.5**.

3.4 Code and Hardware

The full code and its history can be found in can be found on Github at `https://github.com/jacobwulffwold/GAN_SR_wind_field`. The results are reproduceable using the repository. The code is modified from Larsen's [15] code, which again is an adaptation of Vesterkjær's [29] code, which is accessible at `https://github.com/eirikeve/esrdgan`. All parts of the code have been modified, but Vesterkjær's basic structure remains.

The listed experiments and training of the model was done with NVIDIA A100m40 GPUs on the IDUN cluster [34] at NTNU. With one GPU, training the model for 100k iterations takes a bit less than two days. The GPU memory usage strongly depends on the input data size and the batch size. With batch size 32 and 16x16x10 –> 64x64x10 training the model needs approximately 22GB GPU memory.

3.5 Experimental Setup

3.5.1 Standard Key Hyperparameters

Table 3.1 lists key standard hyperparametes that are used unless specified otherwise. An exhaustive list can be found in appendix A.

Table 3.1: Standard key hyperparameters and their meaning

Input		
Parameter	**Value**	**Meaning/Comments**
include_pressure	True	**G** input channel
include_z_channel	True	**G** input channel
included_z_layers	1-10	10 of the 11 layers closest to the ground (dropping the one furthest down)
interpolate_z	False	Interpolate as specified in Algorithm 1
include_altitude_channel	False	z_{alt} as **G** input channel

Data		
Parameter	**Value**	**Meaning/Comments**
enable_slicing	True	Slice 64x64 slices as specified in Chapter 3.1.3
scale	4	xy dimension scale difference between LR and SR, e.g. 16x16 $\rightarrow$ 64x64
data_augmentation_flipping	True	
data_augmentation_rotation	True	

Training				
Parameter		**Value**		**Meaning/Comments**
G/both	**D**	**G/both**	**D**	
learning_rate		1e-5	1e-5	
weight_init_scale		0.1	0.2	Smaller kaiming weight initialization
multistep_lr_steps		[10k, 30k, 50k, 70k, 100k]		Schedule for reducing learning rate during training
lr_gamma		0.5		Factor of reduction when lr is reduced
niter		90k		Number of training iterations
use_label_smoothing		True		Set HR labels to $0.9 + 0.1\frac{it}{niter}$ instead of 1.0 when training **D**
adversarial_loss_weight		0.005		η_6, as specified in Equation (3.15)

gradient_xy_loss_weight	1.0	η_2, as specified in Equation (3.15)
gradient_z_loss_weight	0.2	η_3, as specified in Equation (3.15)
xy_divergence_loss_weight	0.25	η_5, as specified in Equation (3.15)
divergence_loss_weight	0.25	η_4, as specified in Equation (3.15)
pixel_loss_weight	0.15	η_1, as specified in Equation (3.15)
D_G_train_ratio	1	How often **D** is trained relative to **G**
use_instance_noise	True	Add instance noise when training **D**, $\mathbf{D}(\mathbf{x}) \to \mathbf{D}(\mathbf{x}+\varepsilon)$ $\varepsilon_i \sim \mathcal{N}\left(0,\, 2.0\left(1-\frac{it}{niter}\right)\right)$
G_max_norm	1.0	Max norm for gradient clipping, functions as C in equation (2.35)

So unless otherwise specified, in addition to the wind field, pressure and the z-coordinates of the data are included as low resolution input channels for **G**, data is augmented by slicing, flipping and rotating as specified in Chapter 3.1.3, and the generator and the discriminator are trained in parallel for 90k iterations. The discriminator is intentionally confused initially by adding instance noise to the samples, and prevented from being too confident with one sided label smoothing (see Chapter 2.3.2).

3.5.2 Experiment 1: Input Channels and Handling Vertical Coordinates

With other parameters as specified in Chapter 3.5.1, in **Experiment 1**, the combinations in table 3.2 are tested with two different seeds.

Table 3.2: Combinations tested in **Experiment 1**. **name** is what the combination will be referenced to as, **interpolate** signals whether the data is interpolated as specified in Algorithm 1 or not, $\boldsymbol{z}$, $\boldsymbol{p}$, $\boldsymbol{z_{alt}}$ and **terrain** columns refers to whether these variables are included as input channels to the generator.

name	**interpolate**	$\boldsymbol{z}$	$\boldsymbol{p}$	$\boldsymbol{z_{alt}}$	**terrain**
only_wind	✗	✗	✗	✗	✗
z_channel	✗	✓	✗	✗	✗
p_channel	✗	✗	✓	✗	✗
p_z_channels	✗	✓	✓	✗	✗
z_{ground}_channels	✗	✗	✗	✓	✓
p_z_{ground}_channels	✗	✗	✓	✓	✓
only_wind_interp	✓	✗	✗	✗	✗
z_channel_interp	✓	✓	✗	✗	✗
p_channel_interp	✓	✗	✓	✗	✗
z_p_channels_interp	✓	✓	✓	✗	✗

As touched upon several times, the terrain shape and altitude significantly affect wind flow, but the irregular spacing of the vertical coordinates poses a problem for applying CNN's.

This experiment aims to find the best way to address this, either by giving the model input that contains information about the terrain, altitude and vertical spacing or by interpolating the data to get get a more CNN suited grid.

Note that since data augmentation is enabled, due to the bug mentioned in 3.1.3, many physically distorted data samples were used during training. This might disfavour the models that utilise extra terrain data somewhat, as it makes the terrain-wind relations slightly less accessible to the model. Test scores are not affected, as testing is only done on the non augmented wind field.

3.5.3 Experiment 2: Loss Functions

In response to the results of **Experiment 1**, a slight modification was done of the setup described in Chapter 3.5.1. The D_G_train_ratio variable is changed to 2 after 60k iterations and niter to 100k, so that **G** is trained for 60k+20k iterations and **D** is trained for 60k+40k iterations.

Otherwise the setup remains unchanged, meaning that the $p_z_$channels combination above will be tested, but now varying the loss function. In **Experiment 2** the combinations in table 3.3 are tested with two different seeds. The pixel loss and adversarial loss is kept constant, $\eta_1 = 0.15, \eta_6 = 0.005$. The std_cost combination is equivalent to the $p_z_$channels run in **Experiment 1**, so it will not be run again, but this means that it hasn't been changed as specified above, instead running 90k iterations for both **D** and **G**.

Table 3.3: Combinations tested in **Experiment 2**. **name** is what the combination will be referenced to as, $\boldsymbol{\eta_i}$s are the cost weighing constants defined in Equation (3.15)

name	η_2	η_3	η_4	η_5
only_pix_cost	0.0	0.0	0.0	0.0
grad_cost	1.0	0.2	0.0	0.0
div_cost	0.0	0.0	0.25	0.25
xy_cost	1.0	0.0	0.25	0.0
std_cost	1.0	0.2	0.25	0.25
large_grad_cost	5.0	1.0	0.25	0.25
large_div_cost	1.0	0.2	1.25	1.25
large_xy_cost	5.0	0.2	1.25	0.25

Experiment 2 aims to test four different hypothesis for what parts of the total loss (Equation (3.15)) are most useful to emphasise in training:

H21 Different parts of the wind gradient are about equally useful. If this is correct then grad_cost and large_grad_cost should perform comparatively well, as scaling up $L_G^{\nabla xy}$ and $L_G^{\nabla z}$ does not neglect any parts of the gradient.

H22 The divergence losses L_G^{div} and $L_G^{div_{xy}}$ are most useful. If this is correct then div_cost and large_div_cost should perform comparatively well.

H23 The losses that specifically target the $\frac{\partial}{\partial x}\frac{\partial}{\partial y}$ derivatives, $L_G^{\nabla xy}$ and $L_G^{div_{xy}}$, are most useful. If this is correct then xy_cost and large_xy_cost should perform comparatively well.

H24 The gradient based losses are strictly less useful than $L_G^{adversarial}$ and L_G^{pix}. If this is correct then only_pix_cost should outperform the other combinations.

3.5.4 Experiment 2.5: Loss Function Parameter search

After **Experiment 2** was somewhat inconclusive, a hyperparameter search was conducted in the following search space (based on results of **Experiment 2**

$$0.0 < \eta_1 < 1.0 \tag{3.16}$$
$$0.5 < \eta_2 < 32.0 \tag{3.17}$$
$$0.25 < \eta_3 < 16.0 \tag{3.18}$$
$$0.25 < \eta_4 < 16.0 \tag{3.19}$$
$$0.25 < \eta_5 < 16.0, \tag{3.20}$$

sampled on an uniformly on a logarithmic scale with base 2, except for η_1 which is sampled uniformly on a linear scale. The hyperparameter search is implemented using the ray tune framework [35], with the Optuna [36] search algorithm and an Asynchronous Successive Halving Algorithm (ASHA) [37] scheduler. The Optuna search algorithm implements a Bayesian tree search algorithm, it incorporates the results of previous samples when choosing configurations to test. The ASHA scheduler asynchronously stops poorly performing configurations, allowing tests of more combinations with the same computing power. Note that this biases the search towards early performers, assuming that strong initial performance indicates strong later performance. Ray tune is a flexible framework for implementing the parallel processing of the search.

The maximum training iterations is set to 35k and minimum iterations before stopping a run is set to 1.2k. The search is initialised with promising combinations based on results from **Experiment 2**.

Chapter 4

Results and Discussion of Experiments

4.1 Experiment 1: Input Channels and Handling Vertical Coordinates

4.1.1 Results of Experiment 1

Validation results during training in **Experiment 1**, as specified in Chapter 3.5.2, are displayed in Figure 4.1. Results from using the generator saved as it was on training iteration 80k on the test dataset are gathered in table 4.1.

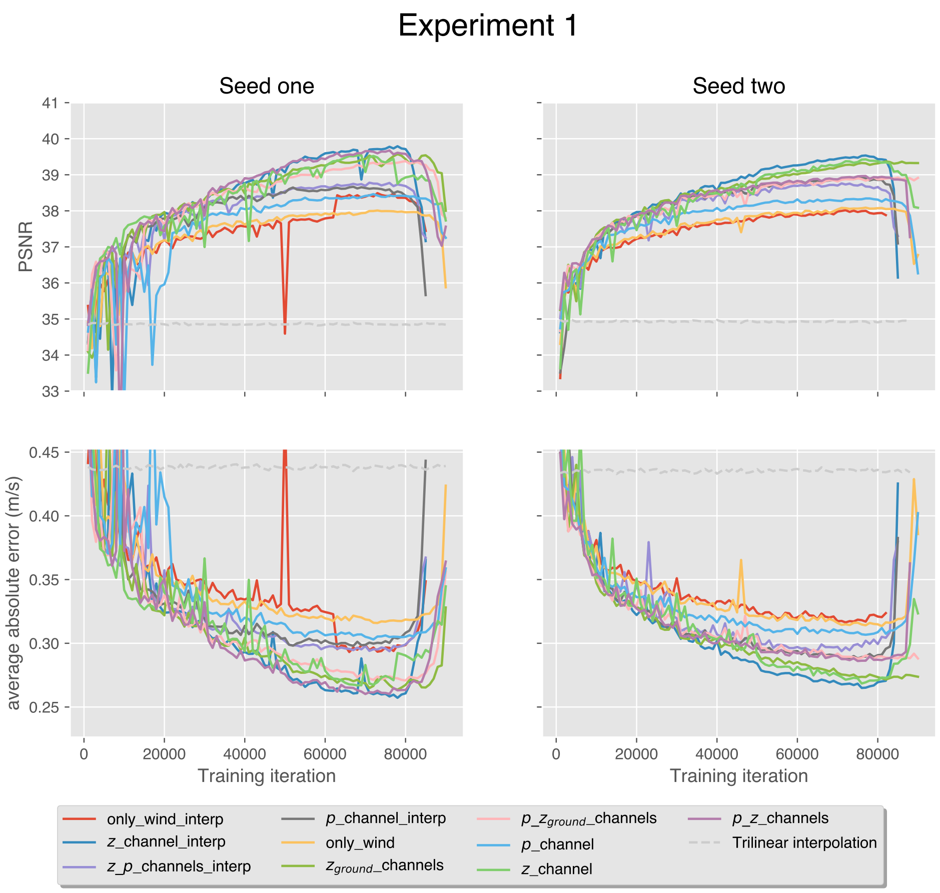


Figure 4.1: Experiment 1 validation results during training for two different seeds, coloured according to the combinations specified in Table 3.2. The top row shows validation peak signal to noise ratio, the bottom row shows the average absolute error of a wind component of the generated wind field. A dashed line for trilinear interpolation performance is included for comparison.

Table 4.1: Test results of **Experiment 1**. **name** references the combinations in table 3.2, **PSNR** is the average peak signal to noise ratio, **pix** represents the average absolute error of a wind component of the generated wind field and 1 and 2 refers to the two different seeds used for training. Since the models working with interpolated low resolution data are trained to produce interpolated high resolution data, the generated data is interpolated back to the original coordinates and evaluated against the original high resolution data during testing. Those results are to the right, and evaluation results against the interpolated high resolution data is to the left in the "interp" rows. Trilinear interpolation performance is included for comparison.

name	**PSNR1 (db)**	**PSNR2 (db)**	**pix1 (m/s)**	**pix2 (m/s)**
only_wind	39.35	39.32	0.283	0.285
z_channel	39.96	40.52	0.268	0.246
p_channel	39.59	39.65	0.278	0.276
p_z_channels	40.27	40.05	0.253	0.261
z_{ground}_channels	40.6	40.61	0.242	0.244
p_z_{ground}_channels	40.54	40.00	0.245	0.264
only_wind_interp	39.06 / 39.10	39.25 / 39.22	0.293 / 0.292	0.288 / 0.289
z_channel_interp	40.59 / 40.53	40.65 / 40.61	0.246 / 0.247	0.243 / 0.243
p_channel_interp	39.47 / 39.49	40.03 / 39.99	0.283 / 0.289	0.265 / 0.267
z_p_channels_interp	39.73 / 39.75	39.77 / 39.77	0.274 / 0.274	0.276 / 0.273
trilinear	36.53		0.377	
trilinear_interp	36.35 / 36.42		0.385 / 0.382	

4.1.2 Discussion of Experiment 1

A number of observations can be made from Table 4.1 and Figure 4.1:

1. The model performs much better than trilinear interpolation.
2. The PSNR and absolute error graphs in Figure 4.1 are mirror images, indicating that the error does not mainly consist of a few large mistakes.
3. The generator's performance degrade significantly towards the end of training, indicating that it overfits a non-optimal discriminator
4. Interpolation does not significantly degrade information quality, as the generated data interpolated back to the original coordinates performs almost as well as compared with the interpolated high resolution data.
5. Adding pressure as an input channel improves performance
6. Adding z-coordinates as an input channel improves performance
7. Interpolation slightly improves performance
8. Overall, including both pressure and z-coordinate input channels performs worse than only including z-coordinates
9. z_channel_interp and z_{ground}_channels are the best overall performers in the experiment.

Two (seeds) is a small sample size, so one cannot draw confident conclusions when the disparities in performance are small between models. Observations 1-6 are deemed relatively conclusive, while observations 7-9 are considered indicative. The intention of using z-coordinates rather than terrain values was that it seemed like a neat way of combining terrain, elevation above ground and vertical spacing into one consistent variable. Seeing that z_{ground}_channels, which splits height above ground and terrain, and z_channel_interp, which has the same vertical spacing everywhere performed well, it would have been beneficial to also have tested terrain_channel and terrain_channel_interp combinations, with only terrain as an extra input channel, for respectively interpolated and non-interpolated data.

The main conclusion of **Experiment 1** is that the convolutional model works well despite irregular coordinates, and performs much better when z-coordinates are included as an extra input channel together with the wind field.

4.2 Experiment 2: Loss Functions

4.2.1 Results of Experiment 2

Validation results during **Experiment 2**, for the combinations specified in Chapter 3.5.3 can be seen in Figure 4.2. Results from using the generator saved on training iteration 80k for std_cost and 90k for the rest of the combinations on the test set are gathered in Table 4.2. As explained in Chapter 3.5.3, 90k for the other values means 75k training iterations for the generator, so the std_cost model applied to the test set has 5k more training iterations than the others in the table.

Table 4.2: Test results of **Experiment 2**. **name** references the combinations in table 3.3, **PSNR** is the average peak signal to noise ratio, **pix** represents the average absolute error of a wind component of the generated wind field and 1 and 2 refers to the two different seeds used for training.

name	**PSNR1 (db)**	**PSNR2 (db)**	**pix1 (m/s)**	**pix2 (m/s)**
only_pix_cost	38.00	37.37	0.335	0.344
grad_cost	39.67	39.73	0.276	0.273
div_cost	39.65	36.62	0.276	0.378
xy_cost	38.84	31.73	0.309	0.519
std_cost	40.27	40.05	0.253	0.261
large_grad_cost	41.05	40.94	0.234	0.239
large_div_cost	41.42	41.35	0.225	0.227
large_xy_cost	41.55	40.72	0.223	0.248

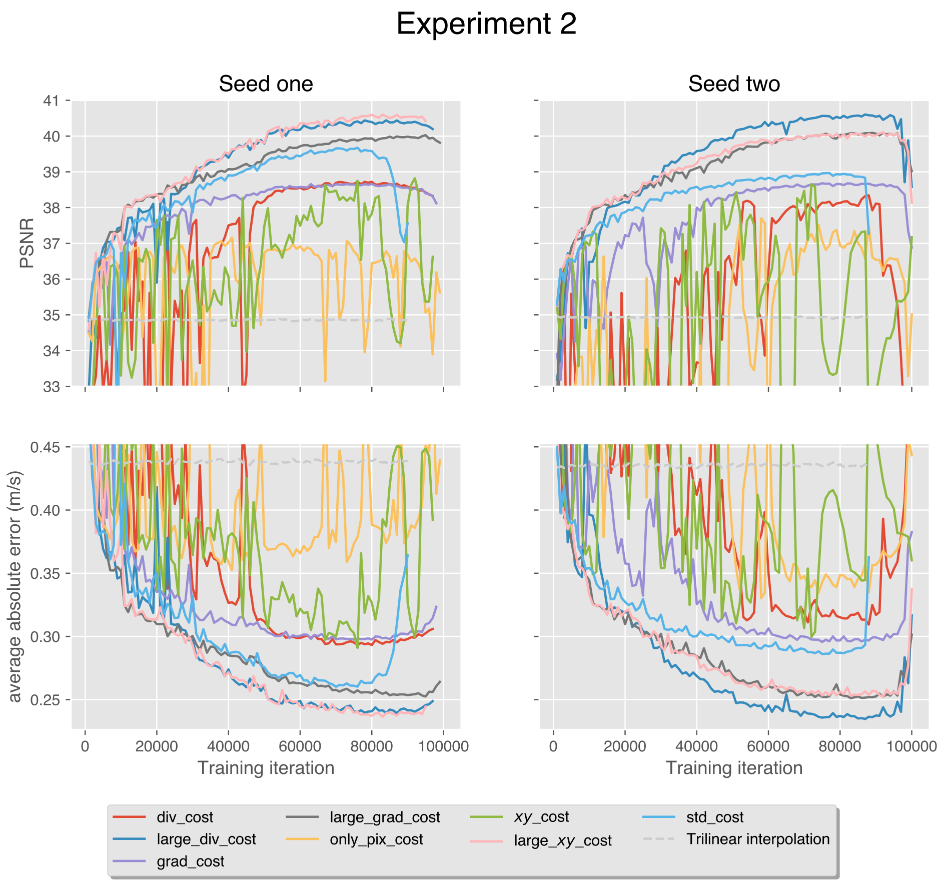


Figure 4.2: Experiment 2 validation results during training for two different seeds, coloured according to the combinations specified in Table 3.3. The top row shows validation peak signal to noise ratio, the bottom row shows the average absolute error of a wind component of the generated wind field. A dashed line for trilinear interpolation performance is included for comparison.

4.2.2 Discussion of Experiment 2

From Figure 4.2 and Table 3.3 we see that the gradient-based losses improve performance significantly with the current setup. However, the results are not very conclusive for the hypotheses introduced in Chapter 3.5.3, except for **H24**, which is falsified, at least with the current setup. As the runs with the largest losses are the best performers, **Experiment 2** does not tell us if we are anywhere close to the actual best values, which might be much bigger. The results point slightly in favour of emphasising $L_G^{div_{xy}}$ and deemphasising $L_G^{\nabla z}$, but this is considered weak, especially considering that there seems to be a key point in training around iteration 25k, at which "large" runs split towards a better or worse trajectory. The results could also indicate too low learning rate for the generator, if increasing the absolute value of the loss improves performance.

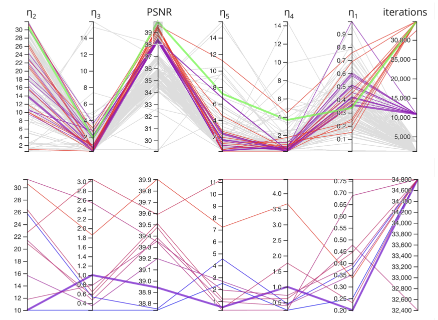
As in **Experiment 1**, the models' performance degrade towards the end of training, so the increase in training the discriminator was not sufficient to address this problem, even perhaps making it worse. However, we see some of the runs start to oscillate, indicating that the dip in performance towards the end might be the start of an oscillating pattern.

Due to the ambiguity of these results, and the large space of possible combinations, a hyperparameter search was conducted, **Experiment 2.5**.

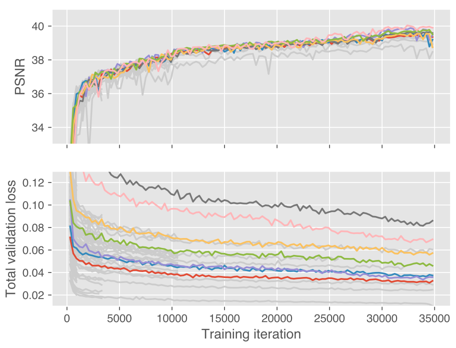
4.3 Experiment 2.5: Loss Function Parameter Search

4.3.1 Results of Experiment 2.5

The results from the parameter search can be seen in Figure 4.3, and the parameters of the five best performing runs are displayed in Table 4.3. The decomposed validation loss during training of the best performing loss function, hereby named L_G^*, is displayed in Figure 4.4. The norm of the gradient before clipping during training is displayed in Figure 4.5.



(a) Parameter combinations tested in the search and their performance. Plot generated by ray.tune.



(b) PSNR and total validation loss during training

Figure 4.3: Results of **Experiment 2.5**. In the top left corner the scores of all tested combinations and their parameters are displayed, with the best performing combinations highlighted. The η_is are the loss weighing constants defined in (3.15), "iterations" is what iteration the run was stopped at, PSNR is the validation PSNR score after that iteration. In the bottom left corner we see the same table for only the the best performers with the parameter axes scaled according to their spanned parameter range. To the right we see validation results during training, with the best performers coloured. In the top right corner we see the validation PSNR score, in the bottom right corner we see the total validation loss.

η_1	η_2	η_3	η_4	η_5	**PSNR (db)**	**pix (m/s**
0.336	30.6	1.86	7.21	3.66	39.9	0.259
0.757	22.6	3.05	11.2	4.45	39.6	0.265
0.446	26.4	0.643	1.18	0.62	39.5	0.271
0.231	20.9	0.565	0.398	1.76	39.5	0.277
0.477	21.4	0.464	0.965	0.391	39.4	0.271

Table 4.3: Best performing combinations in **Experiment 2.5**. The η_is are the loss weighing constants defined in (3.15), **PSNR** is the validation PSNR score after the final training iteration and **pix** is the average absolute error of the generated wind field.

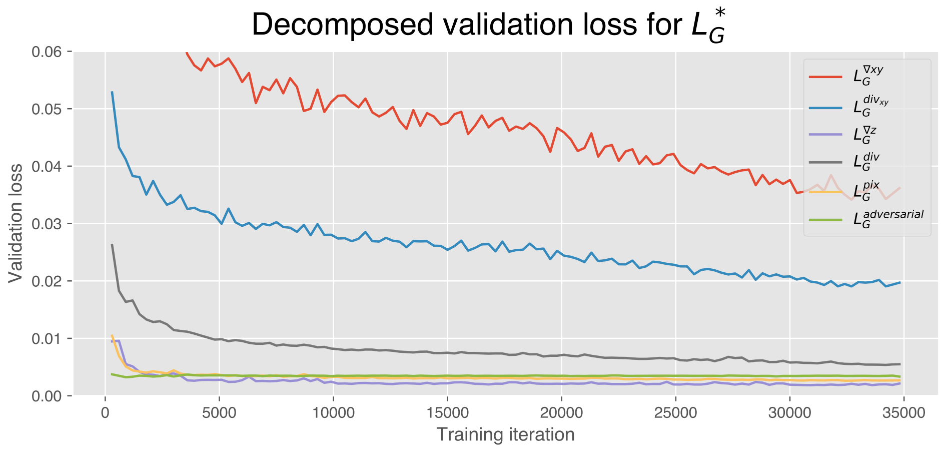


Figure 4.4: Decomposed validation loss for the best performing loss function in the parameter search L_G^*. The losses in the plot are as the ones defined in Chapter 3.3.

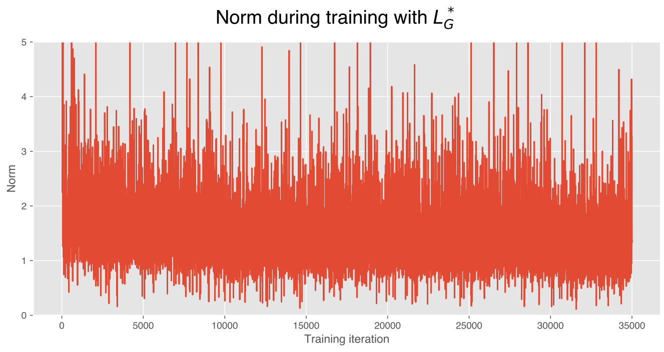


Figure 4.5: The unclipped norm of the gradient during training with L_G^*. With the current setup, all values over 1 are clipped. (Equation (2.35))

4.3.2 Discussion of Experiment 2.5

Some key observations from Figure 4.3:

1. Performance still seems to somewhat favour large absolute loss.
2. The samples (and best performers) are heavily biased towards large η_2 (scaling up $L_G^{\nabla xy}$) and low η_3 (scaling down $L_G^{\nabla z}$)
3. Performance seems to favour larger η_4 ($L_G^{div_{xy}}$) than η_5 (L_G^{div})
4. Pixel loss for the best performers varies significantly

Figure 4.5 shows us that, with the increased absolute size of the loss function, the norm of the gradient was so large that it was clipped according to equation (2.35) for almost all training iterations. So most optimisation steps during training were the same length, regardless of the loss landscape, which is not optimal for performance. However, it might actually have been beneficial for the parameter search, because when each model has a set optimisation step length only the relative weighting of the different losses matter, thereby neatly allowing us to compare only the relative scaling of the losses. Looking at the total validation losses in Figure 4.3b reveals that the best performing model has the second largest absolute loss, so correspondingly scaling down the values in Figure 4.5 we see that most of the runs probably were not as often clipped as L_G^*. All in all, it is unclear whether this effect in sum was positive or negative for the parameter search. But we've learned that the learning rate and clipping needs to be adjusted.

In total, **Experiment 2.5** favours hypothesis **H23**, namely that the gradient-based losses focused on the $\frac{\partial}{\partial x}\frac{\partial}{\partial y}$ derivatives are most useful. Figure 4.4 shows that the best performing loss function L_G^* is heavily dominated by $L_G^{\nabla xy}$, $L_G^{div_{xy}}$ and L_G^{div}, in that order. The best scaling for the pixel and adversarial losses seem underdetermined by the experiment.

Chapter 5

Evaluation of the Best Performing Model

5.1 Hyperparameters of the Best Performing Model

Based on the results of **Experiment 1**, **Experiment 2**, **Experiment 2.5** and further testing, a number of changes were made to the hyperparameters of the model. Most significantly, adversarial learning has been dropped. Changing to alternating training between **D** and **G**, experimenting with turning on or off instance noise and label smoothing, adjusting the adversarial loss weight and initiating with a generator pre-trained on content loss gave at best a small increase in performance over pure content loss, on a scale in which it could simply be due to a learning rate reset. Therefore, we will instead look at the best generator without adversarial learning, named G_{best}. Modified generator parameters are listed in Table 5.1. Otherwise, the generator setup is unchanged from the full list of hyperparameters in Appendix A.

Table 5.1: Modified hyperparameters of G_{best} from the standard setup.

Input		
Parameter	**Value**	**Meaning/Comments**
include_pressure	False	**G** input channel
interpolate_z	True	Interpolate as specified in Algorithm 1

Training		
Parameter	**Value**	**Meaning/Comments**
learning_rate	8e-5	
niter	150k	Number of training iterations
adversarial_loss_weight	0	η_6, as specified in Equation (3.15)

gradient_xy_loss_weight	3.064	η_2, as specified in Equation (3.15)
gradient_z_loss_weight	0	η_3, as specified in Equation (3.15)
xy_divergence_loss_weight	0.721	η_5, as specified in Equation (3.15)
divergence_loss_weight	0.366	η_4, as specified in Equation (3.15)
pixel_loss_weight	0.136	η_1, as specified in Equation (3.15)
pixel_loss_weight_pre-train	0.136	η_1, as specified in Equation (3.15), used in pre-training
G_max_norm	∞	Max norm for gradient clipping, functions as C in equation (2.35)

As seen in Table 5.1, the coefficients in L_G^* has been scaled down by a factor 10 and $L_G^{\nabla z}$ has been dropped completely.

$$L_{G_{best}}^{tot} = \eta_1 L_G^{pix} + \eta_2 L_G^{\nabla xy} + \eta_4 L_G^{div} + \eta_5 L_G^{div_{xy}} \tag{5.1}$$

with η_i values specified in Table 5.1. G_{best} was first trained for 100k iterations with pixel loss $\eta_1 = 0.034$ then trained for 150k iterations with $\eta_1 = 0.136$. Restarting with four times as high pixel loss improved not only the pixel wise error but also caused lower gradient-based losses. Also, the learning rate has been increased by a factor 8, and gradient clipping has been dropped.

The results in this chapter are also subject to the bug in data augmentation, as there was not time to fully redo training when the bug was discovered. Ongoing runs without the bug indicate that performance improves slightly with correct data augmentation.

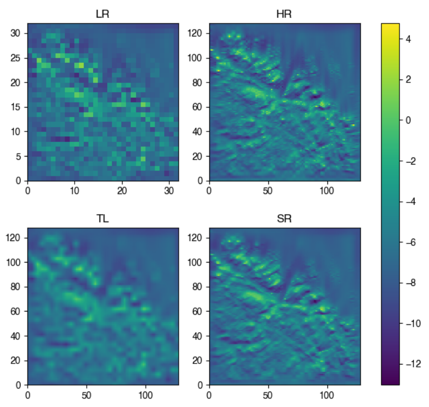
5.2 Evaluation of the Model

5.2.1 Metrics and Overall Performance

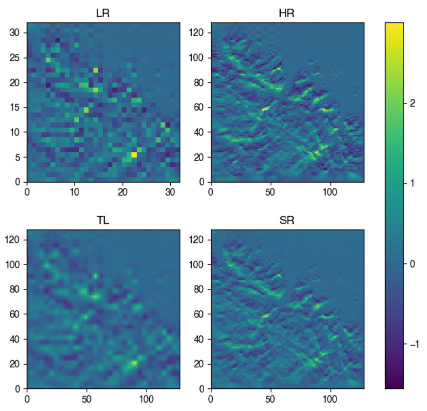
The average test scores of G_{best} are compared against trilinear interpolation in Table 5.2. In Figure 5.1 we see the second lowest horizontal slice, while in Figure 5.2 we see the second highest horizontal slice, of the wind field in low resolution (LR), high resolution (HR), interpolated (TL) and super-resolved by G_{best} (SR).

Table 5.2: Test result of G_{best}. **PSNR** is the average peak signal to noise ratio for a single sample in the test set, **pix** represents the average absolute error of a wind component of the generated wind field, **pix-vector** means the average length of the error wind vector, **pix-vector relative** is average value of the **pix-vector** divided by the average wind speed of the sample. Though being interpolated, the generated wind field is not interpolated back and compared to original data as in Table 4.1, as this did not affect the results significantly. Trilinear interpolation performance is included for comparison.

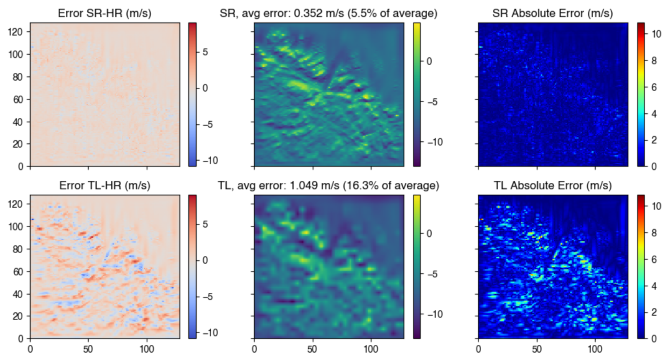
name	**PSNR (db)**	**pix (m/s)**	**pix-vector (m/s)**	**pix-vector relative**
G_{best}	47.14	0.116	0.24	6.12%
trilinear_interp	36.35	0.385	0.80	18.5%



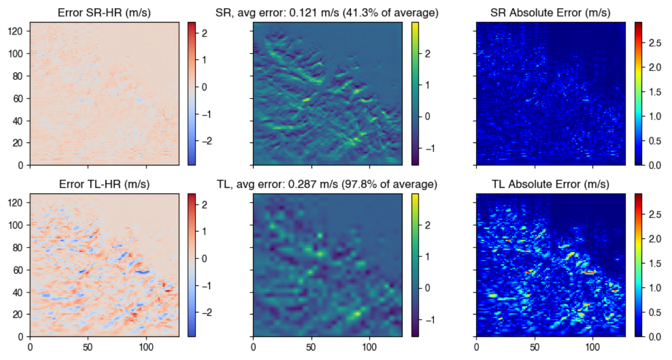
(a) Wind velocity along the x-axis



(b) Wind velocity along the z-axis

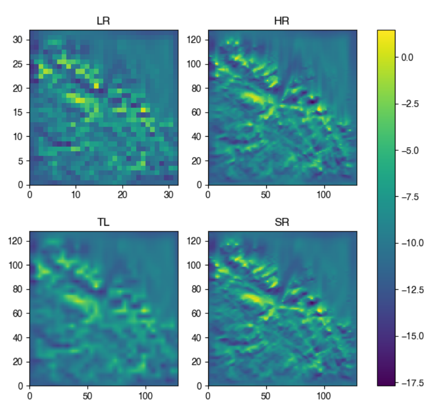


(c) Error for the wind component pointing along the x-axis

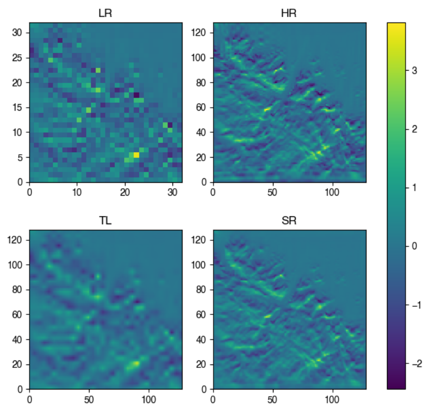


(d) Error for the wind component pointing along the z-axis

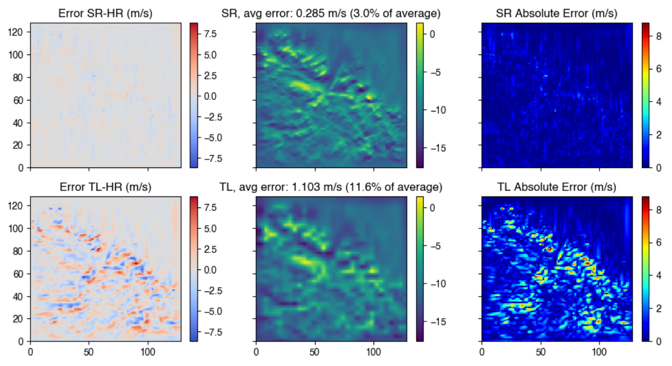
Figure 5.1: Comparison of the second lowest horizontal 2D slice of the wind field for a randomly selected wind field. LR means low resolution, the input of the model, TL means trilinear interpolation of the LR data, SR means the super-resolved wind field generated by the trained network and HR means the true high resolution wind field. "Avg error" refers to average absolute error for the displayed wind component in the displayed slice, the "% of average" means this value divided by the average of value of that wind component in that slice for the HR wind field.



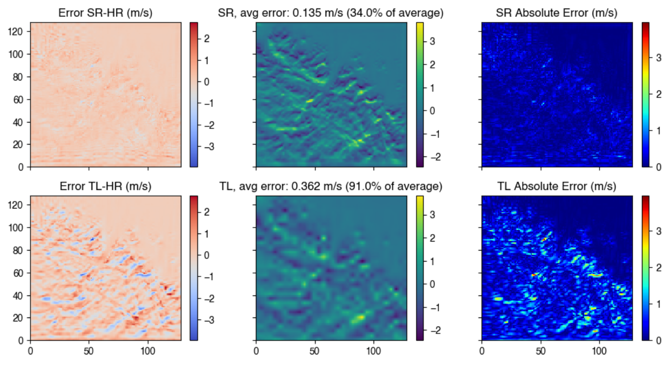
(a) Wind velocity along the x-axis



(b) Wind velocity along the z-axis



(c) Error for wind velocity along the x-axis



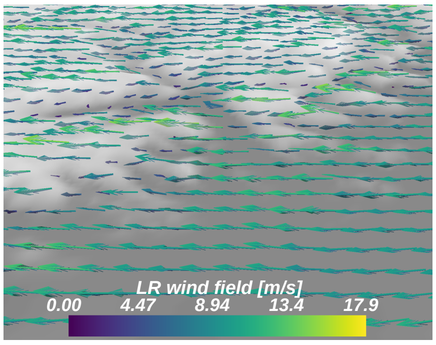
(d) Error for wind velocity along the z-axis

Figure 5.2: Comparison of the second highest horizontal 2D slice of the wind field for a randomly selected wind field. LR means low resolution, the input of the model, TL means trilinear interpolation of the LR data, SR means the super-resolved wind field generated by the trained network and HR means the true high resolution wind field. "Avg error" refers to average absolute error for the displayed wind component in the displayed slice, the "% of average" means this value divided by the average of value of that wind component in that slice for the HR wind field.

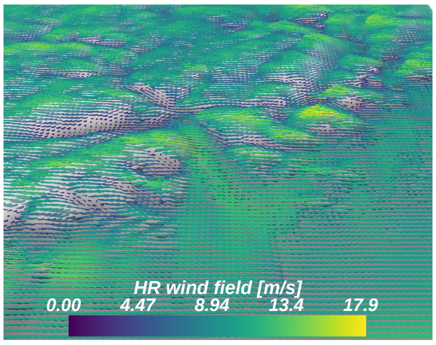
Next we will look more in detail at the generated 3D wind flow, to see if the physics informed loss functions has helped the model produce physically reasonable results.

5.2.2 Generated 3D Wind Flow and Turbulence Modelling

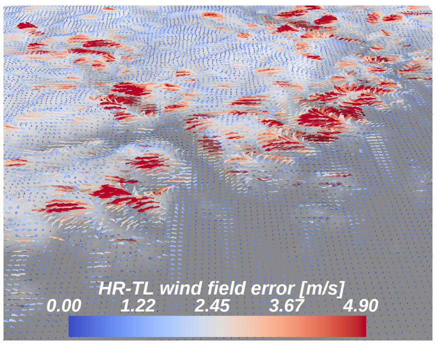
We now compare the entire 3D fields as we did for some selected 2D slices and wind components above. First we will look at a sample with relatively large average wind speed at 8.2 m/s. For this wind field the average length of the error wind vector of the SR field is 0.41 m/s (5% of average) and the average absolute error of the TL field is 1.34 m/s (16% of average). In figure Figure 5.3 we see how the generated wind fields adapts to the terrain, comparing for LR, HR, TL and SR data, as before.



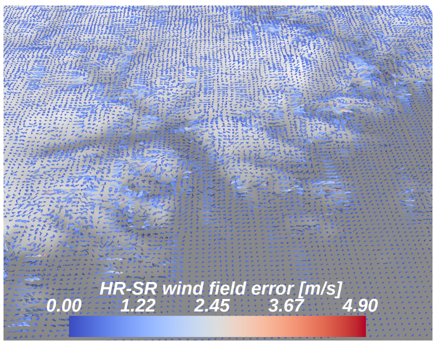
(a) LR wind field



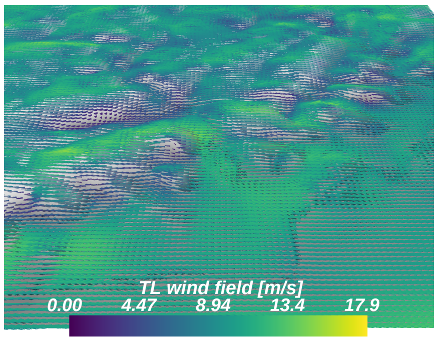
(b) HR wind field



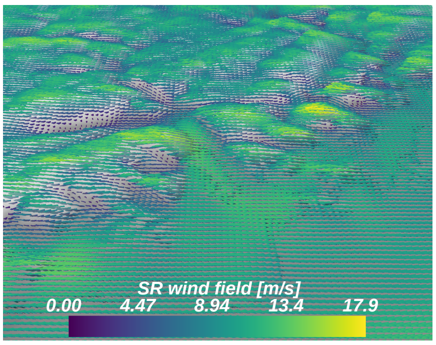
(c) TL error, HR-TL



(d) SR error, HR-SR



(e) TL wind field

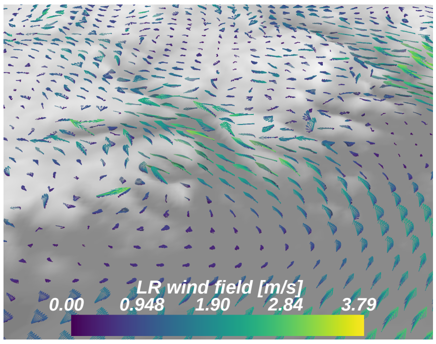


(f) SR wind field

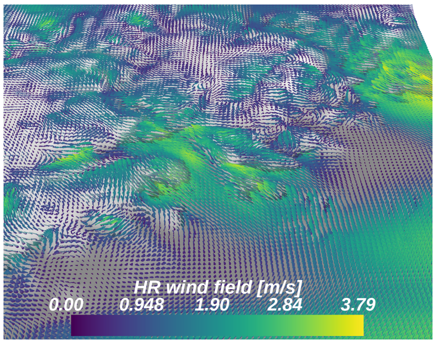
Figure 5.3: Comparison of LR, HR, TL and SR wind fields from a randomly selected sample. LR means low resolution, the input of the model, TL means trilinear interpolation of the LR data, SR means the super-resolved wind field generated by the trained network and HR means the true high resolution wind field.

Figure 5.3 demonstrates that the SR wind field adapts much more crisply to the terrain than the interpolated data. Looking more closely at Figure 5.3c, we can see that the interpolation error follows regular terrain following patterns, with TL interpolation systematically misjudging regions of quickly changing terrain. Meanwhile Figure 5.3d shows a much more chaotic pattern, indicating that the model doesn't have obvious weak spots, but makes calibrated best guesses when inferring the wind field between data points.

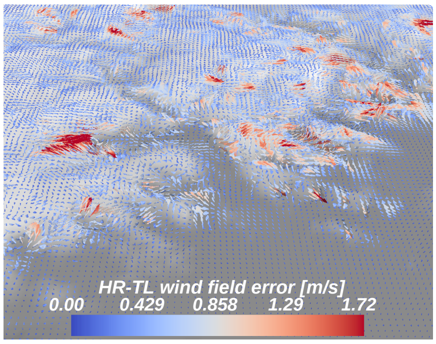
So the model seems to have learned general terrain adaptation. The model, however, is trained to minimise absolute error in gradients and component-wise wind speed, which means that for any batch during training, the large wind fields will dominate the loss function. We should therefore check how it fares with less powerful winds, so we look at a sample with average wind speed 1.04 m/s in Figure 5.4.



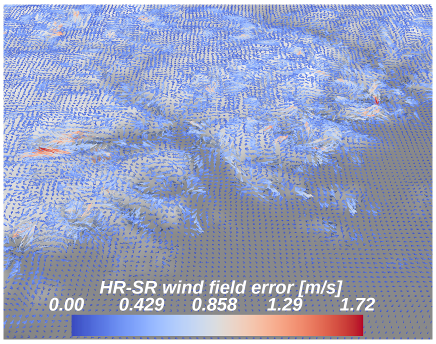
(a) LR wind field



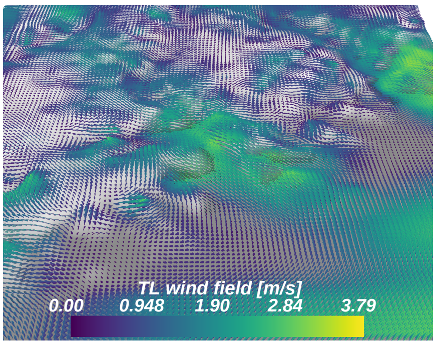
(b) HR wind field



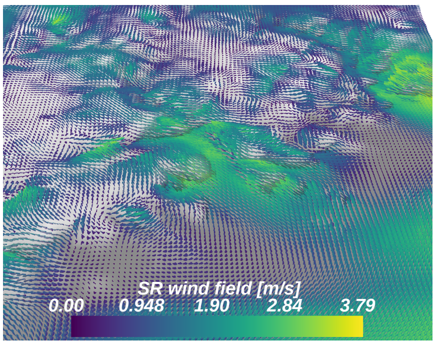
(c) TL error, HR-TL



(d) SR error, HR-SR



(e) TL wind field



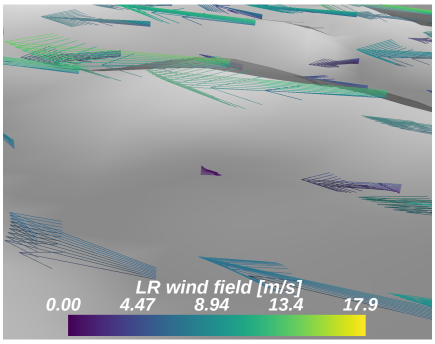
(f) SR wind field

Figure 5.4: Comparison of LR, HR, TL and SR wind fields from a randomly selected sample. LR means low resolution, the input of the model, TL means trilinear interpolation of the LR data, SR means the super-resolved wind field generated by the trained network and HR means the true high resolution wind field.

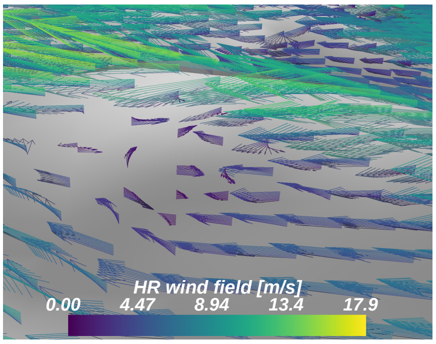
For this area, the average absolute error is 0.19 m/s (18% of average speed) for SR and 0.31 m/s (30% of average speed). We also clearly see that the relative error is higher in Figure 5.4, since even with the colour scale in Figures 5.4d going up to 160% the average wind speed it we see more error in Figure 5.4d than in figure 5.3d, where the colour scale maxed out at 60% of the average wind speed. Comparing 5.4d and 5.4e more closely, we can actually see SR underperforming TL in quite large areas of low wind speed. We conclude that the model's emphasis on large absolute error makes it perform worse on very weak wind fields. This is not necessarily a bad thing, in wind power it is the absolute, not the relative, size of the error that matters for estimating power production.

Figure 5.4 also demonstrates a problem with the vertical cutoff of the training data. In the figure we see large areas in which the wind seems to stop completely. What is actually happening is that the wind is pushed upwards by the terrain. With terrain following coordinates only up to 40m above ground level and weak winds, the data doesn't cover the wind continuing higher up, instead making it seem like the wind only blows at the top of hills. Given that we are trying to push the model to learn traits like mass conservation this is not ideal. Extending the input data to higher altitude above ground up should therefore improve performance.

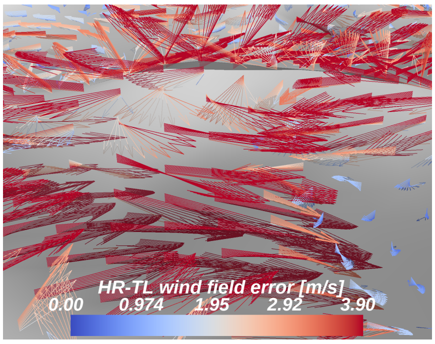
Next, let us see how the model deals with the harder parts of modelling wind flow. Figure 5.5 zooms in on a turbulent region of the HR, LR, TL and SR wind fields.



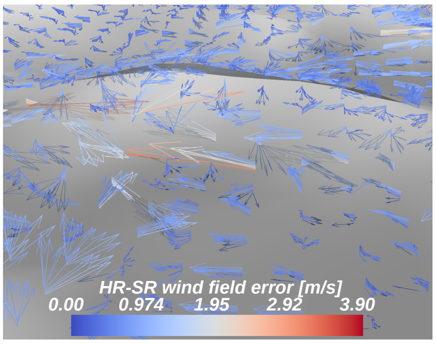
(a) LR wind field



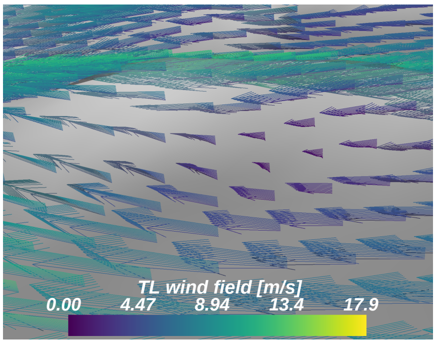
(b) HR wind field



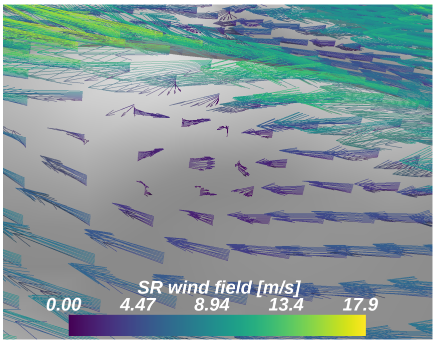
(c) TL error, HR-TL



(d) SR error, HR-SR



(e) TL wind field



(f) SR wind field

Figure 5.5: Comparison of LR, HR, TL and SR wind fields in a turbulent region. LR means low resolution, the input of the model, TL means trilinear interpolation of the LR data, SR means the super-resolved wind field generated by the trained network and HR means the true high resolution wind field.

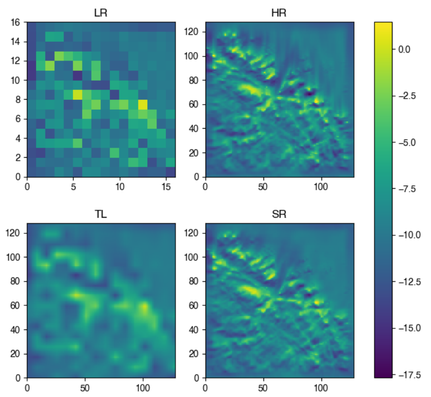
While interpolation almost completely smoothes out the turbulence area, the super-resolved wind field is remarkably accurate. With only the datapoints in 5.5a it recreates the swirling pattern of 5.5b, albeit slightly more regular than the actual data. This might be a reason why it was hard to improve performance with adversarial learning. For chaotic regions, the best guess for the model is probably a smoothed out swirling pattern like we see above. However when the HR data is sufficiently chaotic, the discriminator gets an easy way to tell whether the data is real or fake by targeting the chaotic areas. In order to fool the discriminator the generator might have to generate more chaotic patterns, but the more random the pattern, the more likely it is to be wrong, so instead the content loss ensures that the generator stays in line.

5.2.3 Changing Scaling and Slicing

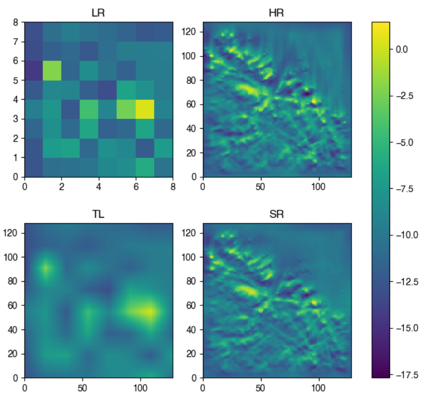
Having a functioning setup we can see how well it performs for higher scale gaps than 4x4 resolution increase, and how disabling the data slicing affects the results. Table 5.3 shows the results of scaling the model with a 4x4, 8x8 and 16x16 increase with and without data slicing. In Figure 5.6 we compare the results of 8x8 and 16x16 resolution increase when not slicing the wind field, for the same horizontal layer as in Figure 5.2.

Table 5.3: Test results varying scale of resolution increase and whether data slicing is used in training. **PSNR** is the average peak signal to noise ratio for a single sample in the test set, **pix** represents the average absolute error a wind component of the generated wind field, **pix-vector** means the average length of the error wind speed vector, **pix-vector relative** is average value of **pix-vector** divided by the average wind speed of the sample. Though being interpolated, the generated wind field is not interpolated back and compared to original data as in Table 4.1, as this did not affect the results significantly. Trilinear interpolation performance is included for comparison.

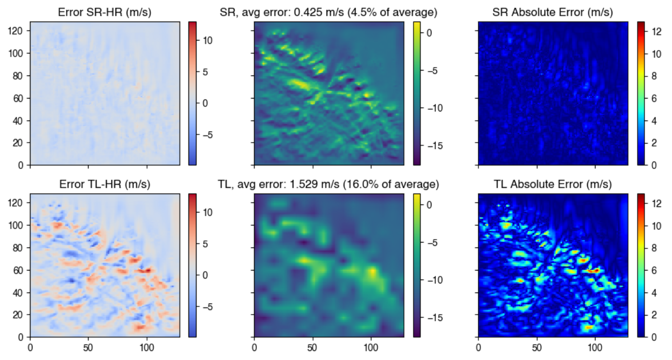
name	**PSNR (db)**	**pix (m/s)**	**pix-vector (m/s)**	**pix-vector relative**
G_{best}	47.14	0.116	0.24	6.1%
G_{best} no slicing (4x4)	49.27	0.090	0.19	5.0%
4x4 trilinear	36.35	0.385	0.8	18.5%
G_{8x8}	41.95	0.204	0.43	11.3%
G_{8x8} no slicing	44.25	0.159	0.33	9.1%
8x8 trilinear	33.86	0.528	1.12	25.8%
G_{16x16}	34.2	0.502	1.11	26.68%
G_{16x16} no slicing	41.57	0.216	0.46	12.8%
16x16 trilinear	32.77	0.609	1.29	30.6%



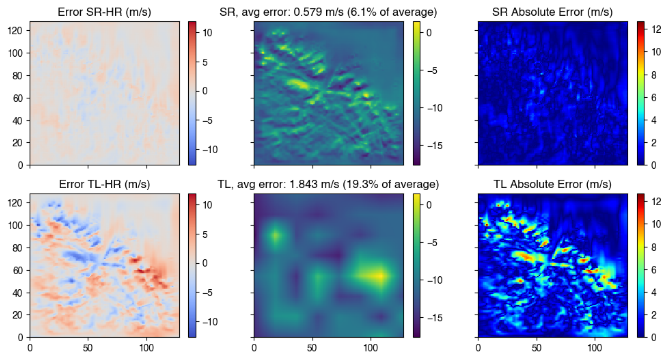
(a) Wind velocity for 8x8 resolution increase



(b) Wind velocity for 16x16 resolution increase



(c) Error for 8x8 resolution increase



(d) Error for 16x16 resolution increase

Figure 5.6: Comparison of 8x8 and 16x16 resolution increase for models trained without slicing on wind velocity along the x-axis for the second highest horizontal 2D slice of the wind field for the same wind field as in Figure 5.2. LR means low resolution, the input of the model, TL means trilinear interpolation of the LR data, SR means the super-resolved wind field generated by the trained network and HR means the true high resolution wind field. "Avg error" refers to average absolute error for the displayed wind component in the displayed slice, the "% of average" means this value divided by the average of value of that wind component in that slice for the HR wind field.

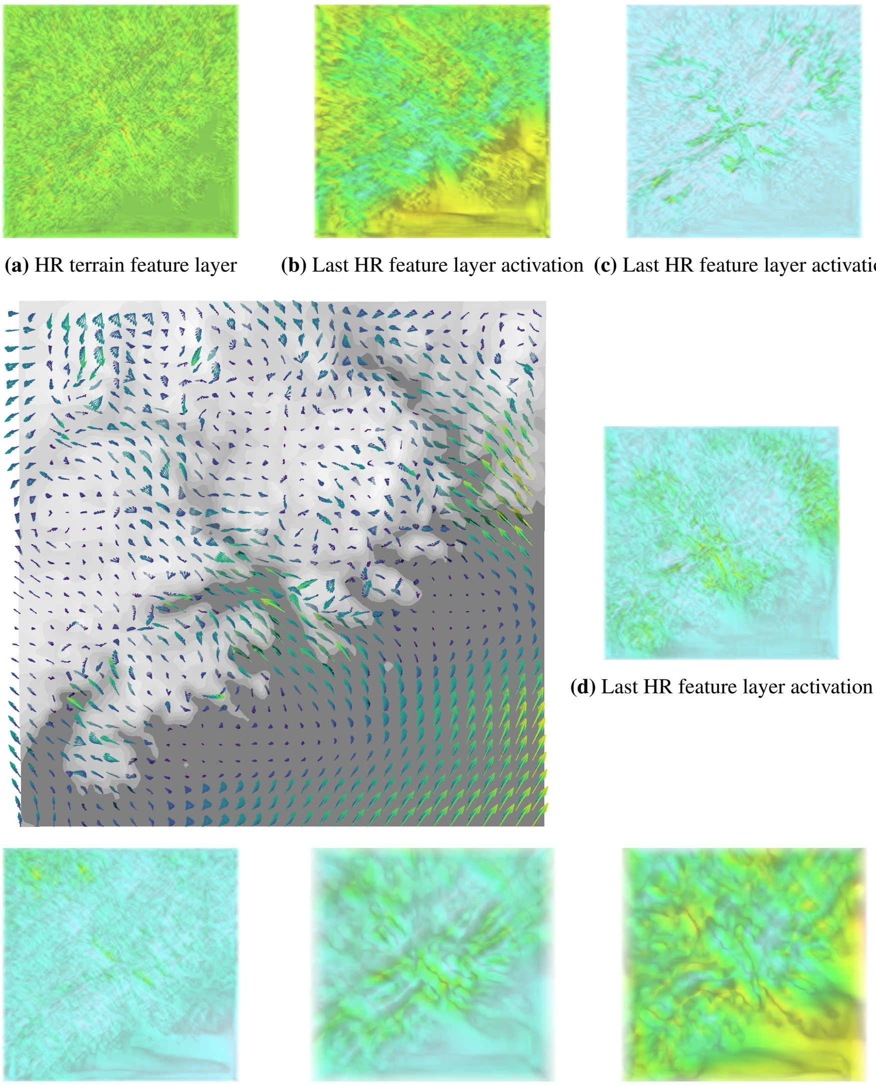
A number of points can be made from these results:

- The model is flexible, it can be trained for multiple resolution increase scales.
- The model is obviously gaining a lot from having the high resolution terrain information available, as it can still produce terrain-following wind patterns even for a 16x16 resolution increase.
- The model performs much better when slicing the dataset is turned off

Given that the model is relying so much on the high-resolution terrain information, the key question becomes to what degree the model is overfitted to the local terrain. Until the model is tested on terrain not present in the training data we remain uncertain. Turning off data slicing means allowing the model to overfit the local terrain even more, having the same datapoint locations in the same terrain for each sample. This poses an opportunity for getting better results by intentionally overfitting the model to a specific area, for example around a wind farm, to better monitor and operate the farm. If this is the goal then further gains can probably be made by turning off rotation and flipping data augmentation as well.

5.3 Inside the Model

To see what the model actually focuses on, we can look at the output feature maps of hidden layers of the model. Being fully convolutional, the model maintains the spatial orientation, so we can map the feature maps directly onto the 3D wind field. Figure 5.7 shows the low reaolution low velocity wind field shown in Figure 5.4, and some selected feature maps generated by G_{best} when given that input. In the figure we see one terrain feature map, four samples from the last feature layer after activation and two samples from the last layer before upscaling (see Figure 3.6 for context).



(a) HR terrain feature layer **(b)** Last HR feature layer activation **(c)** Last HR feature layer activation

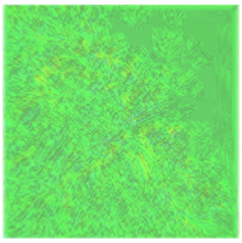
(d) Last HR feature layer activation

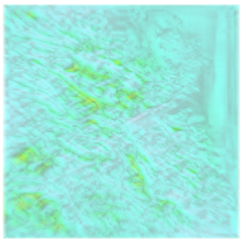
(e) Last HR feature layer activation **(f)** Last LR feature layer **(g)** Last LR feature layer

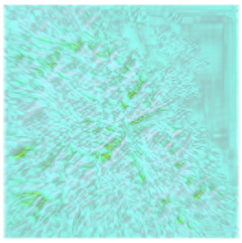
Figure 5.7: Feature maps of the trained model. In the centre we see the low resolution wind field over the terrain, the input of the model, and around we see feature maps generated by feeding this input to the model. In the top left corner we see a terrain feature layer, in the bottom right corner we see to feature maps from the last layer before upscaling, the rest are from the last feature layer of the model after activation.

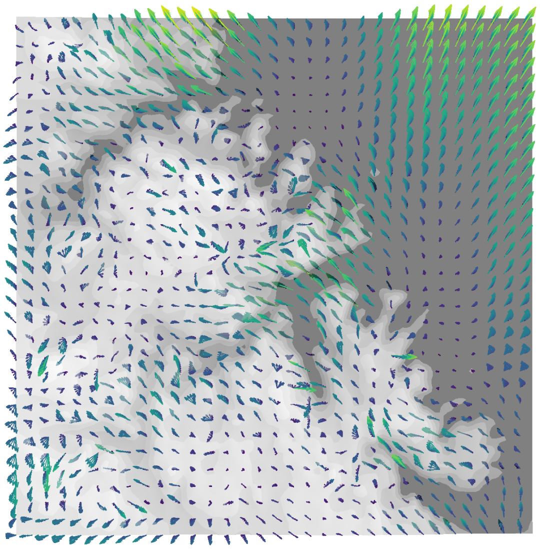
Not surprisingly, we can see the model emphasising specific wind directions and terrain shapes. For example, the features in Figure 5.7d seem to be activated by wind speed in the negative x direction and the features in Figure 5.7e seem to be activated by wind speed in the negative y direction. The features in Figure 5.7c seems to be activated in the areas with sharp terrain changes across an axis roughly parallel to the $x = y$ diagonal, which interestingly also is parallel to the coastline and incoming wind speed. In fact, many of the feature maps seem to utilise a grid along the diagonals of of the figures, indicating that the model has found these axes to be the most useful horizontal decomposition of the wind and terrain data. This also makes sense given that the rotation and flipping done when loading data keeps the coastline pointing along one of the diagonals, so that even if the image is rotated or flipped it will maintain one axis parallel to the coast line and one axis perpendicular to the coastline. In that case, it is fitted to the current terrain.

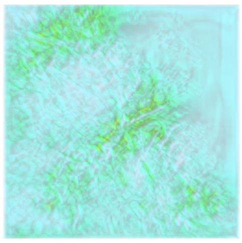
As mentioned, there was an error in the code causing the rotation in training to apply only to the spatial points, not the orientation of the vectors. To understand how the model treats such rotation, and to test our guesses of the meaning of feature layers in Figure 5.7, Figure 5.8 displays the same feature layers for the same input wind field, but with the spatial points (but not orientation of the vectors!) rotated 90 degrees.

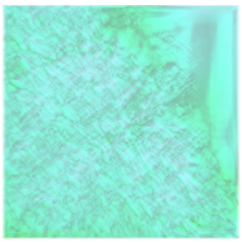

(a) HR terrain feature layer

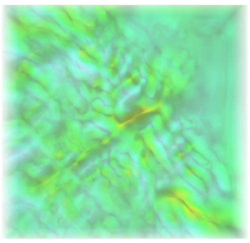

(b) Last HR feature layer activation


(c) Last HR feature layer activation




(d) Last HR feature layer activation


(e) Last HR feature layer activation


(f) Last LR feature layer

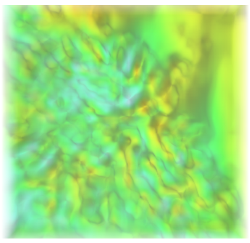

(g) Last LR feature layer

Figure 5.8: Feature maps of the trained model produced by inputting the wind field in Figure 5.7 rotated 90 degrees withoout changing the orientation of the vectors. In the centre we see the low resolution wind field over the terrain, the input of the model, and around we see feature maps generated by feeding this input to the model. In the top left corner we see a terrain feature layer, in the bottom right corner we see to feature maps from the last layer before upscaling, the rest are from the last feature layer of the model after activation.

First, we can see clearly how, since the vector orientation is not changed, the new wind pattern does not match the terrain as it did above. The wind crosses ridges with no effect in some places, and gets pushed by non-existing ridges in other. For the rotated field we can see that our guesses about the maps in Figure 5.8c and Figure 5.7d seem to be reasonable, but many of the other maps have changed completely. This is a good sign with regards to overfitting, as with the vectors remaining the same the model could have learned to just use relations between the vectors and their exact local shape of the terrain. In that case, all the feature maps would be the same, only rotated 90 degrees. That many of them change significantly indicates that the model is sensitive to the orientation of the vectors relative to the terrain, and has found (rotated) actual meaningful connections between the terrain and wind flow direction.

Chapter 6

Conclusion and Further Work

In this thesis I have described a 3D convolutional GAN network for super-resolving the near surface 3D wind field with a 4x4 horizontal resolution increase. The model is inspired by the ESRGAN [28] image super-resolution architecture, modified to work with 3D wind flow. A feature dropout layer was added before the last convolutional layer in the generator and after the feature extractor part of the discriminator. Most importantly, a terrain feature extractor was added to the generator, so that it can combine the features extracted from the low resolution data with high resolution terrain information.

Atmospheric flow in complex terrain poses a challenge for 3D CNNs. Information wise one wants a non-regular grid, densely spaced and terrain following close to the ground and sparsely spaced and flat higher up. However, CNNs treat all regions of space the same, and therefore require the same relations to hold in all spatial regions. In **Experiment 1** various approaches to this was tested, and we found that incorporating the z-coordinate of each data point as an extra input channel improved performance significantly. Interpolation to enforce even vertical spacing relative to the ground level also increased performance somewhat, but the evidence for that is weaker.

For incorporating our knowledge of the physics into the model, multiple wind gradient based losses were introduced in Chapter 3.3, and tested in **Experiment 2** and **Experiment 2.5**. A loss function penalizing, in descending order, differences in the $\frac{\partial}{\partial x}\frac{\partial}{\partial y}$ derivatives, horizontal divergence $\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}$ and divergence between the super-resolved and actual high resolution data, mediated by a pixel loss in average absolute error was found to increase performance greatly. No setup was found in which adversarial learning significantly improved performance.

Incorporating these changes into the model, and calibrating the hyperparameters further, the resulting non adversarial generator super-resolved the test dataset wind field with an average PSNR of 47.14 db, average length of the error wind vector 0.24 m/s, and average length of the error wind vector divided by average wind speed 6.12%, as opposed to trilinear interpolation scores 36,42 db, 0.80 m/s and 18.5%. A closer inspection of the 3D wind flow revealed that the model approximate terrain adaptation and turbulence remarkably well, indicating that the physics informed losses in the loss function combined with the terrain information successfully allowed the model to learn terrain adaptation and atmospheric wind flow. The model also works reasonably well for different larger scales of resolution increase.

A bug was found in the rotation and flipping data augmentation process, which lead to the wind vectors' orientation not being rotated as the terrain and their position is. Though this meant the model learned to produce non reasonable wind fields from non reasonable input during training, early results after fixing it indicates that it was only inhibiting performance slightly, with the model still learning meaningful relations between wind flow and terrain.

Specifically adressing the three points from the introduction, it has been found that

1. Microscale 3D atmospheric wind flow can be well approximated by CNNs from low resolution wind data and high resolution terrain data even if the data is irregularly spaced vertically.
2. Loss functions comparing specific parts of the wind gradient of the generated and high-resolution wind fields improved performance significantly, with gains from adversarial learning being negligible in comparison.
3. The proposed fully convolutional 3D generator with wind gradient based loss function super-resolves the near surface 3D wind field remarkably well, both in terms of absolute error, and in subjectively appearing to meaningfully approximate the physics of terrain adaptation of wind flow.

6.1 Further Work

As encouraging as these results are, many parts of the model design have not yet been properly calibrated and tested. Gains in performance can be looked for in

- Improving the architecture of the terrain feature extractor. The current design is a minimum way to include terrain feature extraction, and the number of features has not been properly calibrated.
- Testing without dropout layers, or with other dropout percentages
- Trying a U-net generator architecture. A U-net [38] is a U-shaped encoder-decoder network with skip connections between each corresponding dimension scale in the encoder and the decoder. Such a structure makes intuitive sense for super-resolving the wind field in the manner of this thesis. With a U-net, one could start by interpolating the low resolution wind field, concatenate that with the *high resolution* terrain data/z-coordinates and use this as input for the generator. This way we build all our starting point knowledge into the model from the beginning, as opposed to the current approach: not using the interpolated data, starting with the low resolution

terrain information as input, and concatenating the high resolution terrain information late in the network.

- Improving adversarial learning. Chaotic wind patterns in turbulent regions might pose a fundamental challenge for adversarial learning with 3D atmospheric flow, and the results of this work are somewhat discouraging for adversarial approaches as opposed to informed content loss. Working with adversarial training is also more demanding both computationally and conceptually. Still, improvement by better design of the adversarial process is not ruled out, and remains a possible avenue for further research. Using the feature extractor of a pre-trained discriminator, or for the discriminator saved at some interval during training, to add a feature loss might for instance give better granularity to the adversarial feedback. The code has support for this loss, but it has not been tested properly. Experimenting with such a feature loss can be combined with experimenting with the dropout probability of the discriminator.

Also, the model requires further modifications and testing before actually being usable in wind farm development. In this thesis the model was trained with 3D data 2-40 meters above ground. Wind farms operate in the domain $\sim$ 50-250 meters above ground, so the model will need to be calibrated for a different height domain. Predicting at these heights should be much easier, as wind fields are less complex further away from the ground. Furthermoere, despite the slicing augmentation technique, the model has not experienced much variation in the terrain, and has not been tested on terrain not present in the training dataset. The model might therefore be significantly overfitted to the terrain in the dataset. It should either be trained for many different terrain shapes, or specialised for predicting wind flow in a particular area.

For the model to be useful it also needs to be tailored to the datasets available and the scale one is interested in. Much of available wind data is only available in 2D, or only available in limited geographical domains. Given that the loss function focuses mainly on horizontal derivatives, and that the code already has 2D support, it might be fruitful to test the model on 2D super-resolution. Finally, in the scheme of downscaling, a 4x4 increase in resolution is relatively small, but Chapter 5.2.3 gives us reason to believe that the model can easily be adapted to other scales, provided the requisite terrain and wind field training data.

All in all, the proposed model should be considered a proof-of-concept for CNN driven super-resolution of 3D microscale atmospheric wind flow.

Bibliography

[1] International Energy Agency, "Wind Electricity," 2022.

[2] N. Zehtabiyan-Rezaie, A. Iosifidis, and M. Abkar, "Data-driven fluid mechanics of wind farms: A review," *Journal of Renewable and Sustainable Energy*, vol. 14, no. 3, p. 032703, 2022.

[3] "ERA5: Fifth generation of ECMWF atmospheric reanalyses of the global climate." `https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels?tab=form`, Accessed 2023. Retrieved May 19, 2023.

[4] M. Rummukainen, "State-of-the-art with regional climate models," *Wiley Interdisciplinary Reviews: Climate Change*, vol. 1, no. 1, pp. 82–96, 2010.

[5] J. Barcons, M. Avila, and A. Folch, "A wind field downscaling strategy based on domain segmentation and transfer functions," *Wind Energy*, vol. 21, 02 2018.

[6] W. C. De Rooy and K. Kok, "A combined physical–statistical approach for the downscaling of model wind speed," *Weather and Forecasting*, vol. 19, no. 3, pp. 485–495, 2004.

[7] P.-A. Michelangeli, M. Vrac, and H. Loukos, "Probabilistic downscaling approaches: Application to wind cumulative distribution functions," *Geophysical Research Letters*, vol. 36, no. 11, 2009.

[8] T. Salameh, P. Drobinski, M. Vrac, and P. Naveau, "Statistical downscaling of near-surface wind over complex terrain in southern france," *Meteorology and Atmospheric Physics*, vol. 103, no. 1-4, pp. 253–265, 2009.

[9] A. Winstral, T. Jonas, and N. Helbig, "Statistical downscaling of gridded wind speed data using local topography," *Journal of Hydrometeorology*, vol. 18, 11 2016.

[10] M. Oh, J. Lee, J.-Y. Kim, and H.-G. Kim, “Machine learning-based statistical downscaling of wind resource maps using multi-resolution topographical data,” *Wind Energy*, vol. 25, no. 6, pp. 1121–1141, 2022.

[11] K. Höhlein, M. Kern, T. Hewson, and R. Westermann, “A comparative study of convolutional neural network models for wind field downscaling,” *Meteorological Applications*, vol. 27, no. 6, p. e1961, 2020.

[12] J. Dujardin and M. Lehning, “Wind-topo: Downscaling near-surface wind fields to high-resolution topography in highly complex terrain with deep learning,” *Quarterly Journal of the Royal Meteorological Society*, vol. 148, no. 744, pp. 1368–1388, 2022.

[13] O. Miralles, D. Steinfeld, O. Martius, and A. C. Davison, “Downscaling of historical wind fields over switzerland using generative adversarial networks,” *Artificial Intelligence for the Earth Systems*, vol. 1, no. 4, p. e220018, 2022.

[14] 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*, vol. 117, no. 29, pp. 16805–16815, 2020.

[15] T. N. Larsen, “On the applicability of a perceptually driven generative-adversarial framework for super-resolution of wind fields in complex terrain,” Master’s thesis, NTNU, 2020.

[16] D. T. Tran, H. Robinson, A. Rasheed, O. San, M. Tabib, and T. Kvamsdal, “Gans enabled super-resolution reconstruction of wind field,” *Journal of Physics: Conference Series*, vol. 1669, p. 012029, oct 2020.

[17] J. Hansen, “Fluid Dynamics Of The Atmosphere And Ocean: Mit course lecture notes.” `https://ocw.mit.edu/courses/12-800-fluid-dynamics-of-the-atmosphere-and-ocean-fall-2004/pages/lecture-notes/`, 2004. Retrieved May 22, 2023.

[18] J. Boussinesq, *Théorie de l’écoulement tourbillonnant et tumultueux des liquides dans les lits rectilignes à grande section...*, vol. 1. Gauthier-Villars, 1897.

[19] D. C. Wilcox *et al.*, *Turbulence modeling for CFD*, vol. 2. DCW industries La Canada, CA, 1998.

[20] C. Meissner, A. R. Gravdahl, and B. Steensen, “Including thermal effects in cfd simulations,” in *European Wind Energy Conference and Exhibition*, p. 1, 2009.

[21] *Wind Farm Modeling in a Realistic Environment Using a Multiscale Approach*, vol. Volume 10: Ocean Renewable Energy of *International Conference on Offshore Mechanics and Arctic Engineering*, 06 2017. V010T09A051.

[22] T. R. Oke, *Boundary layer climates.* Routledge, 2002.

[23] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[24] X. Kong, X. Liu, J. Gu, Y. Qiao, and C. Dong, “Reflash dropout in image super-resolution,” 2022.

[25] X. Li, S. Chen, X. Hu, and J. Yang, “Understanding the disharmony between dropout and batch normalization by variance shift,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2677–2685, 2019.

[26] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” 2014.

[27] A. Jolicoeur-Martineau, “The relativistic discriminator: a key element missing from standard gan,” 2018.

[28] 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.

[29] E. Vesterkjær, “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks.” `https://github.com/eirikeve/esrdgan`, 2019. GitHub repository.

[30] 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,” 2017.

[31] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.

[32] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.

[33] Y. Seity, P. Brousseau, S. Malardel, G. Hello, P. Bénard, F. Bouttier, C. Lac, and V. Masson, “The arome-france convective-scale operational model,” *Monthly Weather Review*, vol. 139, no. 3, pp. 976–991, 2011.

[34] M. Själander, M. Jahre, G. Tufte, and N. Reissmann, “EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure,” 2019.

[35] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, “Tune: A research platform for distributed model selection and training,” *arXiv preprint arXiv:1807.05118*, 2018.

[36] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[37] L. Li, K. Jamieson, A. Rostamizadeh, K. Gonina, M. Hardt, B. Recht, and A. Talwalkar, “Massively parallel hyperparameter tuning,” 2018.

[38] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” 2015.

Appendix A

Standard Hyperparameters

Table A.1: Full list of standard hyperparameters with explanation

Overall		
Parameter	**Value**	**Meaning/Comments**
include_pressure	True	**G** input channel
include_z_channel	True	**G** input channel
included_z_layers	1-10	10 of the 11 layers closest to the ground (dropping the one furthest down)
conv_mode	3D	Alternatives: 3D, 2D, Horizontal3D (separate kernel for each horizontal layer)
interpolate_z	False	Interpolate as specified in section 3.1.3
include_altitude_channel	False	include z_{alt} as G input channel
load_G_from_save	False	Initialise **G** with a model trained without adversarial cost

Data		
Parameter	**Value**	**Meaning/Comments**
enable_slicing	True	Slice 64x64 slices as specified in section 3.1.3
scale	4	xy dimension scale difference between LR and SR, e.g. 16x16 $\rightarrow$ 64x64
batch_size	32	
data_augmentation_flipping	True	
data_augmentation_rotation	True	

Architecture		
Parameter	**Value**	**Meaning/Comments**

G/both	D	G/both	D	
num_features		128	32	D features double for every DownConv block except the last
terrain_number_of_features		16		Number of features in the terrain feature extractor part of **G**
num_RRDB		16		
num_RDB_convs		5		RDBs per RRDB
RDB_res_scaling		0.2		Residual scaling, α in the pink part of Figure 3.6
RRDB_res_scaling		0.2		Residual scaling, α in the green part of Figure 3.6
hr_kern_size		5		Kernel size after upscaling in the generator (5x5x5)
kern_size	kern_size	3	3	Standard kernel size (3x3x3)
weight_init_scale		0.1	0.2	Smaller kaiming weight initialization
lff_kern_size		1		local feature fusion layer of RDB
dropout_probability		0.1	0.2	
G_max_norm		1.0		Max norm for gradient clipping, functions as C in equation (2.35)

Training				
Parameter		**Value**		**Meaning/Comments**
G/both	**D**	**G**/both	**D**	
learning_rate		1e-5	1e-5	
adam_weight_decay_G		0	0	
adam_beta1_G		0.9	0.9	
adam_beta2_G		0.999	0.999	
multistep_lr		True		Reduce learning rate during training
multistep_lr_steps		[10k, 30k, 50k, 70k, 100k]		schedule for reducing learning rate during training
lr_gamma		0.5		factor of reduction when lr is adjusted
gan_type		relativisticavg		section 2.3
adversarial_loss_weight		0.005		η_6 (3.15)
gradient_xy_loss_weight		1.0		η_2 (3.15)
gradient_z_loss_weight		0.2		η_3 (3.15)
xy_divergence_loss_weight		0.25		η_5 (3.15)
divergence_loss_weight		0.25		η_4 (3.15)
pixel_loss_weight		0.15		η_1 (3.15)
d_g_train_ratio		1		How often D is trained relative to G
use_noisy_labels		False		Add noise to labels when training D
use_label_smoothing		True		Set HR labels to $0.9 + 0.1\frac{it}{it_{max}}$ instead of 1.0 when training D

flip_labels	False	Randomly flip (set HR_label=false, SR_label=True) some labels when training D
use_instance_noise	True	Add instance noise when training $\mathbf{D}$, $\mathbf{D}(\mathbf{x}) \to \mathbf{D}(\mathbf{x}+\varepsilon)$ $\varepsilon_i \sim \mathcal{N}\left(0,\, 2.0\left(1-\frac{it}{niter}\right)\right)$
niter	90k	Number of training iterations
train_eval_test_ratio	0.8	Ratio for training, remaining fraction split equally between validation and testing