Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Thomas Nakken Larsen

On the applicability of a perceptually driven generative-adversarial framework for super-resolution of wind fields in complex terrain

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
July 2020

NTNU
Kunnskap for en bedre verden

Thomas Nakken Larsen

On the applicability of a perceptually driven generative-adversarial framework for super-resolution of wind fields in complex terrain

Master's thesis in Cybernetics and Robotics
Supervisor: Adil Rasheed
July 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

Great strides have been made in recent years in single-image super-resolution (SISR) tasks by utilizing high-dimensional feature activations from pre-trained feature extractors to introduce a perceptual loss in generative-adversarial networks (GANs). A perceptually driven GAN model was recently proposed for super-resolving wind fields in complex terrain. While the generative model was shown to produce plausible wind fields, no statistical analysis was shown, nor was the perceptual aspect of the model justified for application on wind data.

This thesis investigates the applicability of such a perceptually driven model to super-resolve low-resolution wind fields. An initial stability analysis found that the perceptual loss component for the generative model consistently failed to converge. Consequently, an in-depth analysis was performed on the wind data from the perspective of the convolutional feature extractor used to construct this perceptual loss. Considering that the feature extractor was explicitly trained to classify RGB images, wind datasets were converted into an RGB representation to build an intuition for their equivalent visual complexity. It was hypothesized that the generative model was unable to fully learn the visual characteristics of the presented wind data due to the significant difference between the dataset used to train the feature extractor and the wind data used to train the generative model. Thus, the variations in feature activations are thought to act as a source of noise for the generative model rather than helping it improve the accuracy in its super-resolved wind fields. By setting the model to super-resolve wind data from the top of the domain, it was able to converge as expected. Therefore, it was shown that the perceptual feedback from the feature extractor inhibits the model.

By enforcing an agreement evaluation between the Peak Signal-to-Noise Ratio (PSNR) and the Learned Perceptual Image Patch Similarity (LPIPS) metrics, it was shown that minimizing the perceptual loss is not synonymous with learning the governing equations of airflow in the relevant domain. In conclusion, the proposed model was deemed insufficient for the task of super-resolving wind fields in complex terrain.

Sammendrag

Det har nylig blitt gjort store fremskritt innenfor datasyn for konstruksjon av høyoppløste bilder fra referansebilder med lav oppløsning ved å benytte høy-dimensjonale aktiveringer fra “feature-extractors” for å danne et perseptuelt tap i “generative-adversarial” nettverk (GANs). En perseptuelt drevet GAN-modell ble nylig foreslått for å øke oppløsningen av atmosfærisk vind i komplekst terreng. Dog det ble vist til at den generative modellen kan produsere sannsynlige vindfelt, ble det ikke fremvist noen statistisk analyse, og det perseptuelle aspektet av modellen var ikke begrunnet for anvendelse på vinddata.

Denne masteroppgaven undersøker anvendeligheten av en slik perseptuelt drevet modell for å øke oppløsningen av atmosfærisk vind med opprinnelig lav oppløsning. En initiell stabilitetsanalyse viste at den perseptuelle tapskomponenten for den generative modellen konsekvent ikke klarte å konvergere. Følgelig ble en grundig analyse utført på vinddata fra perspektivet til den konvolusjonelle “feature extractor”’en som ble brukt til å konstruere dette perseptuelle tapet. Flere vinddatasett ble representert som RGB-bilder for å bygge en intuisjon for deres visuelle kompleksitet, tatt i betraktning at den anvendte “feature extractor”’en opprinnelig ble trent til å klassifisere RGB-bilder. Det ble antatt at den generative modellen ikke var i stand til å lære de visuelle egenskapene til vinddataene på grunn av den signifikante forskjellen mellom datasettet som ble brukt til å trene “feature extractor”’en og vinddataene som ble brukt til å trene den generative modellen. Videre ble det antatt at variasjoner i aktiveringer fungerer som en kilde til støy for den generative modellen i stedet for å hjelpe den med å forbedre nøyaktigheten i sine genererte høyoppløste vindfelt. Ved å sette modellen til å heller øke oppløsningen av vinddata fra toppen av domenet, klarte den å konvergere som forventet. Dermed ble det vist at den perseptuelle tilbakemeldingen fra “feature extractor”’en hemmer modellen.

Ved å sammenligne mellom målinger av “Peak Signal-to-Noise Ratio” (PSNR) og “Learned Perceptual Image Patch Similarity” (LPIPS), ble det vist at å minimere det perseptuelle tapet ikke nødvendigvis korresponderer med å lære den grunnleggende dynamikken av luftstrømninger i det relevante domenet. Avslutningsvis ble den foreslåtte modellen ansett som utilstrekkelig for oppgaven med å øke oppløsningen av vindfelt i komplekst terreng.

Acknowledgments

I wish to give credit where credit is due; this thesis would not be possible to realize without the assistance provided by the following people:

- My supervisor, Adil Rasheed, who introduced me to GANs and gave me the opportunity to work with this topic.
- My co-student, Duy Tan Tran, for providing the Python implementation of the proposed model and being a valuable discussion partner regarding the issues found with the model.
- The HPC group at NTNU, for allowing me to use their high-performance computational cluster, IDUN, to train numerous instances of the neural network model.
- I also acknowledge the support from I got from the OPWIND: Operational Control for Wind Power Plants project (Grant No.: 268044/E20).
- And last, but not least, a special thanks to my family and all the friends I've made along the way for the incredible support they have given me throughout the Cybernetics and Robotics course at NTNU.

Trondheim, 01.07.2020 Thomas Nakken Larsen

Preface

This thesis concludes a Master of Science in Cybernetics and Robotics at the Department of Engineering Cybernetics of the Norwegian University of Science and Technology (NTNU). It was created under the supervision of Adil Rasheed during the spring of 2020.

The preceding specialization project considered a different topic within supervised learning, thus the author has no prior experience in working with generative-adversarial networks (GANs). Initially, the thesis was intended to extent the application of a novel GAN model to predict airflow using simulated satellite and LIDAR data. Complications ecountered underway lead the thesis to change directions multiple times. Ultimately, the thesis show how the proposed model is unfit for the task of airflow super-resolution. In order to determine the underlying issues within the model, a rigorous literature search has populated the background chapter with detailed information relating to the fundamental issues related to assumptions made for the convergence of GAN frameworks, as well as some introductory details relating to the perceptual aspect of image super-resolution.

The Python implementation of the ESRGAN model was provided by Duy Tan Tran but was originally implemented by Eirik Ekjord Vesterkjær. All airflow datasets were created using simulated data from the coupled HARMONIE-SIMRA system. Execution of the ESRGAN model was facilitated by the HPC Group at NTNU, utilizing the IDUN cluster. All plots and figures in this thesis were created using the Python library matplotlib and the scientific data visualization engine Mayavi. Other figures are used with explicit consent from their respective authors, and are cited below the figure.

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network. 1, 10–14, 16, 17, 27, 33, 34, 36, 40, 75

cGAN Conditional GAN. 30, 32

CNN Convolutional Neural Network. 1, 17, 21, 33–35, 50, 54, 75

CV Cross-Validation. 9

DCGAN Deep Convolutional GAN. 21

EMD Earth Mover's Distance. 28, 32

ESRGAN Enhanced Super-Resolution Generative Adversarial Network. 34, 36, 38–40, 43, 44, 47, 48, 50, 52–54, 56, 70, 72, 75, 80, 83, 92, 93

GAN Generative Adversarial Network. 1, 21, 23–25, 27–32, 34, 36, 45–47, 69, 71, 92, 101

HARMONIE Hirlam Aladin Regional Mesoscale Operational Numerical prediction In Europe. xi, 6–8, 36, 41, 52, 56, 99

HPC High Performance Computing. 40

HR High-Resolution. 32–34, 36, 38, 43, 44, 72, 80

IPM integral probability metric. 28, 29, 32

JSD Jensen-Shannon Divergence. 24, 28, 29, 31, 32

KLD Kullback-Liebler Divergence. 24

LeakyReLU Leaky Rectified Linear Unit. 11, 36

LPIPS Learned Perceptual Image Patch Similarity. 34, 35, 54, 55, 83–90, 93

LR	Low-Resolution. 32–34, 36, 43
ML	Machine Learning. 1, 8, 10
MSE	Mean Squared Error. 13, 33, 34, 54, 55, 83–85, 93
NetCDF	Network Common Data Form. 41, 42, 99
NTNU	Norwegian University of Science and Technology. 40
PCA	Principal Component Analysis. 10
PSNR	Peak Signal-to-Noise Ratio. 33–35, 53–55, 57, 59, 63, 66–68, 71, 72, 80, 83, 84, 89, 93
RaGAN	Relativistic Average GAN. 29–32, 34, 46, 59, 68, 71
RDB	Residual Dense Block. 46
ReLU	Rectified Linear Unit. 11, 12, 34, 36
RGAN	Relativistic GAN. 29, 32
RRDB	Residual-in-Residual Dense Block. 34, 36, 46
SIMRA	Semi Implicit Method for Reynolds Averaged navier-stokes equations. xi, 7, 8, 36, 41, 42, 48, 52, 56, 73, 79, 94, 99
SISR	Single-Image Super-Resolution. 32–34, 36, 48, 70, 72, 92
SR	Super-Resolution. 33, 34, 38, 53, 80
SRCNN	Super-Resolution Convolutional Neural Network. 33
SRGAN	Super-Resolution Generative Adversarial Network. 34
SSIM	Structural Similarity Index. 34
tanh	Hyperbolic Tangent. 11, 12, 16
WGAN	Wasserstein GAN. 28, 29, 32, 102

Chapter 1

Introduction

Data-driven methods, especially in Machine Learning (ML) algorithms using Artificial Neural Networks (ANNs), have become increasingly popular due to the advancement in computational power and the increase in open-source datasets over the last decade. Convolutional Neural Networks (CNNs) have become highly proficient in supervised learning tasks such as classification and low-dimensional regression, even outperforming human performance in computer vision tasks ([25, 23, 44, 17]). Of particular interest is the Generative Adversarial Network (GAN) framework, which is a purely data-driven approach that can approximate high-dimensional probability distributions. When Goodfellow *et al.* introduced it in 2014, the framework was notoriously hard to stabilize and train to convergence, but recent contributions have identified and alleviated several of the issues related to the fundamental training algorithm ([10, 42, 2, 29, 31]). This framework enables the use of neural networks in unsupervised learning problems and has become widely applied in computer vision image generation, style-transfer, and super-resolution. In particular, the use of a CNN to evaluate the perceptual distance between a super-resolved image and its reference has significantly improved the state-of-the-art performance ([20, 8, 27, 50]).

Numerical simulation of complex differential equations in fluid dynamics dominates the state-of-the-art in fluid simulations. Despite that fluid dynamics long have been accurately described through the Navier-Stokes equations, their sheer complexity makes them infeasible to solve directly in real-time on current hardware. In wind engineering applications, nested models are interpolated to solve the flow at different scales. For applications such as weather forecasting and wind-power estimation, there is a large amount of high-dimensional data stored from simulated atmospheric flow in geographic domains [38, 39]. A novel method that utilizes a perceptually driven GAN model for super-resolving airflow in complex terrain was recently proposed ([47]). This generative model aims to learn the governing equations related to ground-level airflow in a geographical domain to upscale low-resolution wind fields to high-resolution in real-time. The model applies methods that have significantly improved the state-of-the-art in image super-resolution, namely the use of a pre-trained, convolutional feature extractor. While it has been shown that the proposed model can produce plausible results, the fundamental dynamics in the generative-adversarial framework has not been justified for learning governing equations for airflow.

1.1 Problem description

This thesis considers an in-depth investigation of the proposed perceptually driven, super-resolution generative-adversarial framework aimed at reconstructing ground-level airflow in complex terrain. Through initial analysis, it was discovered that the model struggled to converge consistently. The subsequent investigation was formed through an iterative-inductive process aimed at determining the cause of this convergence issue.

1.2 Thesis outline

This thesis is divided into 5 chapters:

- **Chapter 1** is the current chapter. It presents the motivating factors for investigating the validity of combining computer vision methods for solving tasks traditionally performed by numerical simulations of wind fields. Three research questions are raised regarding the application of the proposed model.
- **Chapter 2** describes the relevant background required to understand the internal dynamics of the proposed model. The basics of machine learning, artificial neural networks, and deep learning principles lead to the presentation of generative-adversarial networks. Typical failure modes are described, and recent contributions to avoid them are explained. Finally, the typical application of GANs in single-image super-resolution tasks is presented along with the proposed model for airflow data.
- **Chapter 3** presents three experiments aimed to find whether the proposed model applies to airflow data.
- **Chapter 4** shows the results of each experiment and provides a logical progression between each conducted experiment.
- **Chapter 5** concludes the thesis by answering the research questions presented in Chapter 1 and suggests potential further progression of the covered topics.

1.3 Research questions

1. Why does the previously proposed GAN-based super-resolution model consistently fail to reproduce its results?

2. What is the fundamental issue with applying the pre-trained feature extractor to airflow data?
3. Is the generative model's task of minimizing a perceptual loss synonymous with learning the governing equations of airflow in the relevant domain?

Chapter 2

Background

This chapter establishes the theory required to justify the methods used in Chapter 3. Three main topics are presented; first, the governing equations of airflow are described and used to quantify the computational complexity of solving these numerically. A coupled multi-scale model solving the governing equations is presented as the source of data used in the thesis. The second topic is machine learning, a massive topic, thus only the most relevant topics are described. Lastly, artificial neural networks are introduced, which provides a basis for presenting data-driven generative models using artificial neural networks.

2.1 Flow in complex terrain

Fluid flow is governed and restricted by the fundamental conservation laws. Although these laws are generally applicable, Equations 2.1-2.6 present specifically termed governing equations for atmospheric flow, as described by Rasheed *et al.* [39]. Where applicable, the corresponding name of each equation is stated. The notations used in Equations 2.1-2.6 are described in Table 2.1.

$$\nabla \cdot (\rho_s \mathbf{u}) = 0 \qquad \text{Conservation of mass} \tag{2.1}$$

$$\frac{D\mathbf{u}}{Dt} = -\nabla \left(\frac{p_d}{\rho_s} \right) + \mathbf{g}\frac{\theta_d}{\theta_s} \nabla \cdot \mathbf{R} + \mathbf{f} \qquad \text{Conservation of momentum} \tag{2.2}$$

$$\frac{D\theta}{Dt} = \nabla \cdot (\gamma_T \nabla \theta) + q \qquad \text{Conservation of energy} \tag{2.3}$$

$$\frac{Dk}{Dt} = \nabla \cdot (\nu_T \nabla k) + P_k + G_\theta - \epsilon \qquad \text{Turbulent kinetic energy} \tag{2.4}$$

$$\frac{D\epsilon}{Dt} = \nabla \cdot \left(\frac{\nu_T}{\sigma_e} \nabla \epsilon \right) + (C_1 P_k + C_3 G_\theta) \frac{\epsilon}{k} - C_2 \frac{\epsilon^2}{k} \qquad \text{Turbulent dissipation} \tag{2.5}$$

$$\nu_T = C_\mu \frac{k^2}{\epsilon} \tag{2.6}$$

In Equations 2.1-2.6, the subscripts s, d indicate the associated term's hydrostatic value and deviation from the hydrostatic value, respectively. Thus, $p = p_s + p_d$, $\theta = \theta_s + \theta_d$, $\rho = \rho_s + \rho_d$. The hydrostatic equation is given by $\partial p_s / \partial z = -g \rho_s$. Although ρ_s normally isn't measured, it can be calculated using the ideal gas law: $\rho_s = p_s / R\theta (p_o/p_s)^{R_g/C_p}$, where C_p is the specific heat capacity for an ideal gas at constant pressure and R_g is the gas constant. $\mathbf{R}$, P_k, and G_θ are given by Equations 2.7-2.9.

$$R_{ij} = \nu_T \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} k \delta_{ij} \tag{2.7}$$

$$P_k = \nu_T \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \frac{\partial u_i}{\partial x_j} \tag{2.8}$$

$$G_\theta = -\frac{g}{\theta} \frac{\nu_T}{\sigma_T} \frac{\partial \theta}{\partial z} \tag{2.9}$$

Table 2.1: Physical representations of terms in governing equations for air flow.

Term	Description
$\mathbf{u}$	Velocity
p	Pressure
θ	Potential temperature
ρ	Density
$\mathbf{R}$	Stress tensor
$\mathbf{f}$	Source/sink term
$\mathbf{g}$	Acceleration due to gravity
γ_T	Thermal diffusivity
q	Temperature source

Some of the terms described in the equations above have different physical applications depending on the scale. In a mesoscale model, the stress tensor **R** and source/sink term **f** can be used to represent the Coriolis forces; in a supermicroscale model, **R** and **f** can be used to represent aerodynamic resistance offered by turbines.

Similarly, while Equations 2.4-2.5 constitute a two-equation turbulence model in micro- and supermicroscale context, the turbulent dissipation term, ϵ, in Equation 2.4 is substituted with an approximation in a mesoscale context. This approximation is given by $\epsilon = (C_\mu^{1/2}K)^{3/2}/l_t$, where l_t is computed as:

$$l_t \approx \frac{\min(\kappa z, 200m)}{1+5Ri} \tag{2.10}$$

$$Ri = \frac{\frac{g}{\theta}\frac{\partial \theta}{\partial z}}{\left(\frac{\partial u}{\partial z}\right)^2} \approx -\frac{G}{P} \qquad \text{Richardson number} \tag{2.11}$$

Therefore, the two-equation turbulence model can be reduced to a single-equation model when considering a mesoscale context. Note that the stability correction term in Equation 2.10, $(1+5Ri)$, is replaced with $(1-40Ri)^{-1/3}$ in convective conditions. Finally, the remaining unexplained terms are constant, scalar coefficients specified as:

Table 2.2: Scalar coefficients of governing equations for airflow

C_μ	C_1	C_2	C_3	κ	σ_K	σ_ϵ
0.09	1.92	1.43	1.00	0.40	1.00	1.30

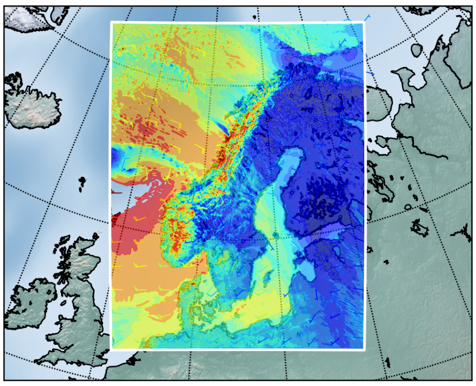
The mentioned sets of governing equations are not trivial to solve. The current state-of-the-art is still utilizing numerical models for approximating airflow, and the numerical methods differ depending on the scale of the relevant domain. A coupled system of two numerical models at different scales is presented in the next chapter.

2.2 HARMONIE-SIMRA: a coupled multi-scale model for airflow data generation

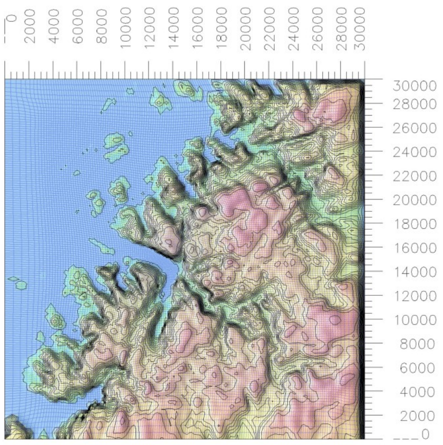
This chapter presents a brief introduction to a coupled set of two numerical models for solving the governing equations described in the previous chapter. Their introduction here is meant to serve as a motivation for the use of data-driven methods later on in the thesis. Details regarding the computational methods are therefore out of the scope of this thesis, but the reader can refer to Rasheed *et al.* [38] and Rasheed *et al.* [39] for more information.

The Hirlam Aladin Regional Mesoscale Operational Numerical prediction In Europe (HARMONIE) is a non-hydrostatic, mesoscale model "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" [39]. The domain covered by the HARMONIE model is shown in Figure 2.1a.

The Semi Implicit Method for Reynolds Averaged navier-stokes equations (SIMRA) is a microscale model for anelastic flow. It utilizes the Boussinesque approximation and has a fine resolution near wall boundaries to resolve interaction with terrain and ocean surfaces. SIMRA "solves prognostic equations for all velocity components, potential temperature and pressure" [39]. These variables are solved using Equations 2.1-2.2 described above. Additionally, turbulent kinetic energy and turbulent dissipation are solved using Equations 2.4 and 2.5, respectively. The domain covered by the SIMRA model is shown in Figure 2.1b.



(a) HARMONIE domain



(b) SIMRA domain and mesh (Bessaker, Norway)

Figure 2.1: Visual representation of the domains covered in numerical models for atmospheric flow.

Source: Rasheed *et al.* [39]

The coupled HARMONIE-SIMRA system is formed by initializing the SIMRA microscale model using interpolated information from the HARMONIE mesoscale model. Table 2.3 shows how Rasheed *et al.* reports the computational resources required to run the HARMONIE and SIMRA models. Although the specific computational models are omitted, their time complexity and use of resources is relevant for later chapters.

As described, the coupled HARMONIE-SIMRA system is based on solving the set of governing equations for airflow numerically, using the resources described in Table 2.3. This thesis considers a data-driven generative model for solving these equations. The generative model is based on a set of artificial neural networks which, when fully trained, can solve the governing equations faster, potentially

Table 2.3: Computational details and resources used to run the HARMONIE and SIMRA models. Recreated from Rasheed *et al.,* [39].

Model	Cores	Domain size [km]	Mesh elements	Time [minutes]
HARMONIE	1840	$1875 \times 2400 \times 26$	$46 \cdot 10^6$	87
SIMRA	48	$30 \times 30 \times 2.5$	$1.6 \cdot 10^6$	13

several orders of magnitude. Before this model is presented, relevant background for machine learning methods are covered.

2.3 Machine learning

Machine Learning (ML) is a field of study that has recently gained popularity due to the increase in speed, availability, and capacity of computational power in hardware, as well as the increased availability of large datasets. This development has made ML algorithms realizable, as fundamental ML algorithms have previously existed as purely theoretical due to their computational demand. ML techniques are algorithms intended for a computer to perform actions or make conclusions based on discovering patterns in data. Several approaches exist, depending on the type of data and the task to perform. This thesis presents a model that utilizes aspects in both *supervised* and *unsupervised learning* problems. Therefore, these approached are explained, condensed from their description in [41, 16].

2.3.1 Supervised learning

Supervised learning is a task where an algorithm has access to a dataset consisting of input-output pairs $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$. It is assumed that the outputs, y_i, are generated by an unknown function $f(x_i)$, where there are no restrictions on the values or dimensions of x_i or y_i. The supervised learning task ultimately attempts to approximate the function $f(x_i)$, given x_i and y_i.

Approximating this *hypothesis*, $\hat{f}(x_i)$, for the true function $f(x_i)$ that is able to map all inputs x_i to outputs y_i is not a hard task. However, the hypothesis should also *generalize* to unseen data. An algorithm's ability to generalize is usually evaluated by withholding a *test set* during the training phase. A hypothesis is formed from the training data, and its ability to generalize is evaluated on the test set. If an algorithm is perfectly capable of classifying training data but performs significantly worse on test data, it is said to be *overfitted* to the training data. Overfitting can be avoided by carefully selecting appropriate algorithms for the task using *a priori* knowledge of the task at hand.

Additionally, most modern algorithms are equipped with adjustable *hyperparameters.* As no algorithm can be expected to work for any dataset, hyperparameters allow tuning the algorithm based on prior knowledge of the applied data. In contrast, *parameters* refer to internal variables that are formed through the training and constitute the algorithm's hypothesis.

The selection of hyperparameter values is more often than not non-deterministic. Several combinations of hyperparameters may lead an algorithm to converge, although another set of values may lead to better performance. Therefore, hyperparameter *tuning* is standard practice while validating an algorithm. Tuning is often done through trial and error, where the algorithm is run multiple times using different sets of hyperparameters. Ideally, one would like to minimize the number of hyperparameters to tune, as this optimization problem may also suffer from the *curse of dimensionality*, which is explained further in Chapter 2.5. During the tuning process, it is essential to avoid invalidating the results by *peeking* at the test set. If the test set is used to improve the algorithm's performance through tuning, then some information from the test set has leaked into the learning algorithm and corrupted the experiment. Therefore, it is common practice to withhold an additional part of the training dataset for validation, commonly called a *validation set*. There exist many different methods for splitting the data into training, validation and test sets, depending on the algorithm, and the nature and amount of the available data. Among the most common methods is k-fold Cross-Validation (CV), where the algorithm is run k times using a different fraction $1/k$ of the training set as a validation set for each execution. This way, the whole training set is utilized both for training and validation, and the validation performance is evaluated as an average across the k executions. Only after the tuned algorithm is fixed and final can it be evaluated on the test set for the final results.

Supervised learning tasks can either be intended for *classification* or *regression* tasks. In a classification task, the target output values are represented by a finite set of discrete values (e.g "Real", "Fake"), often described as "labels". In contrast, a regression task involves approximating the output value(s) from a continuous distribution of values (e.g. height 0-1000m). In both cases, the algorithm generates a predicted output $\hat{y_i} = \hat{f}(x_i)$, and receives feedback through a *loss function*, $L(y_i, \hat{y_i})$, to update the algorithm's parameters between iterations in the training data.

2.3.2 Unsupervised learning

Unsupervised learning is, in contrast to supervised learning, a task where an algorithm has access to a dataset consisting of input data $x_1, x_2, ..., x_N \in X$, with no known corresponding output values $y_1, y_2, ..., y_N \in Y$. The goal of an unsupervised learning task is usually to draw inferences from the dataset, typically in data where the underlying patterns are unknown. This task is less intuitive than supervised learning due to the absence of a "correct" mapping, or "true" labels. To signify the distinction (and relation) between this task and supervised learning, let's rephrase the supervised learning task; suppose that the input-output pairs in supervised learning are represented by some joint probability density $\Pr(X, Y)$. Now, supervised learning can be described as a density estimation problem and the algorithm's task is to approximate the conditional density $\Pr(Y|X)$, given the relation $\Pr(X, Y) = \Pr(Y|X) \cdot \Pr(X)$, where $\Pr(X)$ is the marginal density of the values in X. The conditional density is normally found using a loss function in an optimization problem to find the optimal hypothesis. Due to the (typically) low dimensionality of Y in supervised learning, solving this optimization problem tends to be sufficient, and estimating $\Pr(X)$ is not necessary [16].

In comparison, the unsupervised learning task is to directly find intrinsic properties of $\Pr(X)$ without any of the aids present in supervised learning. Furthermore, the typical dimensionality of X tends to be significantly higher in unsupervised learning tasks. Unsupervised algorithms, such as Principal Component Analysis (PCA) attempt to map a sub-space of X, $\hat{X} \in \mathbb{R}^k$, that explains the majority of the variance in X, where $X \in \mathbb{R}^n$ and $k \leq n$. Other algorithms such as *Clustering*, are presented a dataset hypothesized to contain distinct categories of samples, for which the algorithm attempts to explain $\Pr(X)$ using a set of simpler densities representing distinct categories within the dataset. Due to the lack of an equivalent to a loss function, there are no definitive measure of quality nor any conclusive arguments for the viability of an algorithm. Therefore, one often resorts to heuristics to select and evaluate an unsupervised learning algorithm.

2.4 Artificial neural networks

Although traditional ML algorithms have many application areas due to their relatively simple designs and proofs of convergence, there is a limit to their use when either the complexity or dimensionality of the data in question becomes large. Artificial Neural Networks (ANNs) improve upon these methods by being modeled to approximate *any* function, disregarding the data complexity and dimensionality. These properties come at a cost, typically concerning the sheer amount of data, training time, and computational resources needed to approximate the desired function. Essential building blocks of ANNs are introduced in this chapter, using inspiration from their descriptions in [32] and [41].

The main inspiration for modeling ANNs is the human brain. Mental activity is realized through an ensemble of electrochemical activations in neurons. With some exceptions, individual neurons output a signal based on the sum of inputs from interconnected neurons. If this sum exceeds the threshold potential, the neuron is said to “fire”; otherwise, it does nothing [35]. An “artificial neuron” has a similar structure; a linear combination of weighted inputs are summed and fed to an activation function, which yields a scalar output. This structure can be modeled as:

$$y = f(w^T x + b), \tag{2.12}$$

where $(w, x) \in \mathbb{R}^n$ is the weight and input vectors, $f : \mathbb{R} \to \mathbb{R}$ is the activation function and $(y, b) \in \mathbb{R}$ is the output and bias, respectively. The output is then connected to the input of an arbitrary number of subsequent artificial neurons. A simple model of an artificial neuron is visualized in Figure 2.2. These artificial neurons often referred to as “nodes” or “units”, are assembled in linked *layers.* An ANN is simply a collection of such layers, where the outputs of layer (i-1) are linked to the inputs of layer (i), and the outputs of layer (i) are linked to the inputs of

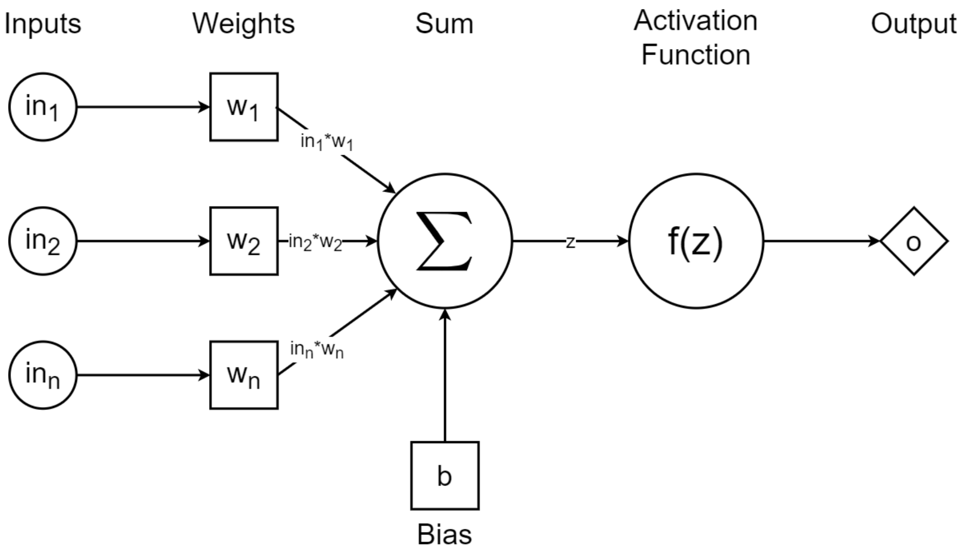


Figure 2.2: A simple model of an artificial neuron.

layer (i+1), as presented in a simple example in Figure 2.3. Networks with this type of architecture are said to be *feed-forward* neural networks. Other network architectures exist, although they exceed the scope of this thesis and is left for the reader to explore. Every ANN has at least one input layer and an output layer, while the topology of the hidden layers is an important design choice that has a significant impact on its properties.

2.4.1 Activation functions

Essentially, any differentiable function $f : \mathbb{R} \rightarrow \mathbb{R}$ has the potential for use as an activation function. However, in the interest of keeping the computational time low and enable the ANN to represent a nonlinear function, it is common to select a simple, nonlinear function for this purpose. Due to the difficulty of designing good activation functions, the typical choice changes as discoveries are made. At the time of writing, a plethora of activation functions have emerged, and the choice typically depends on the application. However, the Rectified Linear Unit (ReLU), presented in Figure 2.4a, is one of the most commonly used activation functions in ANNs today. ReLU is exceptionally simple, as it acts as a purely linear function as long as the input is larger than zero. It improves upon the traditional Hyperbolic Tangent (tanh) and Sigmoid activation functions by reducing the chance of suffering from the *vanishing gradients* problem (Chapter 2.4.3), although it is still susceptible to it. Leaky Rectified Linear Unit (LeakyReLU), presented in Figure 2.4b, avoids this problem by introducing a weak, scaled negative slope for inputs less than zero. Instead of completely cutting off the flow of negative values through the network, LeakyReLU allows some negative values to pass through. Equations 2.13-2.15

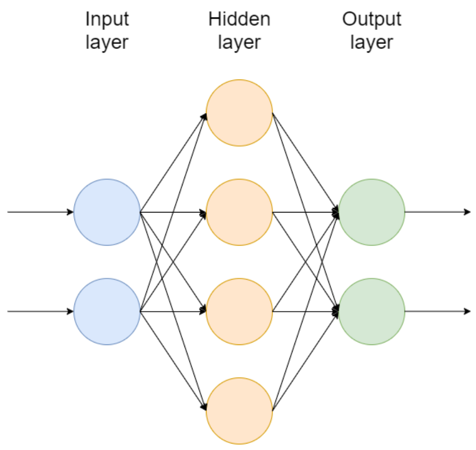


Figure 2.3: Example ANN with a single hidden layer.

describe mathematical representations of the relevant activation functions.

Multiple activation functions are often used for a single ANN. Like the initial choice of activation function, combining different activation functions is a design choice and depends on the application area. For instance, one might choose ReLU as the main activation function for hidden layers, but the application may require an output between -1 and 1. In that case, one may choose to replace ReLU with tanh in the output layer. tanh is presented in Figure 2.4c.

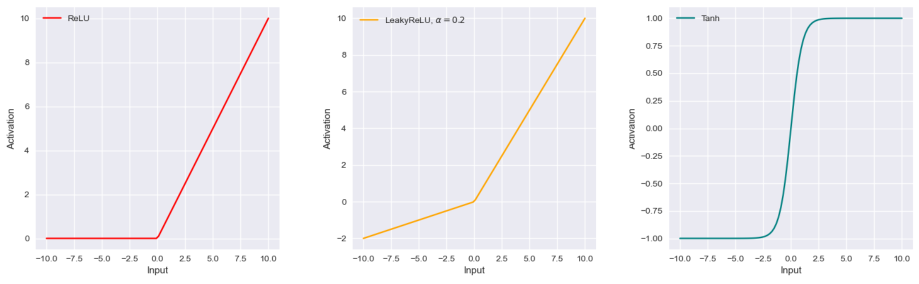
$$\mathrm{ReLU}(x) = \max(0, x) \qquad \text{ReLU activation function} \tag{2.13}$$

$$\mathrm{LeakyReLU}(x) = \max(\alpha x, x) \qquad \text{LeakyReLU activation function} \tag{2.14}$$

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \qquad \text{Tanh activation function} \tag{2.15}$$

2.4.2 Loss functions

A measure of error is necessary for quantifying how well the ANN has approximated the true function. When the network is initialized with random weights, it will naturally produce nonsensical outputs. Updating the network's weights is the only way to alter its hypothesis during training. Loss functions serve the purpose of providing feedback to the network by quantifying the distance between the network's output and the target output. These are typically modeled as $\mathrm{L}(y, \hat{y})$, where $L : \mathbb{R}^n \rightarrow \mathbb{R}$ is some measure of distance or error. Consider the following



(a) ReLU **(b)** LeakyRelu **(c)** Tanh

Figure 2.4: Relevant activation functions used in ANNs.

loss function:

$$
\begin{aligned}
L(y, \hat{y}) &= \frac{1}{N} \sum_{x} ||y(x) - \hat{y}(x)||^2 \qquad \text{Mean Squared Error} \qquad (2.16) \\
&= \frac{1}{N} \sum_{x} ||y(x) - \hat{f}(w^T x + b)||^2,
\end{aligned}
$$

where N denotes the number of input-output pairs (x, y). This widely used loss function represents the Mean Squared Error (MSE) between the target output and the estimated output. Note that $L(y, \hat{y}) \geq 0$. A perfect estimation leads to a loss of zero, indicating that the network has found a sufficient mapping. Conversely, a high loss indicates a poor hypothesis. Earlier, it was described that ANNs can approximate any function. With the introduction of artificial neurons, loss functions and their properties, the training algorithm can be introduced.

By substituting $\hat{y}(x)$ with $\hat{f}(w^T x + b)$ in Equation 2.16, the loss function is no longer represented only by the network's output layer, but rather its full set of layers. Thus, the loss can be spread across the entire network. When the loss function is used to form a classic, unconstrained optimization problem:[1]

$$\min(L(y, \hat{f}(w^T x + b))),$$

then the ultimate goal of approximating a true function is synonymous to solving the optimization problem. Gradient descent is a widely used optimization algorithm in the context of training ANNs. Without going into detail, this algorithm is based on an iterative updating scheme using the loss gradient:

$$\theta_{i+1} \leftarrow \theta_i - \alpha \nabla L(\theta_i), \qquad (2.17)$$

where i is the current iteration, θ represents the parameters (weights and biases) in the network, and α is the first hyperparameter to be introduced: the *learning rate.*

[1]Theory regarding optimization problems is not covered in this thesis; the reader is referred to Nocedal & Wright [33] to explore this topic.

Notice that θ is not defined for $i = 0$, which motivates the need for a parameter initialization scheme. However, applying this algorithm on an ANN induces a more significant problem; calculating the loss gradient, $\nabla L(\theta_i)$, wrt. each network parameter. Fortunately, a simple and efficient algorithm to solve this problem was popularized by Rumelhart *et al.* [40].

2.4.3 Backpropagation

Backpropagation is the backbone of learning in ANNs. Principles described this far all lead up to this algorithm, which solves the seemingly impossible task of adjusting values for each and every weight and bias in an arbitrarily designed feed-forward neural network. Fortunately, the solution is relatively simple; backpropagation utilizes the fact that the outputs of a layer are solely dependent on the local parameters (weights and biases), its activation function and the outputs of the previous layer. In combination with the chain rule, $\frac{df(g(x))}{dx} = \frac{df}{dg}\frac{dg}{dx}$, backpropagation is able to explicitly distribute the loss gradient across all nodes in the network. Notation and descriptions for backpropagation in this chapter are condensed from Nielsen [32]. Their work includes intuitive interpretations of intermediate terms, which is left for the reader to explore.

Establishing specific notation

Before deriving the relevant equations for this algorithm, some notation is necessary to describe individual weights and biases in an arbitrary ANN. Table 2.4 describes this notation and introduces an additional measure, δ^l_j, which will be shown to be particularly useful. For the purpose of keeping the notation as in [32], $C(\cdot)$ is used

Table 2.4: Notation for the backpropagation algorithm

w^l_{jk}		The weight for the connection from the kth neuron in the $(l-1)$th layer to the jth neuron in the lth layer.
b^l_j		The bias of the jth neuron in the lth layer.
z^l_j	$(\sum_k w^l_{jk} a^{l-1}_j) + b^l_j$	The weighted input to the jth neuron in the lth layer.
$\sigma(\cdot)$		The activation function.
a^l_j	$\sigma(z^l_j)$	The activation of the jth neuron in the lth layer.
δ^l_j	$\frac{\partial C}{\partial z^l_j}$	The error in the jth neuron in the lth layer.

temporarily to denote the loss function, $L(\cdot)$. A final addition to this notation is its matrix form, which is defined by simply removing the subscript j from relevant terms in Table 2.4. For instance, a^l represents the vector containing activations of all neurons in layer l.

Deriving the equations

Consider a single neuron, j, in the output layer, L. With the established notation and application of the chain rule, its error can be calculated as

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L),$$

and the corresponding matrix form for all neurons in the output layer would be

$$\delta^L = \nabla_{a^L} C \odot \sigma'(z^L), \tag{2.18}$$

where $\odot$ represents the Hadamard product (element-wise multiplication). It turns out that finding δ^L aids in finding the error in the previous layer:

$$\begin{aligned}\delta_j^{L-1} &= \frac{\partial C}{\partial z_j^{L-1}} \\ &= \sum_k \frac{\partial C}{\partial z_k^L}\frac{\partial z_k^L}{\partial z_j^{L-1}} \\ &= \sum_k \delta_k^L \frac{\partial z_k^L}{\partial z_j^{L-1}} \\ &= \sum_k w_{kj}^L \delta_k^L \sigma'(z_j^{L-1})\end{aligned}$$

Similar to δ^L, this can be represented on matrix form as

$$\delta^{L-1} = ((w^L)^T \delta^L) \odot \sigma'(z^{L-1}) \tag{2.19}$$

Errors for remaining layers can be found by *backpropagating* δ^L until it is calculated for all layers:

$$\begin{aligned}\delta^{L-1} &= ((w^L)^T \delta^L) \odot \sigma'(z^{L-1}) \\ \delta^{L-2} &= ((w^{L-1})^T \delta^{L-1}) \odot \sigma'(z^{L-2}) \\ &\vdots \\ \delta^{l} &= ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^{l})\end{aligned}$$

Using these errors, it is straight forward to explicitly distribute the loss across all weights and biases in the network:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} = \delta_j^l \tag{2.20}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1} \tag{2.21}$$

Vanishing gradients

Now it should clear why both the loss and activation functions are chosen to be differentiable, as it is essential to the backpropagation algorithm. These derivations also uncover some potential issues related to the choice of activation function. Equations 2.18 and 2.19, express the errors' dependencies on the first derivative of the activation function. Recall the function tanh (Figure 2.4c). If its input is sufficiently large, the output is on a near-flat slope and the corresponding derivative is near zero. A neuron with these characteristics is said to be *saturated*, and learns slow as a consequence.

The backpropagation algorithm

With the necessary notation and equations established, the backpropagation algorithm can be defined.

Algorithm 1: The backpropagation algorithm

```
1 Initialized ANN with random weights and biases;
2 Input: Set input layer activations $a_1$ with input data $x$.
3 Feedforward: For each layer $l \in \{2, 3, ..., L\}$, compute $z^l = w^l a^{l-1} + b^l$ and
    $a^l = \sigma(z^l)$.
4 Output error $\delta^L$: Compute $\delta^L = \nabla_{a^L} C \odot \sigma'(z^L)$.
5 Backpropagate the error: For each layer $l \in \{L-1, L-2, ..., 2\}$ compute
    $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
6 Output: The gradient of the cost function is given by
7 $\frac{\partial C}{\partial w^l_{jk}} = \delta^l_j a^{l-1}_k$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.
```

Output from Algorithm 1 is well suited for iterative updating schemes such as gradient descent. Equation 2.17 presented the issue of calculating the loss gradient, which now is solved by the introduction of backpropagation. The combination of these methods enables an arbitrary feed-forward ANN to iteratively reduce its error, and ultimately build a hypothesis that maps the input-output pairs. Although the elements of ANNs and how they are trained is covered, their shortcomings have not been addressed. In the beginning of this chapter, some of the issues were briefly mentioned, and are generally associated with *deep* ANNs. The next chapter will address the most common obstacles and how they are treated.

2.5 Deep learning

Deep learning is a term with no definitive root in literature, but is typically used to underline neural networks designed with a "large" number of hidden layers. A network that is considered "deep" today may change in the future. Deeper networks have the potential to solve more complex problems, although there is no way to determine the optimal topology of a network *a priori.* Increasing the amount of hidden layers results in multiple obstacles for training, one of which being the *curse of dimensionality* [3]. As the number of hidden layers increases, the number of adjustable parameters increases exponentially. The curse of dimensionality dictates that it is infeasible for an optimization problem to completely explore all states of a high-dimensional model given a finite number of training samples. While the strength of Artificial Neural Networks (ANNs) lies within their high-dimensional structure, it is also their curse. This is the reason why deep neural networks are notorious for requiring a large dataset and a lot of training to successfully solve their optimization problem. However, a large dataset is not the only solution, as a number of methods have been developed to reduce the number of parameters in a deep network.

Due to the recent raise in popularity of deep learning, there are many different network designs suited for different purposes. This thesis considers a particular network design consisting of two competing ANNs which will be presented in Chapter 2.6. First, some elemental methods needed to construct these networks are presented.

Convolutional layers

Up until this point, hidden layers in ANNs have been described as *fully connected*, or *dense*, layers where the output of one node in a hidden layer is connected to the inputs of all nodes in the next layer. In contrast, convolutional layers make connections in small, localized regions of the input. These *local receptive fields* enable Convolutional Neural Networks (CNNs) to take advantage of spatial structures in data. Representing layers as 2-dimensional matrices, rather than 1-dimensional vectors, makes it easier to illustrate this property. Figure 2.5 shows local receptive fields in a convolutional layer as a sliding 3x3 window across a 7x7 input. Each window connects the inputs it covers to a neuron in the hidden layer, indicating that the hidden neurons in this example have 9 weights and 1 bias. In this example, the window is said to have a *stride* of 2, as it moves 2 spaces each time. Stride length, along with the size of the local receptive fields are design choices that impact the network's topology and function. Padding the outer edges of the input can allow the widow to move outside the original input dimensions, as shown in Figure 2.6. Whereas hidden neurons in a fully connected layer have individual weights and biases, neurons in a convolutional layer *share* theirs. With shared parameters, one can interpret that all neurons in a hidden layer are looking for the same *feature* at different spatial locations in their input. CNNs are therefore able to take the translational invariance of data into account. For this reason, shared parameters are often called *filters* or *kernels*, and the filtered inputs of a local receptive field is called a *feature map.* A convolutional layer may have several filters looking for different features. Using multiple filters adds a 3^{rd} dimension, a *feature space*, to the output.

To limit the number of parameters in a CNN, convolutional layers are typically configured to produce an output with a smaller spatial size than its input. The

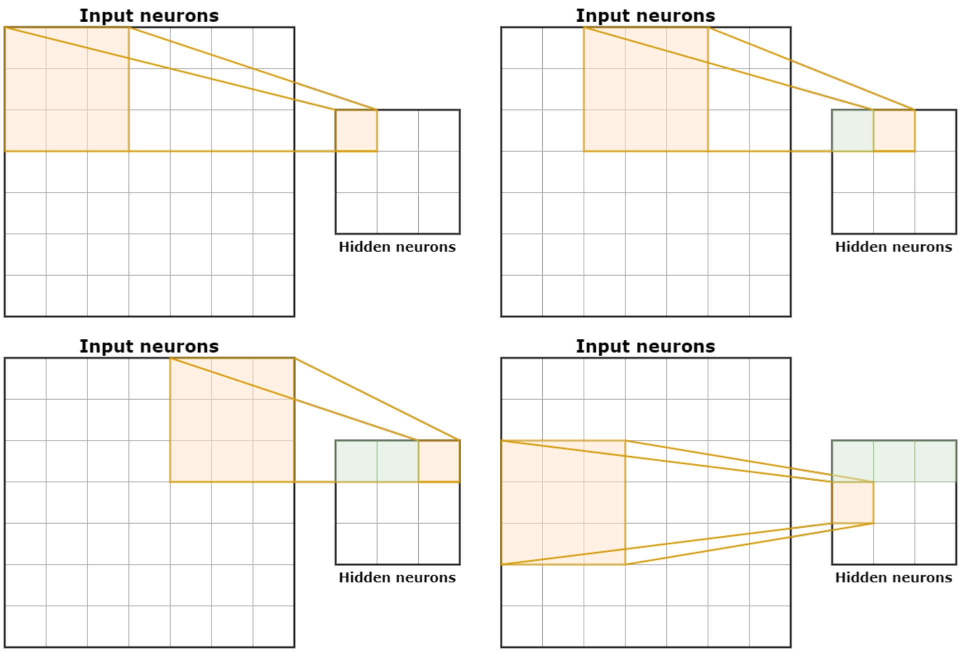


Figure 2.5: Local receptive fields in convolutional layers.

output size can be controlled to a certain extent:

$$O = \frac{W - K + 2P}{S} + 1,$$

where O is the output height/length, W is the input height/length, K is the kernel/filter size, P is the amount of padding and S is the stride length.

Pooling layers

Typically placed immediately after convolutional layers, the *pooling* layer summarizes the convoluted outputs without adding any trainable parameters. Filters in convolutional layers typically have some spatial overlap, as seen in Figure 2.5, which may result in neighbouring outputs having similar traits. Figure 2.7 illustrates how a max-pooling layer selects the highest activation in a 2×2 region, and discards the others. Information regarding the feature's exact location is blurred, but pooling significantly reduces the number of parameters in the network.

Transposed convolution

While convolutional layers typically reduce the spatial dimensions of their input, a transposed convolutional layer does the opposite. However, a transposed convolution is not equivalent to the inverse operation of a convolutional layer. Consider

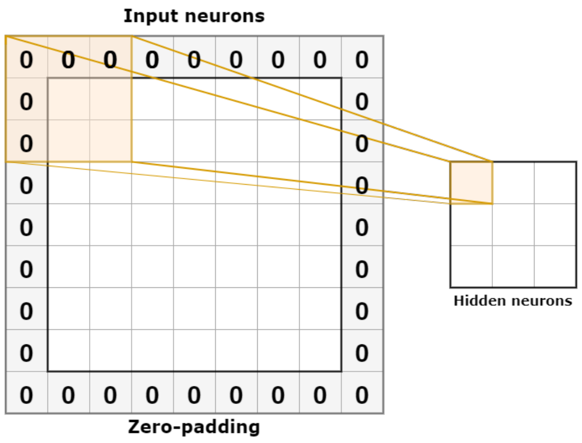


Figure 2.6: Zero-padding of size $P = 1$ in a convolutional layer.

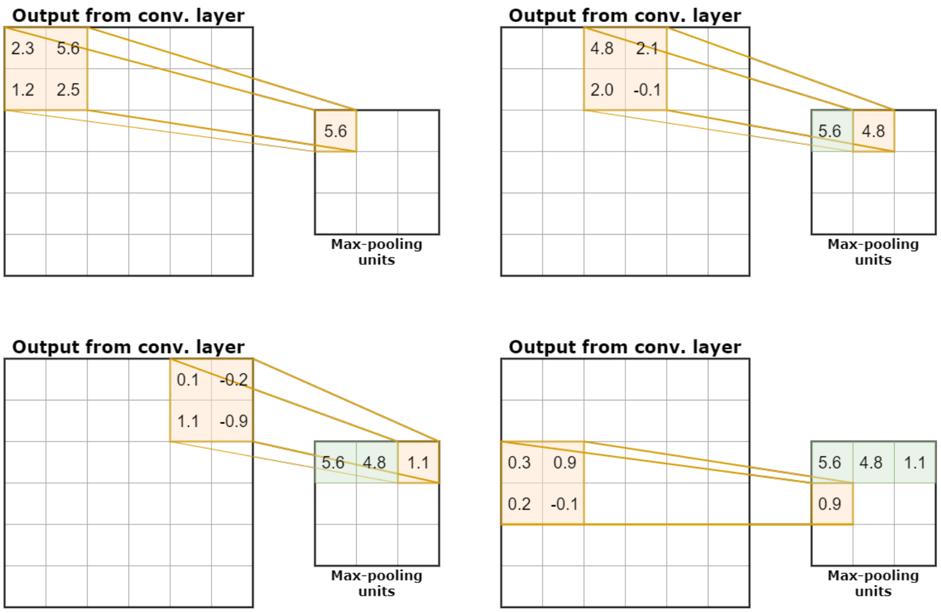


Figure 2.7: Max-pooling layer in CNNs.

Figure 2.8, which illustrates a transposed convolution of a 2x2 input to a 3x3 output. The operation is actually the same as in standard convolution, as seen in Figure 2.5, although the input layer is padded to mimic a larger spatial resolution. Here, a 2x2 input is padded in a specific way to form a 5x5 input. As a result, the convolution reduces the spatial dimensions from 5x5 to 3x3, although the original input was 2x2. Thus, a transposed convolutional layer can be considered like any other convolutional layer, without adding any complexity for backpropagation. This type of layer is useful for tasks such as image reconstruction/upscaling, or in the decoder in a encoder-decoder network. While there are plenty of fast, interpolation-based up-scaling algorithms available, they lack the ability to *learn* the means of interpolation from training data, which transposed convolution facilitates. For details on arithmetic and variations of transposed convolution, see Dumoulin & Visin [9].

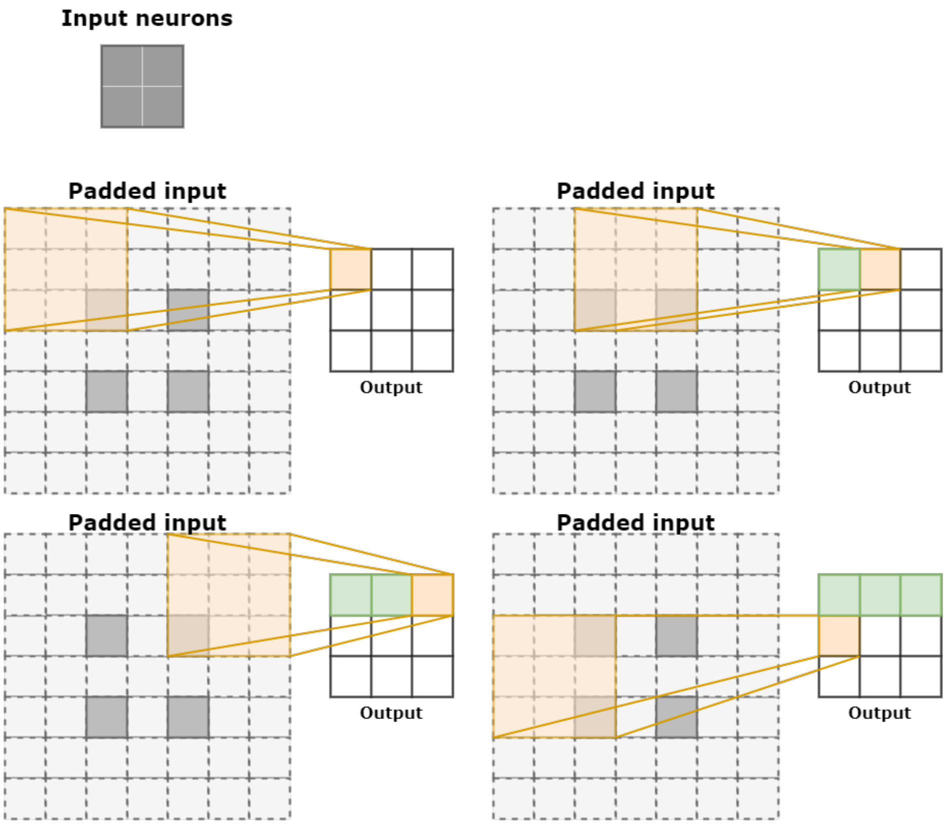


Figure 2.8: Transposed convolutional layer upscaling 2x2 input to 3x3.

Residual learning

In contrast to the hidden layer modifications mentioned earlier, residual learning adds only a simple skip connection to avoid the vanishing gradients problem, thus enabling very deep networks to be constructed. The author of this building block, He *et al.* [17], refers to several deep learning competitions in which residual learning-enabled networks are superior in a variety of applications, making it a formidable addition to the deep learning building block arsenal.

Consider the illustrated residual skip connection in Figure 2.9. Assume that $\mathcal{H}(x)$ is the true function to be approximated by the hidden layers. If one hypothesizes that

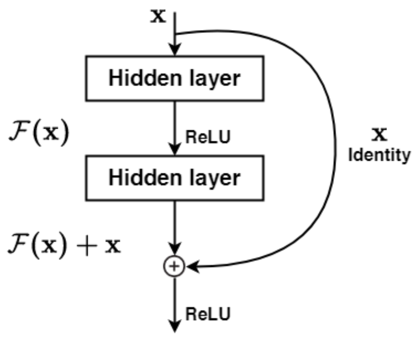

x
Hidden layer
$\mathcal{F}(\mathbf{x})$
ReLU
Hidden layer
x
Identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
ReLU

Figure 2.9: Residual learning: building block

the set of hidden layers can approximate $\mathcal{H}(x)$, then it is equivalent to hypothesize that they can approximate the residual function $\mathcal{H}(x) - x$. This is realized by letting the hidden layers fit the mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$. Regaining $\mathcal{H}(x)$ is simply done by adding skip connections as seen in Figure 2.9. Thus, the hidden layers' functionality remains the same, and the residual skip connection allows the backpropagated gradient to traverse the skip connections [17].

2.6 Generative Adversarial Networks

"Generative Adversarial Networks is the most interesting idea in the last ten years in machine learning."

Yann LeCun, Director, Facebook AI (2016)

The generative-adversarial framework is a relatively new addition to the deep learning arsenal, and provides an interesting solution to "supervise" an unsupervised learning problem. When Goodfellow introduced it in 2014 [11], CNNs had already reached proficiency in image classification [25, 23, 44]. In 2015, a contribution by Radford *et al.* [36] bridged the gap between CNNs models and their application in unsupervised learning tasks by supplementing Goodfellow's Generative Adversarial Network (GAN) with a Deep Convolutional GAN (DCGAN).

An intuitive analogy for generative-adversarial frameworks is a competition between a forger learning to print fake cheques and an investigator learning to distinguish real and fake cheques. Initially, the forger's fake cheques are horrible (imagine a pile of random notes next to the real cheques), and would thus be quite easy to distinguish. Luckily, it is also the investigator's first day on the job; they have no idea what a real cheque should look like. For each generation of fake and real cheques, the investigator learns to discard the odd-looking notes placed on their desk, and the forger is forced to try something else. After some time, the forger may realize that a certain kind of perforated paper works better, then carefully selected words, fonts and inks. It becomes increasingly hard to distinguish fake from real cheques, which forces the investigator to recognize very specific properties, such as valid number sequences for account numbers and signatures. This cycle may continue indefinitely. It can be proven that, given unlimited resources, the forger will end up generating real cheques, and thus the investigator will only have a 50% chance of correctly classifying a forged cheque.

Generative models solve *unsupervised learning* problems in attempts to generate samples of a high-dimensional target distribution. As mentioned in Chapter 2.3, unsupervised learning tasks lack a measure of success; evaluating the validity of a generated sample is a hard problem. Goodfellow *et al.* [11] proposed a novel framework in which a generative model is trained in an adversarial process. Figure 2.10 illustrates this framework. A generative model, G, generates "fake" samples of a distribution, $G(\mathbf{z}) = \mathbf{t_f} \sim T'$ using inputs from a latent space, Z. An adversarial model, D, receives both $\mathbf{t_f}$ and samples of the true distribution, $\mathbf{t_r} \sim T$, and estimates the probability, $D(\mathbf{t})$, of said samples originating from T. Thus, the generator's goal is to fool the discriminator into believing that the generated samples indeed belong to the true distribution; G attempts to find a mapping $Z \rightarrow T$. This competition can be modeled mathematically as a two-player minimax (zero-sum) game using a convex-concave value function, $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{t_r} \sim T} \left[\log D\left(\mathbf{t_r}\right)\right] + \mathbb{E}_{\mathbf{t_f} \sim \mathbf{T'}} \left[\log\left(1 - D\left(G\left(\mathbf{z}\right)\right)\right)\right] \tag{2.22}$$

However, this is only true if the investigator is able to learn optimally between each iteration of fake cheques.

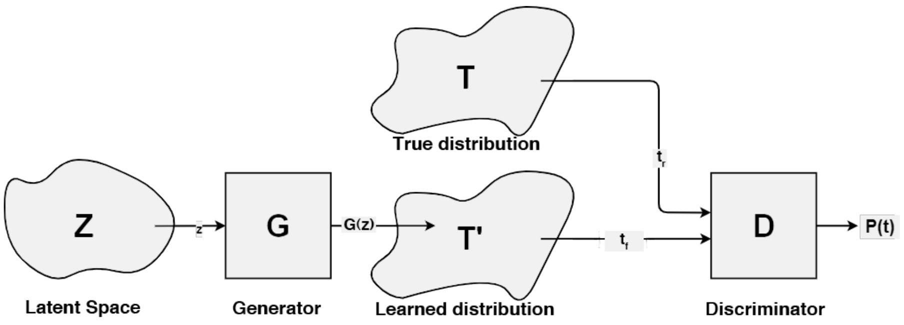


Figure 2.10: Visualized generative adversarial framework.

Thus, D is trained to maximize the probability of assigning the correct label on true and generated samples, while G is trained to minimize the probability of D assigning the correct label on generated samples. Corresponding loss functions are shown in Equations 2.23-2.24. D aims to maximize Equation 2.23, while G aims to minimize Equation 2.24.

$$L_D^{GAN} = \log(D(\mathbf{t_r})) + \log(1 - D(G(\mathbf{z}))) \tag{2.23}$$

$$L_G^{GAN} = \log(1 - D(G(\mathbf{z}))) \tag{2.24}$$

The GAN model converges when G and D reach a Nash equilibrium, which is the optimal point of Equation 2.22. Ideally, this point corresponds to D having a 50% chance of correctly classifying fake samples, coined as an *ideal discriminator.* Figure 2.11 illustrates a few, purely hypothetical, minimax game progressions. Initially, the losses are expected to have large initial fluctuations, as both G and D are learning from scratch. As both networks improve their performance, these fluctuations diminish and the losses hopefully converge. The losses are inversely correlated to a certain extent; since G and D have slightly different loss functions, ultimately parameterized by their individual weights and biases, their losses do change with respect to each other but tend to converge towards different values. Moreover, D solves a much simpler, supervised learning problem compared to G solving an unsupervised learning problem. As a result, D tends to experience smaller loss fluctuations and faster convergence. Overall, one would expect to see an initially low, but increasing discriminator loss before convergence, and the opposite for the generator loss. Uncertain convergence conditions make it hard to determine whether the model has achieved a good local minima or not. GAN's training stability is also particularly fragile; there are multiple modes of failure which may occur during training. Such modes can significantly impact the quality of generated samples and are not necessarily trivial to identify.

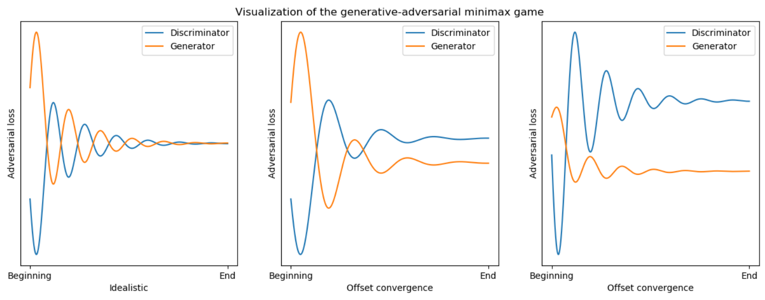


Figure 2.11: Hypothetical visualization of the GAN minimax game, with variations

Before the most common failure modes are presented, some details surrounding the training algorithm must be clarified. Algorithm 2 describes the original gradient descent-based training algorithm for GANs. The only mentioned hyperparameter, k, relates to one of the conditions described in Goodfellow *et al.* [11], that requires D trained to optimality in each iteration to satisfy their global optimality proof. The optimal discriminator D^*_G is defined for a *fixed* G:

$$D^*_G(\mathbf{t}) = \frac{P_T(\mathbf{t})}{P_T(\mathbf{t}) + P_{T'}(\mathbf{t})} \tag{2.25}$$

Substituting D^*_G into Equation 2.22 reformulates the minimax game for an optimal

Algorithm 2: Minibatch stochastic gradient descent training of generative-adversarial networks. **Hyperparameter:** $k \in \mathbb{N} \setminus \{0\}$ is the number of steps to apply to the discriminator. Rephrased from its definition in [11].

```
1  for number of training iterations do
2      for $k$ steps do
3          • Sample minibatch of $m$ noise samples $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from prior $Z$.
4          • Sample minibatch of $m$ examples $\{\mathbf{t}^{(1)}, ..., \mathbf{t}^{(m)}\}$ from data
             generating, true, distribution $T$.
5          • Update the discriminator by ascending its stochastic gradient:
6              $\nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\mathbf{t}^{(i)}\right) + \log\left(1 - D\left(G\left(\mathbf{z}^{(i)}\right)\right)\right)\right]$
7      end
8      • Sample minibatch of $m$ noise samples $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from prior $Z$.
9      • Update the generator by descending its stochastic gradient:
10         $\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} \left[ \log\left(1 - D\left(G\left(\mathbf{z}^{(i)}\right)\right)\right)\right]$
11 end
```

D, given a fixed G:

$$\begin{aligned}
\max_D V(D_G^*, G) &= \mathbb{E}_{\mathbf{t_r} \sim T}\left[\log D_G^*(\mathbf{t})\right] + \mathbb{E}_{\mathbf{t_f} \sim \mathbf{T'}}\left[\log\left(1 - D_G^*(\mathbf{t})\right)\right] \\
\max_D V(D_G^*, G) &= \mathbb{E}_{\mathbf{t_r} \sim T}\left[\log \frac{P_T(\mathbf{t})}{P_T(\mathbf{t}) + P_{T'}(\mathbf{t})}\right] \\
&\quad + \mathbb{E}_{\mathbf{t_f} \sim \mathbf{T'}}\left[\log \frac{P_{T'}(\mathbf{t})}{P_T(\mathbf{t}) + P_{T'}(\mathbf{t})}\right]
\end{aligned} \tag{2.26}$$

Equation 2.26 is maximized when $D_G^* = \frac{1}{2}$, which implies that $P_{T'} = P_T$. Rigorous proofs of this relation to Jensen-Shannon Divergence (JSD) and Kullback-Liebler Divergence (KLD) minimization are shown in Goodfellow *et al.* [11]. Thus, the hyperparameter, k, is intended to tune the *approximation* of an optimal discriminator between each iteration. Note that k influences the algorithm's time complexity $\mathcal{O}(n * (k + 1))$, where n is the number of training iterations and the number of network parameters are disregarded. Even a small increase in k will significantly increase the training time.

2.6.1 GAN failure modes and how to avoid them

In practice, training GANs seldom results in converging to an ideal discriminator, and is at the time of writing still an area of interest for research. Moreover, the proofs of convergence make some unrealistic assumptions, such as unlimited resources and training the discriminator to optimality between each iteration. The latent space, Z and the true data distribution, T, are represented as probability distributions in the convergence proofs. This implies that unlimited training samples are available, which is often not the case. Nevertheless, traversing an infinite set of examples is infeasible in practice. Additionally, only an approximation to an optimal discriminator can be achieved in each iteration of Algorithm 2. A higher value of the hyperparameter k could provide a better approximation, but will also drastically increase the training time. When the number of training iterations, n, is typically chosen in the order of 10^5, k is very expensive to increase.

As a result of these assumptions, GANs are often unstable in practice and require significant tuning to stabilize. Nevertheless, the generative model may still succeed in finding a sufficient mapping depending on its application. For instance, an application for art generation may not require a perfectly converged model to generate satisfactory results. Safety-critical or physics applications would naturally have stricter requirements for the model accuracy and consistency. This chapter covers observable symptoms, consequences and causes of the most common GAN failure modes, followed by recent and state-of-the-art methods developed to avoid these.

Vanishing gradients

Chapter 2.4.3 described the vanishing gradients problem as a consequence of deep neural networks or saturated activation functions. In a GAN context, the vanishing gradients problem is referring to a saturated discriminator output, leading to slow or halted generator learning. As mentioned, GAN's convergence proof assumes an optimally trained discriminator between each iteration. However, this may inhibit the generator's training; if the discriminator outperforms the generator significantly, then the generator may fail due to vanishing gradients from a saturated discriminator output [1]. Consider Equation 2.24. If D is extremely confident, and consistently classifies $D(G(\mathbf{z})) \approx 0.0$, then the generator's loss gradient is also small. Due to the significant impact of saturating discriminator outputs in GAN frameworks, recent contributions for loss functions are often specified as either *saturating* or *non-saturating.*

Mode collapse

In Algorithm 2, each updating step for D considers a fixed G, and vice versa. In G's pursuit of fooling D, it might find a particular output that appears particularly convincing. If D is not optimal, then it can assign a higher probability to certain modes of the true distribution. For instance, if the true distribution contains an equal number of cats and dogs, a non-optimal discriminator may assign a higher probability to true samples of cats rather than dogs. To maximize Equation 2.24, G produces more samples of higher-probability modes, and is said to overfit this particular iteration of D. Thus, G collapses a wide range of elements in Z to a few modes in T' capable of fooling D. In turn, the next iteration of a non-optimal D learns to reject that particular mode. As a result, the generator cycles through a

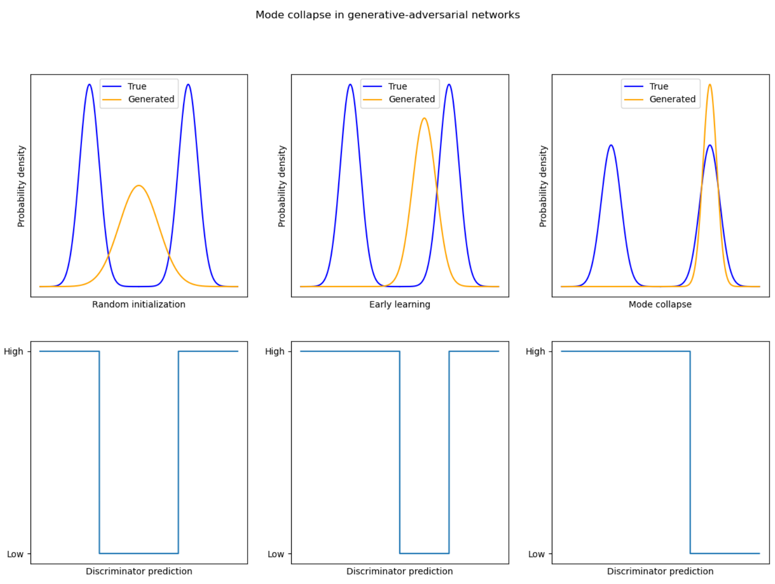


Figure 2.12: Hypothetical mode collapse in GANs.

small set of specific modes which often don't cover the entire distribution. Goodfellow originally named this behaviour after *the Helvetica scenario*, but is commonly called *mode collapse*. Figure 2.12 illustrates a 1-dimensional, bimodal probability distribution for which a hypothetical generator is trained to approximate. Initially (left), the generator is randomly initialized between the two modes. Then (middle), the generator may learn that a certain output resulted in a large ascent in its maximization problem, and moves its output toward this mode. Finally (right), the generator collapses all its samples into a single mode of the true distribution. Of course, the next discriminator iteration picks up on this mode collapse, and learns to reject that mode, even if samples may be drawn from the true distribution. Subsequently (not in figure), the generator finds the next mode, the discriminator rejects that mode and the cycle continues. Mode collapse is easy to illustrate for a 1-dimensional example, but can be hard to show for high-dimensional distributions. One exception is for computer vision tasks, where it is trivial to interpret whether the generator has collapsed into producing a single flower, face or animal. For distributions that cannot be easily visualized, mode collapse is harder to diagnose. Although mode collapse is traditionally attributed to a fault in the training algorithm; specifically, Goodfellow [10] describes that the Nash equilibrium of the minimax game in Equation 2.22 is not shared with the corresponding *maximin* game:

$$\max_D \min_G V(D, G),$$

where the generator now lies in the inner optimization loop, i.e., it is asked to map every point in Z to the sample that the discriminator believes to be the most realistic. Simultaneous gradient descent is particularly susceptible to this, as it does not differentiate between minimax and maximin. However, mode collapse could also be attributed to a lack of capacity in both or either network. At the time of writing, the described symptoms of mode collapse is generally understood, although a sulotion to fix it completely has not yet been found. Some recent modifications to the standard GAN training algorithms have improved the number of modes explored by the generator and will be introduced after the other failure modes are described.

Convergence failure

Algorithm 2 might never let Equation 2.22 converge to its Nash equilibrium. A traditional ANN application such as image classification has a simple task of minimizing a single loss function, e.g. classification error rate; GANs attempts to converge to its Nash equilibrium of two competing networks. This equilibrium can be imagined to lie on a saddle point on the joint loss functions, given the theoretic assumption that Equation 2.22 is convex-concave. Goodfellow [10] points out that even for a trivial value function, $V(x, y) = xy$, the gradient dynamics form a circular orbit around its Nash equilibrium. It can be shown that for an infinitesimal learning rate, the gradients will orbit the equilibrium infinitely. Moreover, a larger learning rate may sent the gradients spiraling out to infinity.

Nagarajan & Kolter [31] show that, under more realistic assumptions, the theoretically convex-concave minimax game in Equation 2.22 is concave-concave. Not only are concave optimization problems seldom stable, but this also suggests that the generator's minimization objective drives the model away from its equilibrium. Then again, these are observations in the game-theoretic function space; the competing networks are ultimately represented in parameter-space by ANNs that are certainly not convex themselves.

Mode collapse is actually one of the more harmful manifestations of non-convergence, as it represents an orbital rotation through a few modes of the true distribution, rather than converging. Another interpretable manifestation can be related to the discriminator's prediction confidence; as more realistic samples are generated, the discriminator's accuracy gets closer to chance level, making its feedback to the generator less meaningful, and eventually random when $D(t) = 0.5$. Thus, it is not uncommon to store intermediate states of the model in case the generator starts overfitting meaningless discriminator feedback [13].

A brief note on recent contributions

A plethora of modifications have been proposed both to alleviate failure modes and to expand on the original GAN's functionality. Although the mentioned failure modes are still relevant, the overall performance and stability of recent GAN implementations have improved drastically. Authors often term their contribution as “major modification”GAN-“auxiliary modification”, resulting in a jungle of different naming conventions either relating to the training algorithm, framework architecture or the input data conditioning. For instance, modifying a conditional GAN model with deep, convolutional neural network architecture using Wasserstein loss in the training algorithm with gradient penalty could be termed a “cDCWGAN-GP”. These abbreviations may become long, thus some authors resort to naming the model after the task it set to solve instead, such as SRGAN [27] for the computer vision super-resolution task. Different naming conventions can therefore make it troublesome to find relevant literature. GANs are becoming highly popular; multiple improvements for a wide variety of GAN-based applications are published each year, especially for computer vision tasks. Only a few, relevant and/or impactful contributions are mentioned here, but the reader is encouraged to explore the numerous GAN modifications available. Hereafter, the original GAN model will be referred to as “Goodfellow GAN”.

Training algorithm modifications

Goodfellow *et al.* proposed a *non-saturating* modification to their original Equation 2.22 to avoid vanishing gradients; instead of minimizing $\log(1 - D(G(\mathbf{z})))$, the generator could attempt to maximize $\log(D(G(\mathbf{z})))$ instead. Equations 2.27-2.28 show the non-saturating loss functions. L_D^{GAN} is unchanged, but $L_{G^*}^{GAN}$ is slightly different from Equation 2.24.

$$L_D^{GAN} = \log\left(D\left(\mathbf{t_r}\right)\right) + \log\left(1 - D\left(G\left(\mathbf{z}\right)\right)\right) \tag{2.27}$$

$$L_{G^*}^{GAN} = \log\left(D\left(G\left(\mathbf{z}\right)\right)\right) \tag{2.28}$$

While this modification might seem trivial, the resulting set of equations can no longer form the fundamental minimax game and is therefore only a heuristic approach. Although Equation 2.28 provides stronger gradients for the generator in early training, it also tends to make the model highly unstable. Arjovsky & Bottou [1] show that this non-saturating loss results in large, noisy updates in the generator’s gradient norms, unless the model has converged already. Furthermore, this behaviour is shown to drastically lower the quality of generated samples.

Arjovsky *et al.* [2] introduced Wasserstein GAN (WGAN): a GAN variant that is quite different from the Goodfellow GAN’s optimization problem. WGAN’s popularity has lead it to be a typical benchmark comparison against novel contributions and present in various stability analyses. Since WGAN is not directly applied in this thesis, only a brief introduction is presented. Instead of approximating a JSD minimization between the true and generated distributions, Arjovsky *et al.* propose a minimization of the Earth Mover’s Distance (EMD), i.e. the Wasserstein-1 distance, between them instead. This metric is shown to be an integral probability metric (IPM), which inherently enforces a 1-Lipschitz constraint on the model. Furthermore, they replace the discriminator with a *critic* that performs a regression rather than a binary classification. As a result, the WGAN critic can be trained to optimality in each iteration without risking the saturation-induced issues seen in Goodfellow GANs. However, to uphold a 1-Lipschitz critic, WGAN requires that its critic's parameters (weights and biases) have a magnitude restriction. In their original paper, this was enforced through weight clipping, e.g. forcing $-0.1 < w, b < 0.1$ after each parameter update. They emphasize that weight clipping is far from ideal to ensure Lipschitz continuity. Gulrajani *et al.* [14] improved this in their proposed WGAN with gradient penalty (WGAN-GP) by regularizing the critic parameters.

Arjovsky *et al.* [2] claims that mode collapse is impossible in WGANs, given the same assumption of training the critic to optimality. Similar to Goodfellow GAN, WGAN requires multiple critic updates per generator update. As mentioned for Goodfellow GANs, increasing the number of critic iterations per generator iteration results in a substantial increase in training time. The WGAN training algorithm (Algorithm 4 in Appendix) shows a slightly more specific training algorithm compared to Algorithm 2, as it specifies that the RMSProp optimizer should be used as opposed to a generic gradient-based optimizer in Goodfellow GAN. Arjovsky *et al.* [2] attributed events of training instability in WGAN to the use of momentum-based gradient optimizers such as Adam. Conversely, Gulrajani *et al.* [14] recommends implementing the Adam optimizer for WGAN-GP.

Finally, Jolicoeur-Martineau [21] proposes a family of Relativistic GANs (RGANs) and Relativistic Average GANs (RaGANs) that modifies the original training algorithm by targeting the discriminator's development in predicting the probability of real data being sampled from the true distribution. Towards the end of (successful) training in Goodfellow GANs, the discriminator ends up predicting that both real and fake samples as real. RGAN takes into consideration the *a priori* knowledge that half of the samples are indeed fake, ensuring that $D(\mathbf{t_r})$ decreases as $D((G(\mathbf{z}))$ increases. Jolicoeur-Martineau argue that this *relativistic* property results in a more accurate approximation to JSD minimization for non-IPM-based GAN models. Moreover, it can be shown that IPM-based GANs, such as WGAN, inherently possess the relativistic property and are a subset of RGANs as a result.

Jolicoeur-Martineau presents empirical evidence from computer vision experiments indicating RaGAN's improved stability compared to Goodfellow GAN and superior performance compared to WGAN. Furthermore, RaGAN with a gradient penalty outperforms WGAN-GP using only a single discriminator update per iteration, significantly reducing the amount of time needed for training. These results are promising, but purely empirical.

Only RaGAN is presented here since it is directly applied in Chapter 3; the reader is encouraged to explore the other variants proposed by Jolicoeur-Martineau [21]. The relevant loss functions are shown in Equations 2.29-2.30, and Figure 2.13 shows how they are envisioned to alter the discriminator's predictions during training compared to a Goodfellow GAN.

$$L_D^{RaGAN} = -\mathbb{E}_{\mathbf{t_r}\sim T}\left[\log\left(\tilde{D}(\mathbf{t_r})\right)\right] - \mathbb{E}_{\mathbf{t_f}\sim T'}\left[\log\left(1-\tilde{D}(\mathbf{t_f}))\right)\right] \tag{2.29}$$

$$L_G^{RaGAN} = -\mathbb{E}_{\mathbf{t_f}\sim T'}\left[\log\left(\tilde{D}(\mathbf{t_f})\right)\right] - \mathbb{E}_{\mathbf{t_r}\sim T}\left[\log\left(1-\tilde{D}(\mathbf{t_r}))\right)\right], \tag{2.30}$$

where

$\tilde{D}(\mathbf{t_r}) = \sigma\left(C(\mathbf{t_r}) - \mathbb{E}_{\mathbf{t_f} \sim T'} C(\mathbf{t_f})\right)$,
$\tilde{D}(\mathbf{t_f}) = \sigma\left(C(\mathbf{t_f}) - \mathbb{E}_{\mathbf{t_r} \sim T} C(\mathbf{t_r})\right)$,
and $C(\mathbf{t})$ is the preactivation of the discriminator output, $D(\mathbf{t}) = \sigma(C(\mathbf{t}))$. Yet, they do not address RaGAN's susceptibility to any failure modes.

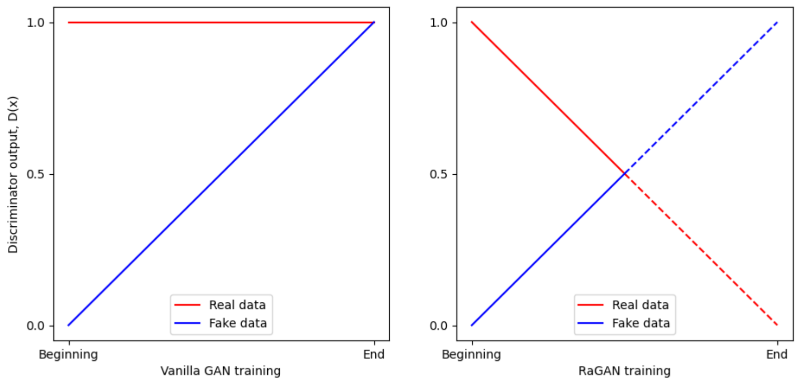


Figure 2.13: Goodfellow GAN vs. RaGAN discriminator prediction development during training. Plots reproduced from Jolicoeur-Martineau [21].

Input data conditioning

So far, details regarding the latent space, Z, have been lacking. In a standard GAN configuration, Z is typically chosen to be a wide Gaussian probability distribution on an arbitrary manifold. Mirza & Osindero [30] propose a Conditional GANs (cGANs) with additional input layers in both the generator and discriminator to enable the model to learn a multi-modal model. By conditioning inputs with label data, y, to the generator and discriminator, a cGAN is able to generate specific modes of a distribution. The minimax Equation 2.22 is slightly modified to accommodate conditional terms $D(\mathbf{t_r}|\mathbf{y})$ and $1 - D(\mathbf{t_f}|\mathbf{y})$. For instance, Goodfellow GANs could learn to generate fake MNIST [26] samples, but one cannot control the generator to generate specific digits. cGAN includes a one-hot encoded digit label as input, and the corresponding digit sample will be generated.

GAN training "hacks"

The summary of the NIPS 2016 Tutorial [10] for training Goodfellow GANs describe multiple tips, or "hacks", for improving training stability and performance. Some relevant methods are presented here, but note that these are mostly based on empirical evidence and heuristics. The tutorial itself underlines that the following methods have shown to be helpful in some contexts, and hurtful in others. A contribution by Salimans *et al.* [42] contain similar tips and tricks.

- **One-sided label smoothing:** intends to inhibit the discriminator from producing extremely confident predictions. Especially early on, the discriminator may become highly confident in its predictions, potentially resulting in vanishing gradients for the generator. To combat this, Salimans *et al.,* Goodfellow ([42, 10]) suggest *smoothing* the labels (y_i in Chapter 2.3.1) fed to the discriminator from "hard" labels $l_{real} = 1.0, l_{fake} = 0.0$ to $l_{real} = 0.9, l_{fake} = 0.0$ for instance. Both authors explicitly advise against smoothing labels for fake data, with the argument that the model should never encourage choosing an incorrect class, but only reduce the confidence in the correct class. This can be shown by adjusting the optimal discriminator Equation 2.25 with the label smoothing terms α and β for smoothing real and fake labels, respectively.

$$D_G^*(\mathbf{t}) = \frac{(1-\alpha)P_T(\mathbf{t}) + \beta P_{T'}(\mathbf{t})}{P_T(\mathbf{t}) + P_{T'}(\mathbf{t})} \tag{2.31}$$

  Equation 2.31 shows how label smoothing affects the optimal discriminator. When β is zero, the only effect is that the optimal discriminator value is scaled down. If β is nonzero, the dynamics of the optimal discriminator changes. For instance, if $P_T(\mathbf{t}) < P_{T'}(\mathbf{t})$, the discriminator will rather encourage the generator to generate samples that resembles the input data rather than true samples.

- **Noisy labels**: typically refers to adding a small random chance of the labels to flip, i.e, $l_{real} = 0.0, l_{fake} = 1.0$. In this thesis, noisy labels will rather refer to an addition to the label smoothing method. Noisy labels adds stochasticity to label smoothing. A small amount of noise, $\eta \sim \mathcal{N}(0, 0.05)$, is added to the smoothed labels. The resulting labels are clipped to stay within the range $0.0 \leq (l_{real} + \eta, l_{fake} + \eta) \leq 1.0$.

It is worth noting that these hacks were suggested for the Goodfellow GAN model, and might not necessarily be applicable for more recently proposed GAN modifications. For instance, RaGAN inherently suppresses the discriminator's prediction confidence; inhibiting the discriminator further through label smoothing might not contribute towards model stability. Due to the heuristic nature of these hacks, experimentation is required to assess their impact on a specific model and task.

Sønderby *et al.,* Arjovsky & Bottou [46, 1] suggested to add a small *instance noise*, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, to real and fake samples before input to the discriminator. The authors show that if T and T' are disjoint, then there is always a perfect[2] discriminator between them. Instance noise is added to make the probabilistic divergence (e.g. JSD) between the T and T' well-defined. Like label smoothing, adding instance noise changes the optimal discriminator:

$$D_G^*(\mathbf{t}) = \frac{P_{T+\epsilon}(\mathbf{t})}{P_{T+\epsilon}(\mathbf{t}) + P_{T'+\epsilon}(\mathbf{t})} \tag{2.32}$$

[2]Not to be confused with an *optimal* discriminator

During training, ϵ is annealed gradually such that the optimal discriminator returns to Equation 2.25 toward the end of training. While this addition resembles other heuristic hacks, is has been backed in a GAN convergence study by Mescheder *et al.* [29]. Since instance noise target divergence minimization in the fundamental GAN framework, this method might be more applicable to more recent models compared to label smoothing.

Summary

Since this chapter covers a wide range of GAN contributions, a short summary of the most relevant information is provided.

- GANs are notoriously hard to train, and can often suffer from vanishing gradients, mode collapse and convergence failure.
- WGAN introduces a new minimization task with an IPM (EMD) that enforces a 1-Lipschitz constraint on the model. The authors claim that WGAN cannot suffer from mode collapse if the discriminator is trained to optimality, which still requires a significant increase in total training time.
- RGAN and RaGAN better approximate JSD minimization in practice by utilizing the *a priori* knowledge that half of the samples fed to the discriminator are fake. The author showed how IPM-based GAN models are inherently relativistic and how the relativistic property can be added to any non-IPM-based GAN model. They also presented empirical evidence of RaGAN outperforming WGAN while only requiring a single discriminator update per generator update. RaGAN is directly implemented in this thesis.
- cGANs use conditional labels to generate specific modes of a target distribution.
- Label smoothing aims to avoid the vanishing gradients problem by making the discriminator less confident in its predictions on true samples. Smoothing fake labels may change the dynamics of the optimal discriminator (Equation 2.31) adversely.
- Noisy labels typically refers to randomly flipping labels of real and fake data. In this thesis, it refers to adding stochasticity to the label smoothing hack.
- Instance noise makes the JSD minimization task well-behaved by altering the (possibly) disjoint true and approximated distributions by adding an annealing Gaussian noise to the true and generated samples.

2.7 Single Image Super-Resolution GANs

Single-Image Super-Resolution (SISR) describes the classic computer vision task of recovering a High-Resolution (HR) image from a single LR image. LR images are produced by downsampling High-Resolution (HR) images:

$$I^{LR} = f(I^{HR}),$$

where f represents the downsampling function. The SISR task is then to find an inverse mapping, $g \approx f^{-1}$ such that LR images can be used to approximately reconstruct their corresponding HR images, denoted Super-Resolution (SR) images:

$$I^{HR} \approx I^{SR} = g(I^{LR}) = f^{-1}(I^{LR}) + R,$$

where R is a residual. This inverse problem is under-determined due to the amount of information loss during downsampling and the lack of any governing equations that model inter-pixel relations. Therefore, there is no unique one-to-one mapping to recover any HR image from LR [19].

Many interpolation-based super-resolution algorithms exists. Although these are simple and fast algorithms, they also suffer from significant blurring and/or artifacting in the SR image, especially when the image contains high frequency or complex details. In contrast, ANNs have shown great potential to recover plausible SR images from LR in recent years. CNN-based SISR was first introduced by Dong *et al.* [8], whom introduced the Super-Resolution Convolutional Neural Network (SRCNN). Their method produces SR images of significantly higher Peak Signal-to-Noise Ratio (PSNR) compared to traditional methods, and marks the start of deep learning-based image super-resolution. However, the SRCNN utilizes a Mean Squared Error (MSE) based loss function (Equation 2.16 in Chapter 2.4), which has the intrinsic property of averaging the error in the entire image. Equations 2.33-2.34 describe PSNR and MSE in a SISR-context.

$$PSNR = 10 * \log_{10}(\max_I^2) - 10 * \log_{10}(MSE) \tag{2.33}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(I_i^{HR} - I_i^{SR})^2 \tag{2.34}$$

where the maximum fluctuation term, $\max_I$, is defined as the maximum possible pixel-value in the image; for images with 8 bits per color channel, this value is $2^8 - 1 = 255$. Quantifying a meaningful difference between two images is a hard problem. While the PSNR metric has been used extensively, it is based on MSE. PSNR has been shown to suffer from large variations even for indistinguishable distortions, rendering it a non-ideal metric [52].

A brief note on digital image processing

There are statistical dependencies among pixels in an image due to objects' geometrical or structural properties, especially for neighboring pixels. This *inter-pixel redundancy* is one of the bases for image compression algorithms, and conversely enables pixel prediction based on its neighbors. Furthermore, some visual information represented in an image is more significant for human perception than others. For instance, contrast (brightness change) contributes significantly more towards perception than fine details or gradual changes in color. The latter properties are examples of *psychovisual redundant* information, and are often targeted by image compression algorithms. Removing psychovisual redundant information from an image does result in a quantitative loss, e.g, measurable MSE between the original and compressed image, although the perceived difference is negligible or unnoticeable [48].

2.7.1 Introducing perceptual loss

Significant improvements to ANN-based SISR were made by Johnson *et al.* [20], introducing a deep convolutional **perceptual loss** for a residual CNN-based style-transfer and super-resolution. Their perceptual loss uses ReLU activations at multiple locations in the convolutional part of the VGG16 [44] network pre-trained on ImageNet [6]. Further SISR improvements were made by Ledig *et al.* [27], who applied a similar perceptual loss in their proposed Super-Resolution Generative Adversarial Network (SRGAN). However, they used a deeper network, VGG19, for their perceptual loss, extracting the ReLU activations of VGG19-54[3]. Finally, Wang *et al.* [50] proposed an Enhanced Super-Resolution Generative Adversarial Network (ESRGAN) model that improves upon the SRGAN model by slight alterations in perceptual loss, implementing a relativistic discriminator (RaGAN), and proposing a novel Residual-in-Residual Dense Block (RRDB) without batch-normalization. In fact, ESRGAN achieved 1^{st} place in the 2018 PIRM Challenge on Perceptual Image Super Resolution [4].

Figure 2.14 illustrates the framework of a SISR task in a GAN context. Note that the main difference between this and the Goodfellow GAN framework shown in Figure 2.10 is the latent space, Z. Now, Z is chosen to be the set of LR images, I^{LR}, found through applying a downsampling function, f, on the HR images, I^{HR}. Recall the description of supervised vs. unsupervised learning in Chapter 2.3. GANs were introduced as a framework that typically accommodate unsupervised learning tasks; SISR utilizes a content-aware difference between a generated SR image and its corresponding HR ground truth image. This difference is embedded in the generator's optimization problem, for instance in the form of a perceptual loss as described above. Therefore, one can consider GAN-based SISR as a supervised, high-dimensional regression task in an unsupervised framework.

2.7.2 The Learned Perceptual Image Patch Similarity metric

Deep perceptual features have not only been applied to loss functions in GAN-based SISR tasks; the Learned Perceptual Image Patch Similarity (LPIPS) was introduced by Zhang *et al.* [52] for evaluating similarity between images. They show that traditional metrics such as PSNR and SSIM often disagree with human judgement of image similarities, and propose a novel perceptual metric based on

[3]VGG19-54 denotes the activations of the 4^{th} convolutional layer before the 5^{th} max-pooling layer of a pre-trained 19-layer VGG network [44]

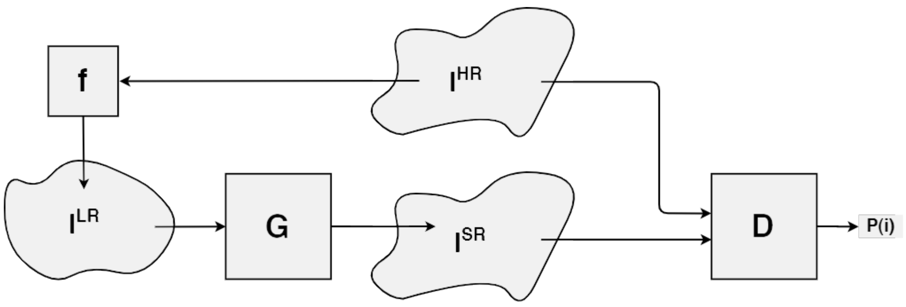


Figure 2.14: Single-Image Super-Resolution task in a generative-adversarial context.

feature distances in light-weight CNNs. This is somewhat similar to the VGG-based perceptual losses described in Chapter 2.7.1, although LPIPS conserves the spatial dimensionality of its inputs. This allows feature differences to be spatially correlated with the input, i.e. one can visualize where and how severely the images differ from each other.

- Feed reference and distorted image, x, x_0 into network $\mathcal{F}$.
- Extract feature maps from multiple layers, L, and unit-normalize in feature-space, defined as $\hat{y}^l, \hat{y}^l_0 \in \mathbb{R}^{H_l \times W_l \times C_l}$ for layer $l \in L$.
- Scale activations by vector $w_l \in \mathbb{R}^{C_l}$ and compute the L_2 distance.
- Average L_2 distances spatially and sum in feature-space.

The L_2 distance is defined as:

$$L_2(\mathbf{x}, \mathbf{y}) = \sum_i \left((x_i - y_i)^2 \right)$$

Ultimately, the LPIPS metric is defined as:

$$d(x, x_0) = \sum_l \frac{1}{H_l W_l} \sum_{h,w} ||w_l \odot (\hat{y}^l_{hw} - \hat{y}^l_{0hw})||_2^2 \tag{2.35}$$

Note that spatially localized feature differences can be obtained by not computing the spatial average of L_2 distances. CNNs used for this metric are pre-trained AlexNet [24], VGG [44] and SqueezeNet [18] architectures. The authors show "the unreasonable effectiveness" of using such deep features as a perceptual metrics, with significantly better correlation with human judgement. Moreover, a distance-based metric provides a lower bound where the images are identical, in contrast to the unbounded PSNR metric.

2.7.3 Enhanced Super-Resolution Generative Adversarial Network for airflow velocity data

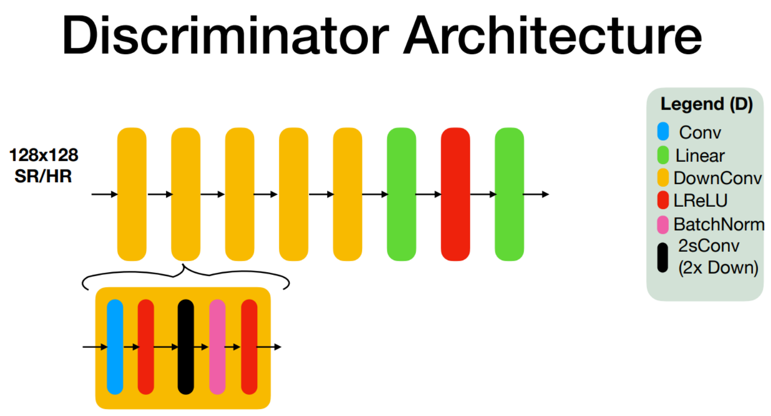
Chapter 2.6 presented how GANs can approximate high-dimensional distributions using a ANN-based generative-adversarial approach. The previous chapter showed how such models are used in SISR tasks to reconstruct HR images from LR, and how SISR performance has improved following the introduction of deep perceptual features. This thesis considers an ESRGAN model, modified for application on 3-dimensional (u,v,w) airflow velocities spanning a 2-dimensional horizontal grid. This data structure is similar to RGB images, as they contain 3 color channels (r,g,b) spanned across a 2-dimensional bitmap. Real-world data collected from the coupled HARMONIE-SIMRA system (Chapter 2.2) is used to train the model. Work done by Tran *et al.* [47] shows promising results when applying the ESRGAN architecture for reconstruction of HR velocity fields from LR at a scale of both 2x and 4x. However, they also raise doubts related to the applied quality metric, statistical confidence and uncertainty in the model output with respect to safety critical applications. These issues are addressed and quantified in this thesis to further development of the model. Thus, the modified ESRGAN is presented here, as implemented by Tran *et al.* [47] and Vesterkjær [49].[4]

Architecture

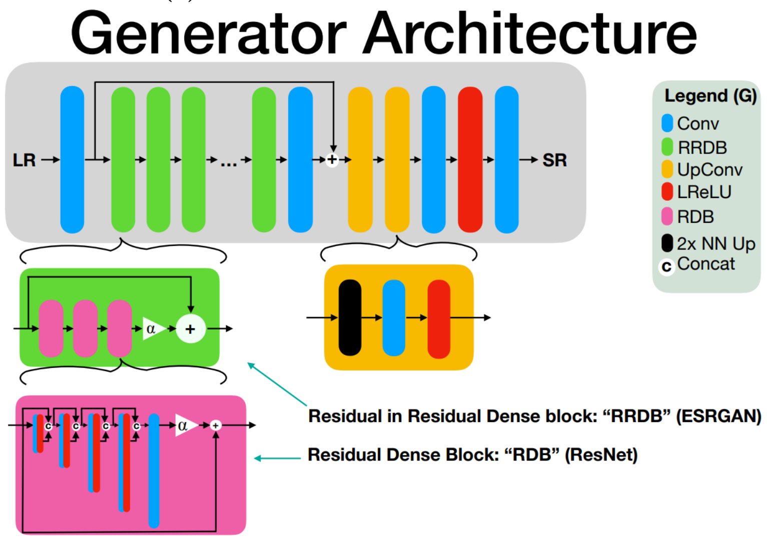
Figure 2.15 shows the architecture of the discriminator and generator. The generator architecture consists of an initial convolutional layer, 16 consecutive Residual-in-Residual Dense Blocks (RRDBs), two convolutional upsampling blocks, and two 1x1 convolutional layers with 5x5 filters at the output. These 1x1 convolutions functions as a feature space pooling while maintaining the spatial dimensions. The discriminator is a typical VGG structure without max-pooling layers. It consists of 5 consecutive convolutional blocks, where the number of 3x3 filters are doubled after each block, increasing from an initial 128 filters to 1024 filters, while the spatial resolution is similarly halved from 128x128 to 4x4. Finally, the discriminator's classifier is constructed from two dense layers. The first layer reduces the 1024x4x4 input vector to 100, and the second layer reduces these to a single output value. All layers in both the generator and discriminator use the LeakyReLU activation function, except for the output layers. In a Goodfellow GAN model, the discriminator's output layer would use a sigmoid activation function to represent the binary classification probability, but this activation is encapsulated in the adversarial loss function.

An auxiliary feature extractor network is used to calculate a perceptual loss between the reference and super-resolved velocity fields, which is added to the generator's loss function. With a similar structure as the discriminator, the feature extractor consist of the convolutional part of a VGG19 network, shown in Figure 2.16. To keep the figure compact, ReLU activations between consecutive convolutional layers

[4]Some discrepancies were found between the documented work and the Python implementation obtained from said author. This thesis describes their model as implemented in their original code. As a result, the model description will deviate somewhat from theirs.



(a) ESRGAN discriminator acrhitecture.


Generator Architecture
LR
SR
...
α
+
c
Legend (G)
Conv
RRDB
UpConv
LReLU
RDB
2x NN Up
c Concat
Residual in Residual Dense block: "RRDB" (ESRGAN)
Residual Dense Block: "RDB" (ResNet)

(b) ESRGAN generator architecture.

Figure 2.15: ESRGAN archtecture.
Source: Vesterkjær, [49]

are not shown. Features are extracted from VGG19-54, which corresponds to the activations of the 4^{th} convolutional layer before the 5^{th} max-pooling layer. The feature extractor is not trained along with the generator and discriminator; it is pre-trained on the ImageNet dataset [6] and standardizes (subtracts mean and divide by standard deviation) its input data channel-wise according to precalculated means and standard deviations from ImageNet.

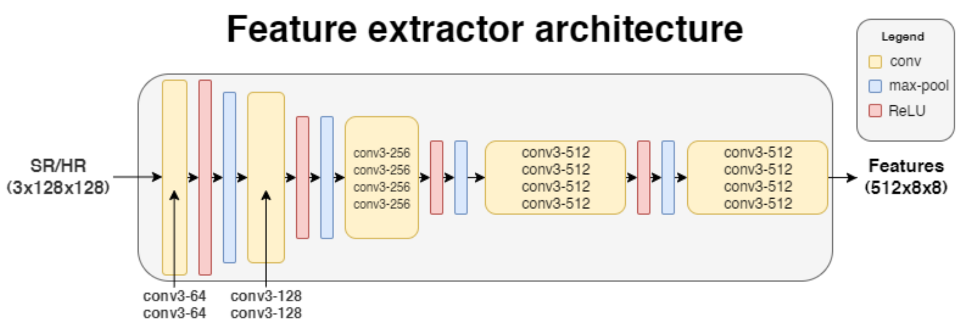


Figure 2.16: ESRGAN feature extractor network. Although not depicted, there are ReLU activations after each convolution.

Loss Functions

A relativistic average discriminator is implemented, using Equations 2.29-2.30 to compute the adversarial losses. While the discriminator is trained purely on adversarial loss, the generator utilizes the weighted sum of two additional loss functions; perceptual loss is found by calculating the L_1 distance between the generated SR and true HR images' feature activations of VGG19-54, and pixel (content) loss is calculated as the L_1 distance between the corresponding SR and HR images.

$$L_1(\mathbf{x},\mathbf{y}) = \sum_i |x_i - y_i| \qquad L_1 \text{ distance} \tag{2.36}$$

$$L_1^{Pixel} = L_1(I^{HR}, I^{SR}) \qquad \text{Pixel loss} \tag{2.37}$$

$$L_1^{Perceptual} = L_1(F(I^{HR}), F(I^{SR})) \qquad \text{Perceptual loss} \tag{2.38}$$

$$\tag{2.39}$$

where $F(\cdot)$ is the VGG19 feature extractor activations for layer VGG19-54. This implementation is equivalent to the original ESRGAN model, and the resulting loss functions can be modeled as:

$$L_D^{ESRGAN} = L_D^{RaGAN} \tag{2.40}$$

$$L_G^{ESRGAN} = L_1^{Perceptual} + \lambda L_G^{RaGAN} + \eta L_1^{Pixel}, \tag{2.41}$$

where $\lambda = 0.005$ and $\eta = 0.01$ discount the adversarial and pixel losses, respectively. Note that the perceptual loss is weighted 200 times more than the adversarial loss, and 100 times more than the pixel loss. Wang *et al.* [50] denotes the entire Equation 2.41 as their "perceptual loss". In this thesis, "perceptual loss" refers only to the term $L_1^{Perceptual}$ in the same equation.

Training algorithm

Algortihm 12 shows a simplified Both the generator and discriminator use Adam optimizers to backpropagate loss gradients and update their network parameters

[22]. The optimizers are linked to a multi-step learning rate scheduler which lowers the learning rate after the model has reached specified milestones during training. This is also similar to the ESRGAN model, although the learning rate schedule's frequency is modified. Furthermore, the discriminator is set to train twice as often as the generator. Unlike ESRGAN, this model does not consider any pixel loss-based pre-training of the generator. Specific hyperparameters are shown in Chapter 3.

Algorithm 3: Simplified ESRGAN training algorithm for airflow data, modified from Algorithm 2

```
1  for number of training iterations do
2      for k steps do
3          • Sample minibatch of paired HR and LR velocity fields.
4          • Generate paired HR and SR velocity fields by passing LR data
             through the generator.
5          • Update the discriminator by ascending its stochastic gradient:
6                      $\theta_D \leftarrow \nabla_{\theta_D} L_D^{RaGAN}$
7      end
8      • Update the generator by descending its stochastic gradient:
9          $L_G^{ESRGAN} = L_1^{Perceptual} + 0.005 \cdot L_G^{RaGAN} + 0.01 \cdot L_1^{Pixel}$
10                     $\theta_G \leftarrow \nabla_{\theta_G} L_G^{ESRGAN}$
11 end
12 .
```

Chapter 3

Methods

This chapter describes the equipment and methods used to achieve the results presented in Chapter 4. Three distinct experiments are presented:

I Stability analysis of previous work on ESRGAN with airflow.

II Assessing the validity of applying a perceptual SISR method to airflow data.

III High-altitude airflow reconstruction with ESRGAN.

Hereafter, these experiments will be addressed as Experiment I, II, and III, respectively.

3.1 Hardware specification

As mentioned in Chapter 1, the recent development in computational hardware is one of the key factors for the viability of executing deep learning algorithms. However, this does not mean that any modern computer is able to execute such an algorithm. The arguably most essential tool is the GPU, which is not necessarily present in every computer. Some of todays mid- to high-end GPUs are capable of realizing ANNs, but their capacity is limited as a result of being directed at consumers. In order to facilitate deep neural networks and large datasets, it is necessary to utilize high-capacity General Purpose GPUs (GPGPUs) intended for industrial purposes.

All models in Experiment I were run on the NTNU IDUN computing cluster [45]. IDUN is an ongoing cooperative High Performance Computing (HPC) project between the faculties at NTNU and its IT division, created for the purpose of rapid testing and prototyping HPC software. At the time of writing, the cluster consists of more than 70 nodes and 90 GPGPUs. Half of the nodes are equipped with at least two Nvidia P100 or V100 CUDA-compatible GPGPUs. Each node contains two Intel Xeon cores, 128-768GB DDR4 RAM and is connected to an Infiniband network.

A single node using one GPGPU on this cluster is sufficient to train the models in Experiment I and III, although the real-time training duration could be improved if parallel-computing principles were implemented (i.e. using multiple GPUs or even nodes simultaneously). Experiment II can be run on virtually any computer.

3.2 Software specification

Two different softwares are used for visualizing collected data in this thesis. The Python library “matplotlib” is used for most 2D visualizations, including all plots in Experiment I, and RGB representations in Experiment IIa. The “Mayavi” application was used through their Python scripting API “mlab” for 2D and 3D vector field visualizations and feature activation maps in Experiment II and III [37]. The neural network architecture is implemented in Python 3.7.2, using the deep learning library PyTorch [34].

3.3 Data generation process

This chapter describes the initial data preprocessing shared across all experiments using airflow data, unless specified otherwise. Airflow data used in this thesis originates from the coupled HARMONIE-SIMRA system (Chapter 2.2) and was downloaded from the THREDDS web server [15] using a Python script. Two files are generated per day, each containing hourly averaged data for the time intervals 00:00-12:00 and 12:00-24:00, respectively. Consequentially, there is 1 hour of overlap between each subsequent file. The files are generated in the Network Common Data Form (NetCDF), for which the Python library netcdf4 was used to load the files' content. Specific variables utilized from the data are presented in Table 3.1 and the full list of variables is shown in Table B.1.

Table 3.1: Relevant variables in the netCDF files.

Variable	Descripton	Unit
x_wind_ml	Wind velocity along x-axis (u)	[m/s]
y_wind_ml	Wind velocity along y-axis (v)	[m/s]
upward_air_velocity_ml	Wind velocity along z-axis (w)	[m/s]
geopotential_height_ml	Geopotential height	[m]

All variables in Table 3.1 share the same dimensions, (time, l, y, x) = (13, 41, 136, 135), as dictated by the domain specified in Chapter 2.2. Intuitively, the three latter dimensions (l, y, x) represents the 3-dimensional geographic domain covered by the HARMONIE-SIMRA system. Thus, the set of wind velocity variables, (u,v,w), represent the simulated 3-dimensional velocity field within this domain. The first dimension (time) represents what time of day the simulated velocity field corresponds to. Date information is stored in each filename. Figure 3.1 illustrates a randomly sampled velocity field from this data. Note that the axes are scaled for visualization, and do not accurately represent the spatial dimensions of the SIMRA domain.

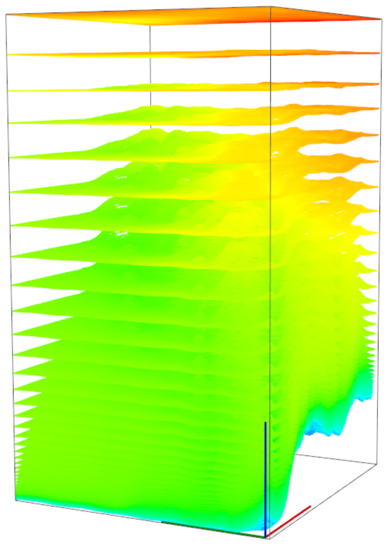
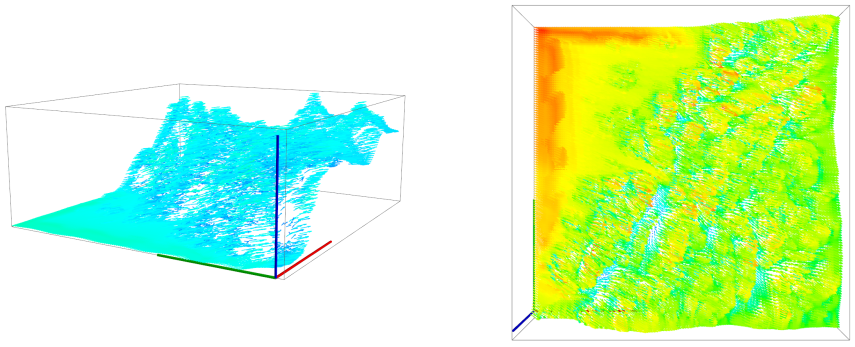


Figure 3.1: Full 3D velocity field sampled from the HARMONIE-SIMRA coupled system.

All experiments consider datasets consisting of velocity fields sliced from a horizontal 2-dimensional sub-space of the domain, with respect to the selected geopotential height over the surface. Hereafter, the term “sample” refers to such a horizontal velocity field sampled at a point in time, collected from data in the NetCDF files. This is visualized in Figure 3.2. Moreover, to conform with the nomenclature associated with neural networks, the velocity components (u, v, w) in a sample will be referred to as “channels” of the sample.

Before shaping the samples for input into a neural network, they must undergo some preprocessing as they were stored as “masked arrays” (numpy.ma.MaskedArray), which are not directly compatible with the implemented neural network model. These arrays contain masks to indicate invalid values in the dataset, likely due to errors or failures in the simulator. A “true” mask indicates the presence of an invalid value; conversely, a “false” mask indicates a valid entry. Coincidentally,



(a) The effect of geopotential height. Same colormap scaling as in Figure 3.1.

(b) Top view. Rescaled colormap.

Figure 3.2: Sliced velocity field from the bottom of the 3D domain of Figure 3.1.

the majority of the masked invalid data were present near the domain boundaries. These were avoided by truncating the edges along the x and y axes, resulting in the new dimensions being $(l = 41, y = 128, x = 128)$. Further, the resulting dimensions for x and y correspond to the dimensions used in the original ESRGAN paper [50]. A few unmasked invalid values were present in the center of the domain. These were ignored by enforcing a maximum threshold value for valid entries. Since the relevant variables contain wind speeds, this threshold was set to 100 [m/s] to accommodate for potential strong winds. Observed unmasked invalid entries tend to have values larger than 10^{22}[m/s], although the minimum observed (invalid) value is 280[m/s]. Samples ignored this way comprise 0.5% of the available data.

3.3.1 Preprocessing and splitting of data

Initially, the coarse Low-Resolution (LR) data must be generated by downscaling the fine High-Resolution (HR) data. This is done by interpolating the HR data using the nearest neighbor algorithm, implemented in the PyTorch library torch.nn.functional.interpolate. This yields HR and LR datasets with resolutions of 128×128, and 32×32, respectively. As described in Chapter 2.3, these datasets are split into training, validation and test sets at ratios of 0.8/0.1/0.1, respectively. The model is only allowed to use the training dataset for learning, and its performance on the validation dataset is used to tune model hyperparameters between training sessions. Only after hyperparameter tuning is concluded is the model's performance evaluated on the test set.

Finally, the datasets are normalized (centered and scaled) by the linear transformation in Equation 3.1. This equation allows for modifying the range of values in the

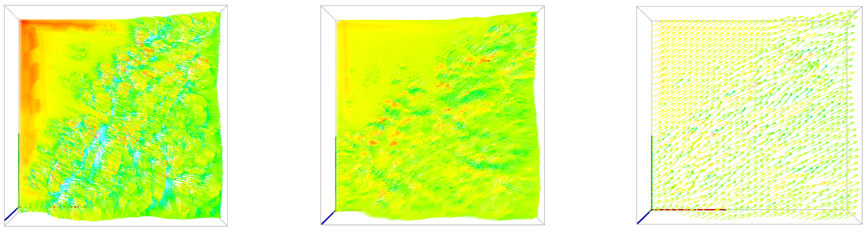

(a) Raw HR data (128 × 128)   **(b)** Normalized HR data (128 × 128)   **(c)** Generated LR data (32 × 32)

Figure 3.3: Channel-wise normalization and downsampling of velocity fields.

data, if necessary. The default normalization range is set to $0 \leq x \leq 1$. Minimum and maximum values are found in the HR training dataset, as the validation and test set should be considered unseen data. Figure 3.3 shows the resulting transformation from raw data to normalized, downsampled data. Finally, the datasets are loaded into PyTorch TensorDataset structures, which in turn are managed by PyTorch DataLoaders interfacing the ESRGAN architecture.

$$x_{norm} = (B_H - B_L) * \frac{x - \min(x)}{\max(x) - \min(x)} + B_L \tag{3.1}$$

where B_H is the upper bound, and B_L is the lower bound.

Normalization is required to shape the data as expected by the ESRGAN model. Note that the trained model will also generate samples with the same range. Therefore, the normalization factors are conserved to denormalize the reconstructed velocity fields using the corresponding inverse function.

3.4 Experiment I: Stability analysis of previous work

It was recently shown that a modified ESRGAN model could produce reasonable super-resolution velocity fields when using downscaled, coarse-resolution fields as input to the generator, and set the discriminator to distinguish between real and generated fine-resolution fields [47]. The base model architecture is presented in Chapter 2. In this experiment, the results of the work done by Tran *et al.* [47] are initially reproduced using recreated datasets from the same data generation source and identical hyperparameters for the model. The Python implementation of the proposed model (Chapter 2.7.3) was provided by the said author and consists of $\sim$ 3000 lines of Python code. Naturally, a considerable amount of time was spent reading and understanding the data flow and logic in their implementation. Their pre-trained model is used as a benchmark against the models trained from scratch in this experiment.

Originally, this experiment was meant to serve as an entry point for learning about GANs and how to train them, with the ultimate goal of ground-level airflow estimation using higher-altitude data. Through extensive testing it became clear that reproducing the results consistently using the hyperparameters and training methods described in [47] was infeasible. Through dialog with the main author of [47], the discrepancies between the work and the actual model were resolved. The resulting sets of hyperparameters are presented in Chapter 3.4.1. Thus, this experiment changed into an assessment of the method's ability to consistently achieve its advertised results. With the lack of similar literature on perceptual GAN-based super-resolution of velocity fields, subsequent experiments are formed through an iterative and inductive process based on empirical results. Multiple models using identical hyperparameters are trained in parallel to assess the model's ability to converge consistently. Ideally, one would only tune a single hyperparameter between each training session. Due to the significant amount of time and resources required to fully train this model, some model iterations consider changes in multiple hyperparameters.

3.4.1 Model hyperparameters and training hacks

The initial sets of hyperparameters related to the architecture and training algorithm of the model is shown in Table 3.2 and Table 3.3, respectively. When hyperparameters are modified between training sessions, updated values for the relevant hyperparameters are described. Note that the number of training iterations differ between the generator and discriminator in Table 3.3. Whenever the number of training iterations are mentioned, it is directed at the discriminator. The corresponding number of training iterations for the generator is calculated using the D/G training ratio. Moreover, the notation 1k = 1000 is used when specifying the number of training iterations.

The GAN training "hacks" described in Chapter 2.6 are present in the implementation. Instance noise is added to real and fake samples is shown in Equation 3.2, where $\epsilon \sim \mathcal{N}(0, 1)$, it is the current training iteration, and N is the total number of training iterations. Note that the noise-scaling term, $\sqrt{\frac{it}{\mathrm{N}}}$, increases the noise level towards the end of training rather than annealing it as intended by Sønderby *et al.,* Arjovsky & Bottou [46, 1]. It is not clear from the previous work whether this was intended or not. Through dialog with the author of the first iteration of this implementation, Vesterkjær [49], the increasing term was labelled as a bug in the code. Unfortunately, this bug remained undiscovered until after all experiments were concluded and there was no time left to fix it and re-train the models.

$$X_{\text{instance noise}}(X, it) = X + \epsilon * \sqrt{\frac{it}{\mathrm{N}}} \tag{3.2}$$

The label smoothing implementation has two modes: static and stochastic. With-

Table 3.2: Hyperparameters related to the model architecture.

Hyperparameter	Generator	Discriminator
Normalization	N/A	Batch normalization
Activation function	LeakyReLU, $\alpha = 0.2$	LeakyRelu, $\alpha = 0.2$
Base # of features	128	128
# of input channels	3	3
# of output channels	3	3
Weight initialization scaling	0.5	1.0
Feature kernel size	N/A	3
Local feature fusion kernel size	1	N/A
# of RRDBs	16	N/A
RDB growth rate	32	N/A
RDB residual scaling	0.2	N/A
RRDB residual scaling	0.2	N/A
HR kernel size	5	N/A

Table 3.3: Hyperparameters related to the model training algorithm.

Hyperparameter	Generator	Discriminator
Batch size	8	8
Initial learning rate	1e-4	1e-4
β_1 (First moment decay)	0.9	0.9
β_2 (Second moment decay)	0.999	0.999
Multi-step learning rate schedule	[10k, 20k, 30k, 40k]	[10k, 20k, 30k, 40k]
Multi-step learning rate decay	0.5	0.5
Loss function	Equation 2.41	Equation 2.40
D/G training ratio	N/A	2
Label smoothing	N/A	Equation 3.4
Instance noise	N/A	Equation 3.2
Training iterations	75k	150k
Validation period	Every 1000 iterations	Every 1000 iterations

out label smoothing, the real and fake labels are set to values 1.0 and 0.0. The static method sets the sample labels to 0.9 and 0.1 for real and fake samples, respectively. It was mentioned that Salimans *et al.,* Goodfellow ([42, 10]) both explicitly advised against smoothing fake labels, as it was shown to change the dynamics of the optimal discriminator (Equation 2.31). Nevertheless, label smoothing is still a heuristic "hack" that was created for a Goodfellow GAN model, and might therefore work differently for the more recent RaGAN implemented in this model. The stochastic method will be termed "noisy labels", which as described in Chapter 2.6 is not to be confused with the typical meaning of flipping the labels. This adds a small noise, $\gamma \sim \mathcal{N}(0, 0.05)$ to the smoothed labels. In summary, the label smoothing model can be described as:

$$l_{\text{static}} = \begin{cases} 0.9, & \text{real sample.} \\ 0.1, & \text{generated sample.} \end{cases} \tag{3.3}$$

$$l_{\text{stochastic}} = \text{clip}\left(l_{\text{static}} + \gamma\right), \qquad \gamma \sim \mathcal{N}(0, 0.05), \tag{3.4}$$

where clip() is a function that ensures that the resulting label stays within the range $0 \leq l_{\text{stochastic}} \leq 1$.

3.5 Experiment II: Assessing the validity of applying a perceptual SISR method to airflow data

While the ideal progression from Experiment I would continue towards finding a stable and consistent method, a limitation in computational resources lead the thesis to take another direction. A topic that is unaddressed in Tran *et al.* [47] is the significant difference in the data applied to an ESRGAN method. Although the data structures are equivalent, there is a considerable difference between a pixel in an RGB image and a point velocity vector in airflow data.

Although the data structure of RGB images and the relevant vector fields are identical, they originate from widely different systems. An RGB image (in this context) is essentially an information medium intended for visual interpretation by a human. There is no model for what an arbitrary image *should* look like. Natural images, however, contain local inter-pixel dependencies. When carefully selected, labelled images are collected into large datasets such as ImageNet, Flickr30k or CIFAR-10 and set in a supervised classification task, neural networks are able to learn meaningful features to differentiate between the different image labels. Chapter 2.7 explained how the use of pre-trained, deep perceptual features such as VGG19-54 in GANs-based image generation has improved state-of-the-art compared to traditional loss functions.

In contrast, the relevant vector fields in this context are generated from an equation-based model of a physical system. As such, their dynamics are defined by the relevant governing equations. Chapter 2.1 described how airflow is governed by a set of PDEs. As a result, the velocity fields are bound to global criteria such as conservation of mass, energy and momentum. However, such data cannot be labelled and shaped into a supervised learning problem. It is therefore not feasible to create a feature extractor specialized on airflow data in the same manner.

In order to train a super-resolving generative model specifically for airflow, one must ensure that the model indeed learns the governing equations of airflow. Similar models exist, however, known fluid flow super-resolution generative models don't utilize a perceptual loss. For instance, tempoGAN, by Chu & Thuerey [5], utilizes deep perceptual features from the discriminator itself. This is a particularly interesting solution, as it solves the mentioned issue of creating a feature extractor for airflow data. To the author's knowledge, perceptual loss is not used in any physics-based generative model, engineering applications outside computer vision tasks. However, the ESRGAN architecture and training algorithm has been used in DeepBedMap [28] to super-resolve terrain data, but they specifically avoided using perceptual loss. It is hard to argue how or why a perceptual loss would enable a generative model to learn governing equations. Nevertheless, the previous work did indeed show that a perceptual model is able to produce better results than the traditional bicubic interpolation method. Investigating how the perceptual loss behaves on airflow velocity data vs. RGB images will give some insight into how and why this method works, and to what extent.

3.5.1 Experiment IIa: Visualizing velocity fields as RGB images

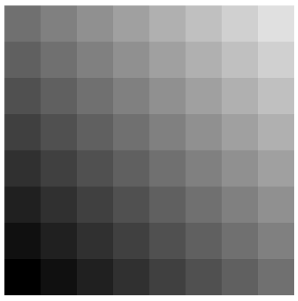
Image color values, $(r, g, b) \in \{0, 1, 2, ..., 2^b - 1\} \subset \mathbb{N}$, are discrete and bounded by the bit-depth, b, of the image. E.g., an 8-bit image has $2^8 = 256$ possible values for each color channel. In contrast, velocity data, $(u, v, w) \in \mathbb{R}$, are continuous and effectively unbounded. Consequently, it is trivial to find a map $(r, g, b) \to (u, v, w)$, but data-informed normalization is required to find a map $(u, v, w) \to (r, g, b)$.

In Experiment I, each velocity component is normalized based on their corresponding global maximum and minimum in the training dataset. As a result, the normalized training data is bounded within the range $0 \leq (u, v, w) \leq 1$, and can be represented as a b-bit RGB image by multiplying each entry with $2^b - 1$ and round them to the nearest integer. However, there is no guarantee that previously unseen validation and test data conform to the normalized range. This normalization method also suppresses dominant velocity components and significantly changes their direction in a Cartesian coordinate system. For instance, the vertical velocity component, w, is bound to have a smaller velocity range compared to the horizontal components, u and v. While information lost through normalization can be regained from denormalizing the data, this information is effectively unknown for the generative model.

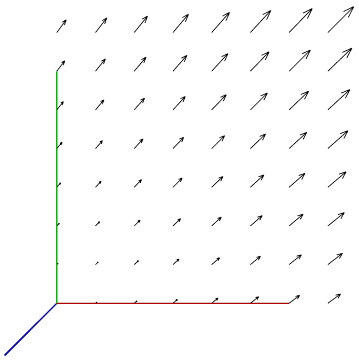
This experiment attempts to build towards an intuition for whether or not airflow velocity fields are eligible as input to a method that is originally constructed for perception-driven SISR. To do this, normalization is performed to visualize and translate data between RGB and vector field representations, using the same dataset and normalization method as in Experiment I. Two additional datasets are created from the same date range, but from higher altitudes in the SIMRA domain. These are thought to emphasize the perceptual difference between complex flow near ground level and smoother flow in higher altitudes.

Relating RGB images to vector fields

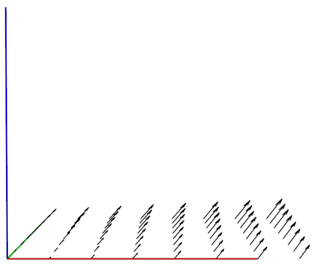
Figure 3.4 illustrates how a trivial gradient image can be represented as a vector field. The 8x8 RGB image in Figure 3.4a appears grayscale due to the symmetric



(a) 8x8 RGB gradient image.



(b) Vector field (Top view).

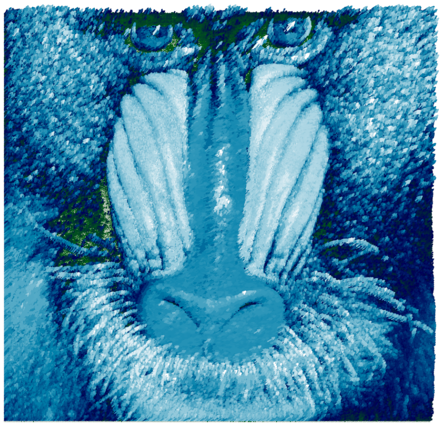


(c) Vector field (Side view).

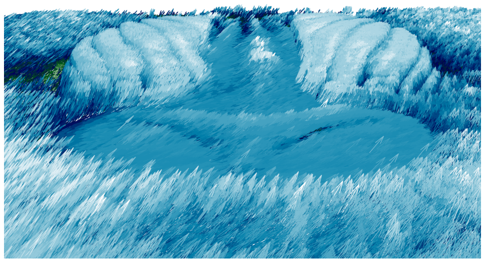
Figure 3.4: Direct translation of an 8-bit RGB image to a 3D Cartesian vector field. The color of coordinate axes in **(a)** and **(b)** correspond to the positive direction of their respective color in the reference image.

color values in the gradient from bottom left ($r = 0, g = 0, b = 0$) to top right ($r = 255, g = 255, b = 255$). Figure 3.5 shows how a more complex 480x500 RGB image can be directly translated to a vector field. A colormap ("ocean") was applied to the vector field, such that small vectors appear dark and large vectors appear bright. The color corresponds to vector magnitude, and is indifferent to direction. The detail view in Figure 3.5c shows how the dominant red and blue colors of the baboon correspond to their corresponding components in the vector field.

(a) RGB baboon (480x500).



(b) Vector field baboon (480x500).



(c) Detailed view (zoom in).

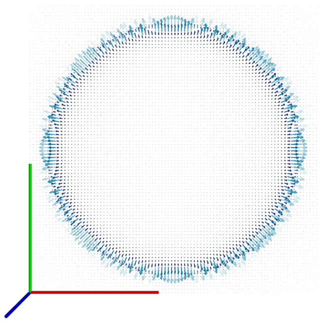
Figure 3.5: Converting a real image from RGB to a 3D vector field.

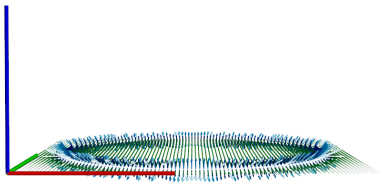
Relating vector fields to RGB images

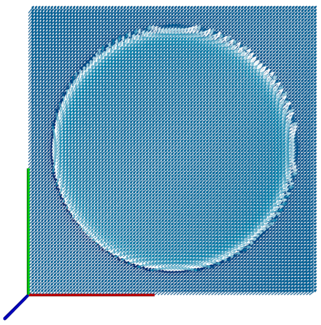
Figure 3.6 shows the inverse operation where a magnetic torus vector field is normalized and represented as an RGB image. This illustrates how gradual changes in vector directions relate to corresponding changes in color. Note that the original zero-vectors are represented by an average gray in Figure 3.6d due to the presence of negative vectors in Figure 3.6a-3.6b and the fact that all the velocity components have the same scale.

3.5.2 Experiment IIb: Investigating perceptual features for airflow data

The perceptual loss applied in ESRGAN is based on VGG19: a CNN intended for image classification, pre-trained on the ImageNet dataset. As such, the feature extractor, VGG19-54, is specifically trained to detect meaningful features in RGB images. When applied to airflow velocity data, will the feature extractor still find


(a) Vector field (Top view).


(b) Vector field (Side view).


(c) Normalized vector field (Top view).

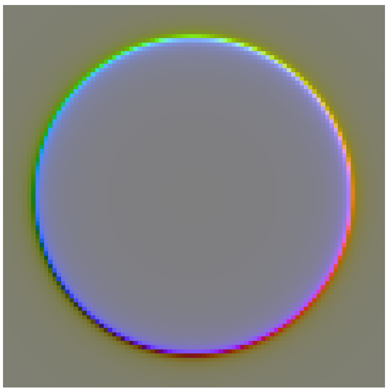

(d) Normalized vector field represented as an RGB image.

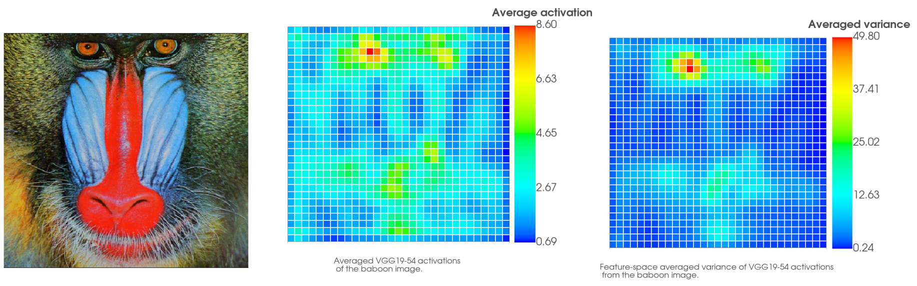
Figure 3.6: Converting a sliced 3D vector field to an RGB image.

meaningful features? What are "meaningful" features in this context?

For a 3x128x128 image input to the feature extractor, VGG19-54 (Figure 2.16) yields a 512x8x8 matrix of convoluted feature maps (Chapter 2.5), or 32768 features. Normally, this matrix is connected to several fully connected layers, each with 4096 nodes, acting as a classifier. Since the relevant VGG19 model was pre-trained on ImageNet, this classifier attempts to distinguish between 1000 different image classes. The pre-trained model has a documented Top-1 and Top-5 error rate of 27.62 and 9.12, respectively.[1] To achieve this accuracy, the classifier is highly reliant on receiving features that are different for each class.

Similarly, it is important for the generator in the ESRGAN model to receive meaningful differences between features of generated and true images in order to improve its super-resolution performance. However, Experiment I does not consider ESRGAN applied to image data. It will be shown that the model's perceptual loss, based on this feature extractor, does not converge in Experiment I. The lack of convergence is hypothesized to stem from an inability to use the information in the downsampled airflow data to mimic the feature extractor's learned perceptual features in the super-resolved velocity fields.

Since the raw output from VGG19-54 is high-dimensional in feature space, it is infeasible to compare in raw form. As described in Chapter 2.5, convolutional layers retain rough spatial information of their feature map. Therefore, more interpretable information can be obtained from calculating a feature-space mean and variance, leaving the spatial dimensions intact. This process smooths the information considerably, but is deemed sufficient for the purpose of this experiment. Figure 3.7 shows how the VGG19-54 activations are spatially correlated to its input image. It is clear that the feature extractor has found significant features around the eyes and mouth of the baboon, represented as large feature activations and high variance in those regions of the image.



(a) Input RGB image. (500x480)

(b) Feature-space average. (31x30)

(c) Feature-space variance. (31x30)

Figure 3.7: Feature activations of VGG19-54 are spatially correlated to the input data.

This experiment explores the variance of feature activations obtained from the VGG19-54 feature extractor for different datasets, as shown in Figure 3.7. Specifically, feature activations from airflow velocity training data is compared to RGB image datasets. The airflow data is processed the same way as in Experiment I. Moreover, the previous work (Tran *et al.* [47]) only considers data obtained from the bottom layer of the HARMONIE-SIMRA domain. Here, two additional airflow datasets are created from the same date range, using the same preprocessing

[1]Error rates are documented for classifying ImageNet samples cropped to a size 224x244.

methods, but from higher altitudes in the domain. High-altitude airflow is less affected by terrain, and is therefore assumed to be smoother compared to ground-level. These datasets are included to test whether the reduced complexity of a high-altitude velocity field impacts the information gained from the feature extractor. The image dataset is created from a random subset of the Flickr30k dataset [51], with a similar amount of samples cropped to 128x128 resolution. The original ESRGAN model was trained on Flickr2K and DIV2K, but Flickr30k was selected to match the number of image samples to the amount of airflow data. Table 3.4 lists details regarding each dataset. Note that the dataset "Bottom-layer" corresponds to the training dataset used in Experiment I.

Table 3.4: Datasets used for quantifying differences in feature activations for perceptual loss.

Dataset	Description	# samples	Processing
"Bottom-layer"	bottom of domain	15628	Channel-wise normalization.
"Middle-layer"	middle of domain	15639	Channel-wise normalization.
"Top-layer"	top of domain	15591	Channel-wise normalization.
"Flickr15.6k"	Flickr30k subset	15600	Crop to 128x128 resolution.

3.6 Experiment III: High-altitude airflow reconstruction with ESRGAN

It will be shown in Experiment II that the perceptual feature extractor network yields significantly less variation when presented a high-altitude dataset compared to the ground-level dataset used in Experiment I. With a relatively constant output from the feature extractor, it is hypothesized that this feedback drives the ESRGAN model towards generating samples with low perceptual variation rather than having the generator mimic perceptual features in its output. Hyperparameters for the executed model correspond to Session 4 in Experiment I, but with instance noise enabled. Deviations from the original hyperparameters are summarized in Table 3.5.

Table 3.5: Hyperparameter changes for Experiment III.

Changes hyperparameter(s)	Original value	New value
Label smoothing	Equation 3.4	Equation 3.3
Training iterations	150k	300k
Multi-step learning rate schedule	[10k, 20k, 30k, 40k]	[50k, 100k, 150k, 200k]

3.7 Performance evaluation

The proposed model evaluates the generated SR velocity fields using the PSNR metric. As mentioned in Chapter 2.7, this metric is originally intended for comparing RGB images, and suffers from qualitative variations due to imperceptible differences in psychovisual redundant information. While these variations may be considered noise when comparing RGB images, they represent important differences in vector magnitudes and directions in airflow data.

Furthermore, the PSNR metric is defined for discrete, bounded data. The airflow data used in this thesis is considered continuous and unbounded. Previous work solved this issue by adjusting the max fluctuation term, R^2, in Equation 2.33 to:

$$R^2 = 1.0 \qquad \text{For normalized training/validation data.} \tag{3.5}$$

$$R^2 = \max(I^{HR})^2 \qquad \text{For denormalized testing data.} \tag{3.6}$$

Additionally, a small epsilon, $\epsilon = 10^{-8}$, is added in the denominator to avoid the risk of an undefined logarithm (or division by zero). This results in a different R^2 term for every data sample. Moreover, vector components are equally likely to assume large negative values. For vector fields containing predominantly negative vectors, the corresponding maximum positive value may be small. In turn, a small R^2 term makes Equation 3.6 more sensitive to MSE. Although there is no clear choice for R^2, setting it to some constant value will ensure that the PSNR metric is calculated equally for all samples. To simulate a corresponding max fluctuation term, R^2 is set to the largest difference in normalization factors calculated for the training dataset. Thus, the resulting PSNR metric can be expressed as:

$$PSNR = 10 * \log_{10}(R^2) - 10 * \log_{10}(MSE + \epsilon) \tag{3.7}$$

where R^2 is the largest possible difference between normalization factors calculated for the training dataset.

Between training sessions, the PSNR validation performance is used to evaluate performance. Training and validation losses are used to identify signs of any failure modes or potential issues during training. Note that the training losses are calculated as a batch-size average, while validation losses are averaged across the entire validation dataset. After hyperparameter tuning is concluded, the models are run on the test set. Test set performance will be evaluated by calculating the PSNR mean and standard deviation of all denormalized, super-resolved samples for each trained model. These will be compared against the traditional bicubic interpolation method. In a real-world context, interpolation would not be performed on normalized data. Since interpolation is performed on "raw" low-resolution data, no processing is needed.

Additionally, the state-of-the-art LPIPS metric is applied for comparison. Zhang *et al.* [52] provide the pre-trained CNNs used in their work, from which the VGG architecture is used in this thesis. Although LPIPS is intended for evaluating differences between RGB images, its bounded nature should provide more interpretable results. Moreover, it has been shown that LPIPS corresponds significantly better to human judgement compared to PSNR. It will be shown that the ESRGAN model mostly attempts to minimize its own perceptual loss during training. However, the previous work did not justify whether minimizing a perceptual loss is synonymous to learning the governing equations of airflow for this model. Therefore, the LPIPS metric is compared against PSNR to find potential disagreements. Arguably, a perfect reconstruction will yield both a negligible perceptual distance (LPIPS ≈ 0.0 as well as MSE ≈ 0.0. If the model is focused on minimizing the perceptual distance and disregards psychovisual redundant information, then the metrics will disagree, e.g., LPIPS will be small, but MSE will be large. "Large" in this context refers to "larger than the corresponding MSE yielded by bicubic interpolation".

LPIPS assumes that its input data is normalized channel-wise to the range $-1 \leq (r, g, b) \leq 1$. Given that the generated airflow data is already within the range $0 \leq (u, v, w) \leq 1$, reshaping this data is a simple matter of transforming it again using Equation 3.1 with adjusted boundary values. Bicubic interpolated data is normalized with normalization factors calculated for the training data.

Chapter 2.7 mentioned that spatial feature differences can be found by not averaging over the spatial dimensions in Equation 2.35. By leaving the spatial dimensions intact, a spatial sample-mean can be calculated across the test set to find areas within the domain where the generative model struggles to reconstruct the airflow. In combination with an overlay of the terrain, this can provide a much more detailed difference in performance between the models and bicubic interpolation.

Ultimately, randomly sampled velocity fields from the best, an average and the worst performing models are illustrated and compared against the corresponding high-resolution field and the bicubic interpolated result. A random-number generator selects different samples for each presented model to avoid "cherry picking" the results. Each velocity component will be investigated individually using filled contour plots with discrete color maps. Thus, detailed structural information can be interpreted from each velocity component of the different methods.

Chapter 4

Results and discussion

This chapter presents a historical development of the investigation of the proposed ESRGAN applied to airflow data. First, details regarding the data collection and preprocessing are presented. Subsequently, the results from each experiment introduced in Chapter 3 are presented. Summarized discussions are included in Experiments I and II to motivate the progression to the next investigation. Ultimately, the performance evaluation concludes the results obtained from trained models in Experiments I and III.

4.1 Data preparation

In order to replicate the results, the data underwent the equivalent preprocessing as in [47]. Thus, data generated by the HARMONIE-SIMRA system in the date range 04.08.2017 to 30.10.2019 was downloaded and preprocessed as described in Chapter 3.3.[1] This corresponds to 818 days consisting of 19632 velocity field samples of hourly averaged data. After rejecting invalid values, 19535 samples remain (0.5% rejected). Table 4.1 presents the resulting sizes of each dataset. Normalization factors calculated for each velocity component of each dataset are presented in Table 4.2.

Table 4.1: Training, validation and test set sizes.

Dataset	Training data	Validation data	Test data	Usage
Bottom-layer	15628 samples	1953 samples	1954 samples	Experiment I
Middle-layer	15639 samples	1954 samples	1955 samples	Experiment II
Top-layer	15628 samples	1953 samples	1954 samples	Experiment II and III

[1][47] describes a different initial date. Through dialog with the author of [39], it was confirmed that the earliest available date is 04.08.2017.

Table 4.2: Normalization factors calculated for each channel in all airflow datasets

Velocity component	Dataset		
	Bottom-layer	Middle-layer	Top-layer
	[max, min]	[max, min]	[max, min]
u	[18.99, -12.54]	[40.38, -27.85]	[36.09, -23.94]
v	[13.54, -13.64]	[31.24, -33.77]	[31.36, -33.64]
w	[5.59, -2.68]	[9.15, -6.80]	[4.43, -3.19]

4.2 Experiment I: Stability analysis of previous work

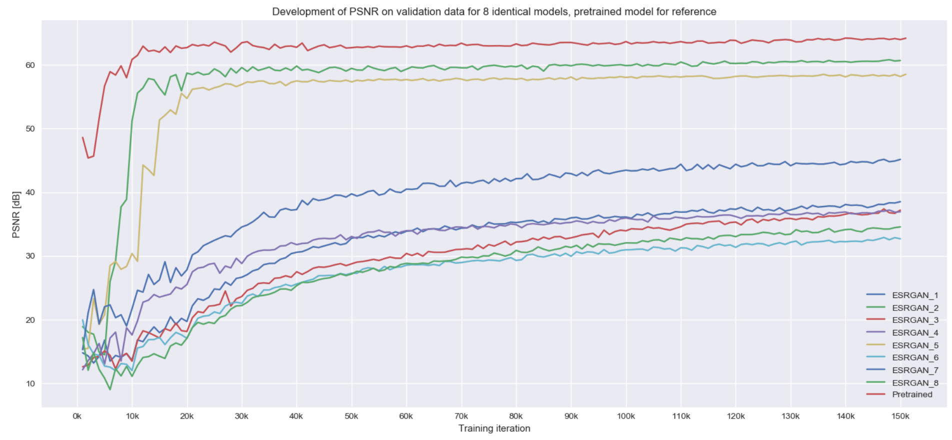
This experiment is divided by subsequent attempts to stabilize the proposed model through an iterative-inductive process. One or more hyperparameters of the model are tuned between each attempt, and several models are trained in parallel to assess the impact on training stability and performance variations. Hereafter, the individual attempts are termed *sessions*.

4.2.1 Session 1: Training stability of the previously proposed model

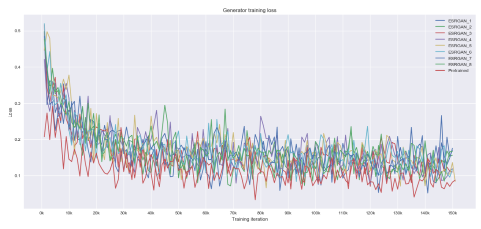
First, an important note regarding the performance metric used is addressed. As described in Chapter 3, PSNR is not an ideal metric. Due to the unconventional use of comparing velocity fields as RGB images, this metric is only used to compare training development for models. Moreover, the scale of the data differs when considered during training vs. testing due to the corresponding normalization and denormalization. Thus, the R^2 (max fluctuation) term is changed depending on the relevant scale. Therefore, the amplitude of the presented PSNRs are used only for comparison between experiments, and should not be compared to conventional values when applied to RGB images.

Chapter 3.4 mentioned how the original goal of this experiment was changed due to training instabilities. A considerable amount of time was spent on dialogue with the author of [47] to modify the Python implementation to correspond to their proposed model due to typos and omitted details. In total, eight identical models were trained to get an initial understanding of the model stability. The pre-trained model was initialized at its end state and trained for an additional 150k iterations to assert whether the current implementation is stable. Note that this results in a total of 300k training iterations for the pre-trained model, so it is expected to perform better. Unfortunately, the pre-trained model's training losses for its first 150k iterations were unobtainable.

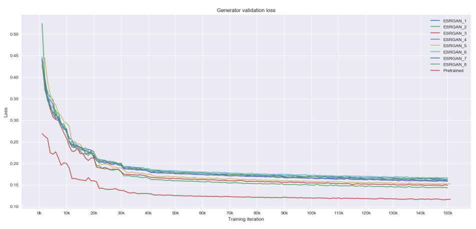
Hyperparameters for these models are shown in Tables 3.2 and 3.3. The resulting training and validation metrics are presented in Figure 4.1. It is recommended that the reader zooms in on the plots to see the axes and scales properly. The models seem to produce varying results consistently. Out of the eight identically



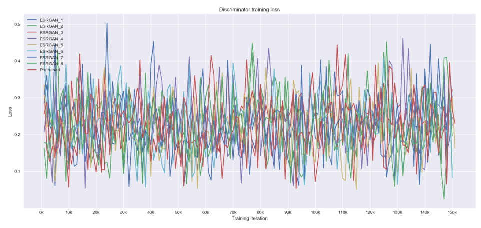
(a) Comparing PSNR development on validation data for 8 identical models against a pre-trained model.



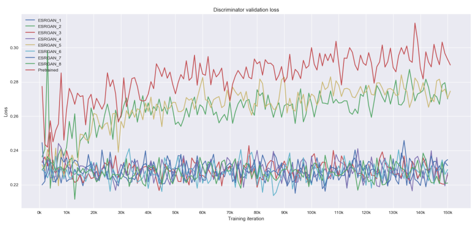
(b) Training loss for generators.



(c) Validation loss for generators.



(d) Training loss for discriminators.



(e) Validation loss for discriminators.

Figure 4.1: Training session 1: Training instability across eight identical ESRGAN models.

trained models, two models in Figure 4.1a seemed to produce significantly better results compared to the others. Figure 4.1e shows that the best performing models have a steeper increase in discriminator validation loss early on, while this loss is oscillating around a constant value for the worse-performing models. Both training losses in Figures 4.1b and 4.1d are heavily affected by noise. Recall that the models use both stochastic label smoothing and instance noise, which both contribute towards confusing the discriminator. While noisy losses are expected, they should converge (to a certain extent) towards a value at the end of the training. The generator losses have a clear exponential trend, but like the discriminator losses, they oscillate without any sign of convergence.

4.2.2 Session 2: Continued training stability analysis with static label smoothing

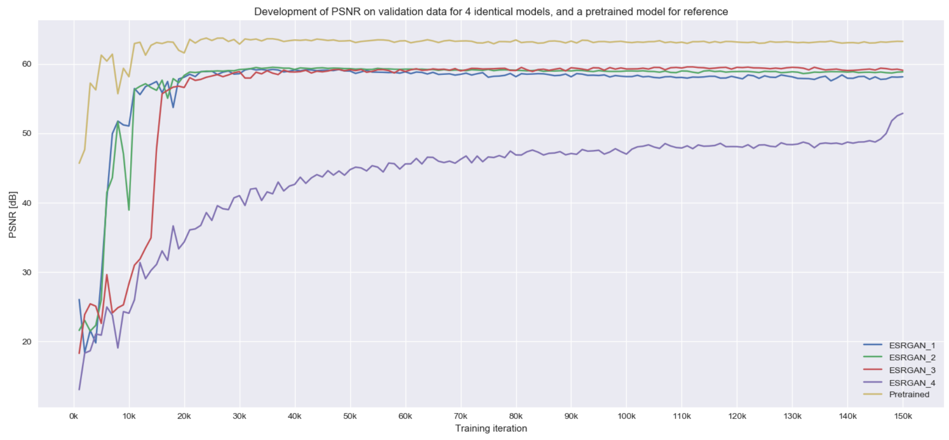
Despite earlier efforts, additional discussions with the author of the model were required to determine the correct proposed model. A change was made to the hyperparameters of the model. Specifically, the label smoothing was changed from stochastic to static, such that real and fake labels are set to flat values 0.9 and 0.1, respectively.

Table 4.3: Hyperparameter change for training session 2.

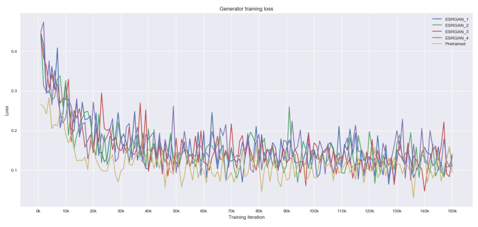
Changed hyperparameter(s)	Original value	New value
Label smoothing	Equation 3.4	Equation 3.3

Following the change in hyperparameters (Table 4.3), four models were trained from scratch, along with another instance of the pre-trained model. Figure 4.2 presents their PSNR performance and losses. With static label smoothing, most models achieve a more consistent PSNR performance, although one of them still failed to converge to a similar state. Note that model "ESRGAN_2" in Figure 4.2a suffers from a barely noticeable convergence failure, as explained in Chapter 2.6. The discriminator training losses (Figure 4.2d) are significantly more stable throughout the training compared to the models in Session 1. In fact, they appear too stable. Recall that the RaGAN losses are symmetrical and that the discriminator loss is intended to increase as the generated samples become less distinguishable from real samples. Since there is no rise in discriminator loss, it is likely that it successfully distinguishes between real and fake samples throughout the training. Moreover, the models seem to share a trend of increasing discriminator validation loss (Figure 4.2e) until 50k iterations. After this point, the validation losses start to diverge from each other. The discriminators may have overfitted the generator training samples, and fail to recognize generated validation samples. Recall that Chapter 2.6 describes such behavior as an initial phase of mode collapse. However, the generator's heavy reliance on content-related losses prevents the model from collapsing a range of low-resolution velocity fields to a single super-resolved output. Hypothetically, this would be recognizable in the PSNR validation performance plot.

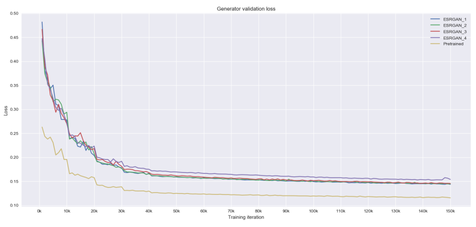
In contrast, the worst model, "ESRGAN_4", experienced poor PSNR performance and almost constant discriminator losses until the end of the training period. Despite that the learning rate reached its minimum value nearly 100k iterations earlier, the model managed to escape a long-lasting local minima. From Figure 4.2a it is unclear whether "ESRGAN_4"'s performance development further would continue increasing or flatten out. It is interesting to note how this escape is very noticeable



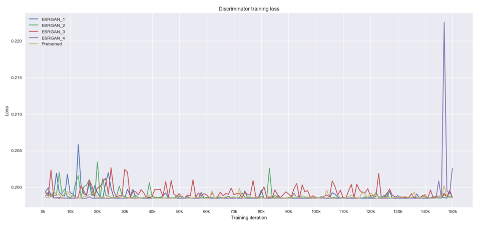
(a) Comparing PSNR development on validation data for 4 identical models against a pre-trained model, noisy labels disabled.



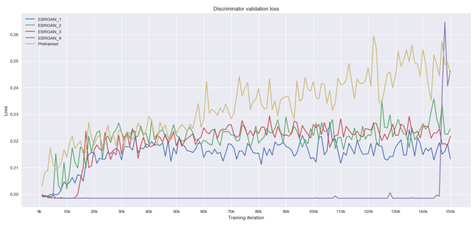
(b) Training loss for generators.



(c) Validation loss for generators.



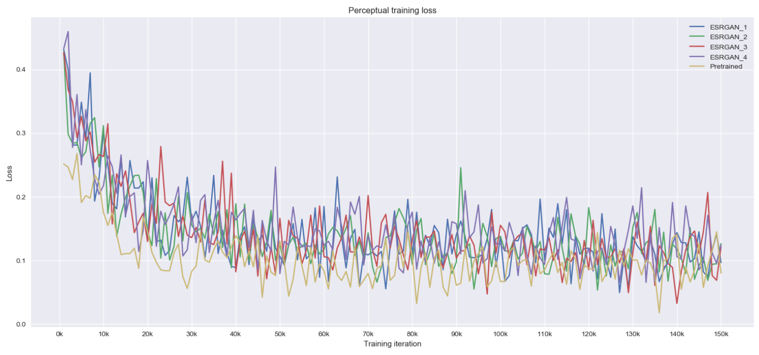
(d) Training loss for discriminators.



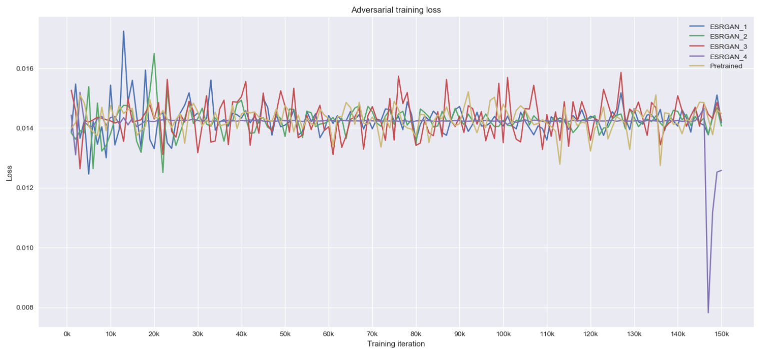
(e) Validation loss for discriminators.

Figure 4.2: Training session 2: Improved training stability and performance.

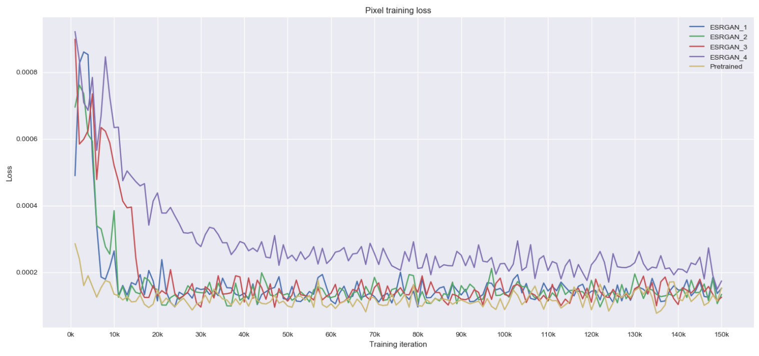
in the discriminator losses, but barely visible in the corresponding generator losses. Figures in 4.3 show the individual weighted components of the generators' losses, as described in Equation 2.41. The perceptual losses in Figure 4.3a decrease initially but oscillate with a relatively constant amplitude of around 0.1 (peak-to-peak) throughout the rest of training. Recall that the adversarial and pixel losses are weighed with factors $\lambda = 0.005$ and $\eta = 0.01$. While the discriminators arguably reach their minimum possible loss value of 0.199 in both training and validation, the corresponding unweighted adversarial losses oscillate around $0.014 * 200 = 2.8$



(a) Perceptual loss component.



(b) Adversarial loss component.

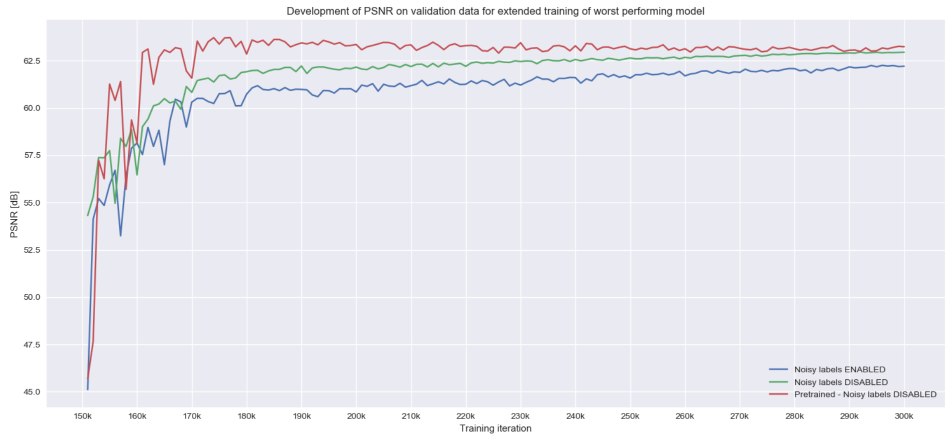


(c) Pixel loss component.

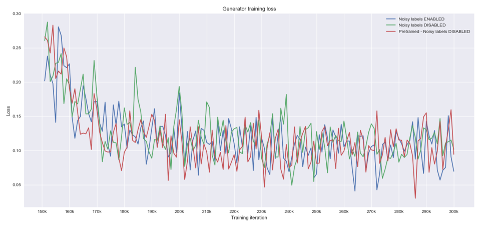
Figure 4.3: Session 2: Decomposing the generators' training losses into their weighted components.

for the generators. The large weighing on perceptual loss, in comparison, overshadows the discriminators' feedback. When "ESRGAN_4" breaks out of its local minima, the discriminator training loss is seen to return to a low value quickly.

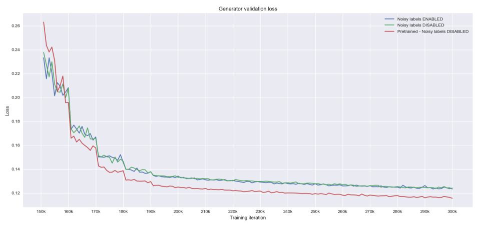
In contrast, the discriminator validation loss remains considerably high. This shows how quickly the discriminator can overfit a sudden change in the generator. It is unfortunate that this event happened so close to the end of the training, as it would be interesting to follow this model's progression further.



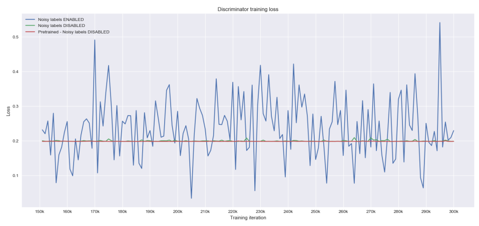
(a) PSNR development for extended training on previously worst performing model, compared to pre-trained model.



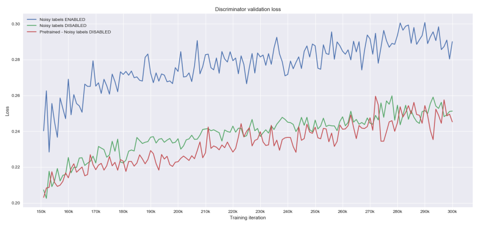
(b) Training loss for generators.



(c) Validation loss for generators.



(d) Training loss for discriminators.



(e) Validation loss for discriminators.

Figure 4.4: Training session 2: Extended training for the worst performing model.

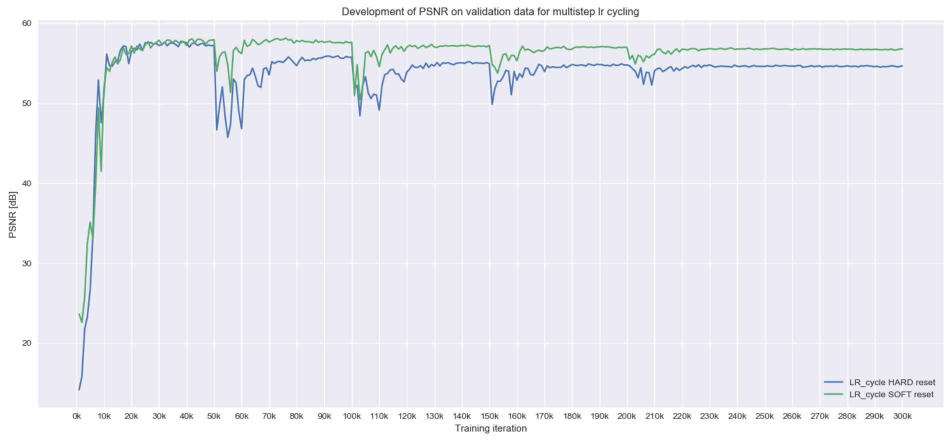
Extended training of the worst performing model

The model instance labeled "ESRGAN_4" in the previous training session managed to escape its poor local minima at the very end of the training period. Training continuation of the worst performing model, "ESRGAN_4", is attempted for another 150k iterations with two parallel instances. Stochastic label smoothing is reactivated for one model, while the other had no change in hyperparameters. Note that this process is equivalent to resetting the learning rate schedule during training. Considering their total training period, the multi-step learning rate schedule was effectively reset after 150k iterations. Figure 4.4 presents the results from training these models. The extended pre-trained model from the previous training session is included in these plots. Note how the model with noisy labels enabled experiences significantly larger variations in discriminator training loss and higher discriminator validation loss while its PSNR performance is worse than the other models. However, the corresponding generator losses assume a shape similar to the other models. Performance-wise, these models are the closest to the pre-trained benchmark yet. Since the amount of training time is one of the most crucial hyperparameters for any neural network model, an increase in training iterations could be beneficial. Furthermore, cycling of the learning rate schedule might have the potential to help poorly converged models out of their local minima. However, the extended training has shown no improvements for the convergence of perceptual loss.

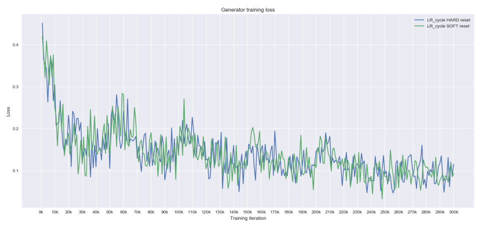
4.2.3 Session 3: Multi-step learning rate cycling

Performance improvements from extended training in Session 2 motivated an exploration of multi-step learning rate cycling in combination with training for twice the amount of iterations. Due to the increase in training time, the number of models trained each session is reduced. In PyTorch, learning rate cycling is implemented for switching between two static learning rates, but not for multi-step learning rate schedules. Therefore, the cyclic multi-step learning rate scheduler had to be implemented manually. To avoid perturbing the model near the end of training, cycling is disabled for the final 20% of iterations. The use of the Adam optimizer introduces some design issues. Adam keeps a moving average of the previous gradients used for updating the parameters in its internal state. In addition to the frequency for which the schedule should be cycled, a hyperparameter for keeping or discarding the "memory" of the Adam optimizer was created and respectively termed "hard" and "soft" resetting. New hyperparameters are defined in Table 4.4. Both hard and soft resetting modes are explored with a cycling period of 50k iterations. With a total of 300k training iterations, learning rate cycling occurs at [50k, 100k, 150k, 200k], and is disabled after the model has trained for 240k iterations. This is summarized in Table 4.5 and Figure 4.6.

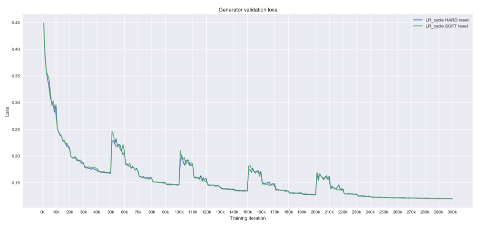
Figure 4.5 shows the training of the two models. Both models reach a similar PSNR performance before the first learning rate cycle, a good basis for comparing effects of the hard vs. soft resetting methods. It is clear that the hard reset method can be detrimental to an initially well-performing model. The soft reset share



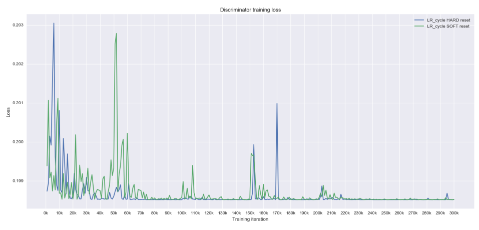
(a) PSNR development for multi-step learning rate cycling each 50k$^{\text{th}}$ iteration.



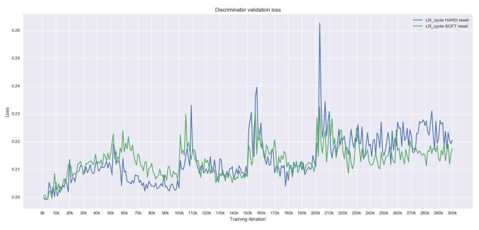
(b) Training loss for generators.



(c) Validation loss for generators.



(d) Training loss for discriminators.



(e) Validation loss for discriminators.

Figure 4.5: Training session 3: Multi-step learning rate schedule cycling each 50k$^{\text{th}}$ iteration.

a similar trend, although less significant. The discriminator losses share similar overfitting characteristics as in Session 2, where the training loss quickly returns to its minimum after every cycle. Validation losses, however, experience increasing spikes after each cycle. Nevertheless, neither model exceeds the best models in Session 2. While these results suggest the effect of learning rate cycling, it is only shown for two initially well-performing models. Nothing can be concluded on its effects on poorly performing models other than the empirical results in Figure 4.4a, which simulated a single cycle at iteration 150k with hard resetting. The impact

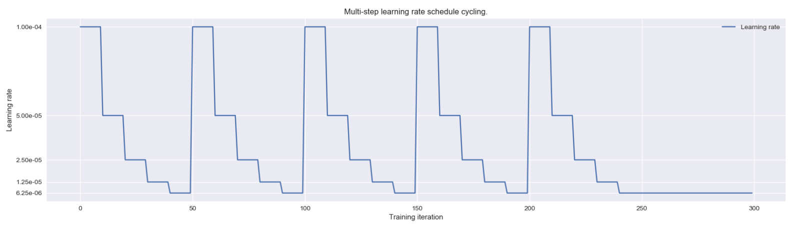


Figure 4.6: Session 3: Learning rate development during training.

Table 4.4: Hyperparameters introduced by implementing multi-step learning rate cycling.

Name	Type	Description
lr_cycle_period	Positive integer	Number of training iterations to execute before cycling the learning rate schedule.
lr_cycle_hard_reset	Boolean	TRUE/FALSE: "Hard"/"Soft", discard/keep Adam optimizer's internal state when cycling.

of soft-reset cycling on a poorly performing model is still unknown.

Lower learning rate cycle frequency and removing instance noise

This training session aims to further explore multi-step learning rate cycling with hard resetting to more closely resemble the extended training in Session 2. The previous models considered a higher cycling frequency than initially investigated, which was shown to have detrimental effects on well-performing models. Moreover, the presented plots are difficult to interpret even without learning rate cycling. Lowering the frequency may increase their readability by allowing models to stabilize more between cycles. An updated learning rate development is presented in Figure 4.7 Additionally, due to the empirical performance improvements from disabling noisy labels in Session 2, this training session also investigates the impact of disabling instance noise as described in Chapter 2.6. Instance noise acts as a source of confusion for the discriminator in early training, and is annealed towards

Table 4.5: Hyperparameter change for training Session 3.

Changed hyperparameter(s)	Original value	New value
Label smoothing	Equation 3.4	Equation 3.3
Training iterations	150k	300k
Learning rate cycle period	N/A (new)	50k (disabled after 200k)
Learning rate cycle reset mode	N/A (new)	Both are used

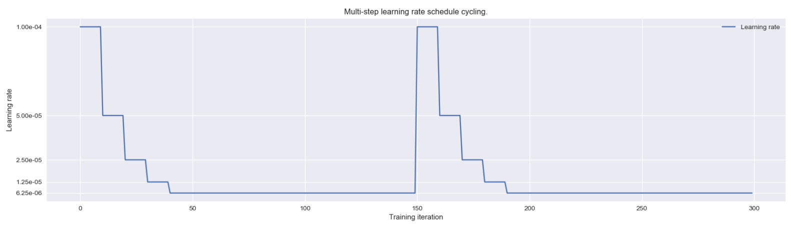


Figure 4.7: Session 3: Updated learning rate development during training.

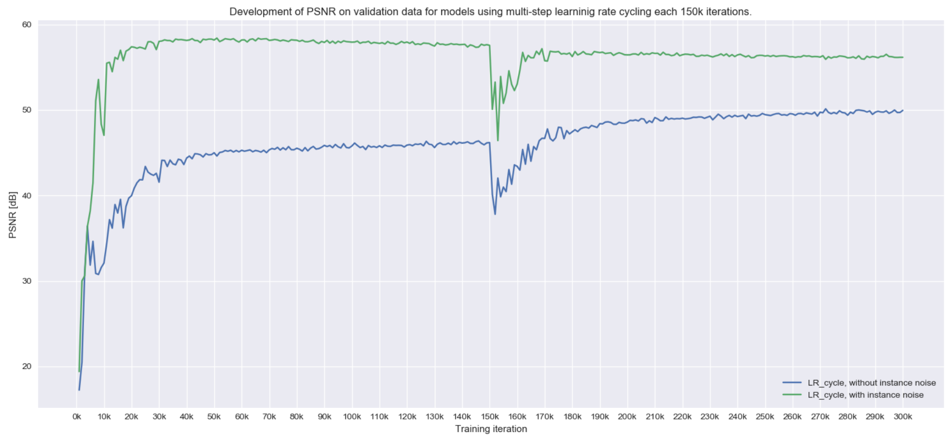
the end of training.

Figure 4.8 shows the results of this session. Considering the model with instance noise disabled, there is a slight improvement in PSNR performance following the cycling event, although not as significant as what was observed with the equivalent model in Figure 4.4a. Based on the poor PSNR performance in combination with the near-constant discriminator losses, it appears that the model is stuck in a bad local minima. In fact, this model seems to have encountered an issue similar to the worst performing model in Figure 4.2; the discriminator's training and validation losses appear constant. However, as the learning rate cycles at 150k iterations, the discriminator losses are essentially unaffected. A small perturbation can be observed in Figure 4.8e, which quickly returns the constant value. It is not clear whether this issue was induced by disabling instance noise or if it corresponds to the behaviour of model "ESRGAN_4" in Session 2.

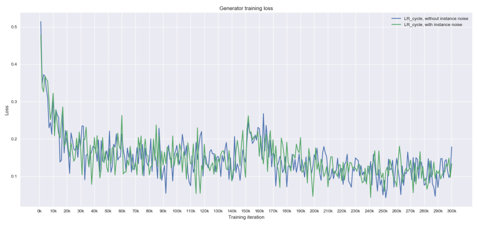
The model with instance noise enabled behaves similar to the corresponding previous model. However, it could be worth noting that the PSNR performance has already started deteriorating from convergence failure before the learning rate cycle event. Therefore, the continued performance deterioration after cycling the learning rate could also be attributed to this failure mode. Considering the empirical results gathered so far, multi-step learning rate schedule cycling has been mildly beneficial for poorly performing models, and similarly inhibiting for well-performing models. For the sake of consistent and predictable training, this is a step in the right direction, but this method might not be the optimal way to achieve consistent results.

The development of instance noise was not considered when implementing multi-step learning rate cycling. As the magnitude of the added noise is decreasing towards the end of training, this should also have been reset to conform with the extended training in Session 2.[2] Although implementing this could better approximate the retrained model, it would further complicate an already complex model. In the interest of avoiding unnecessary complexity, the next experiment attempts a different approach.

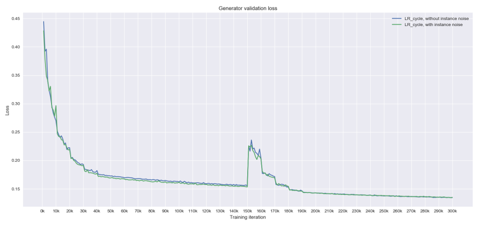
[2]The bug in Equation 3.2 was still undiscovered at this point, but has no significant impact for the conclusion of this thesis.



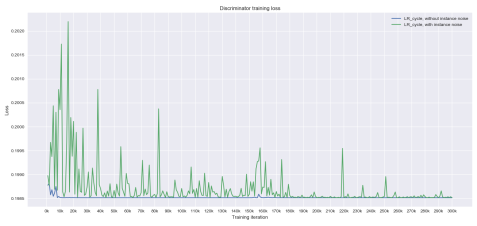
(a) PSNR development for multi-step learning rate cycling each 150k$^{\text{th}}$ iteration.



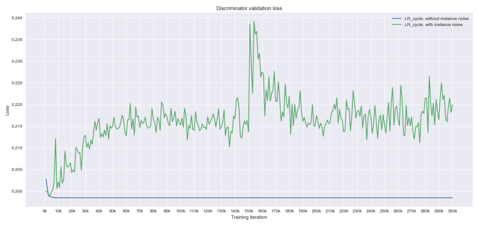
(b) Training loss for generators.



(c) Validation loss for generators.



(d) Training loss for discriminators.



(e) Validation loss for discriminators.

Figure 4.8: Training session 3: Multi-step learning rate cycling each 150k$^{\text{th}}$ iteration.

4.2.4 Session 4: Extended training period and disabled instance noise

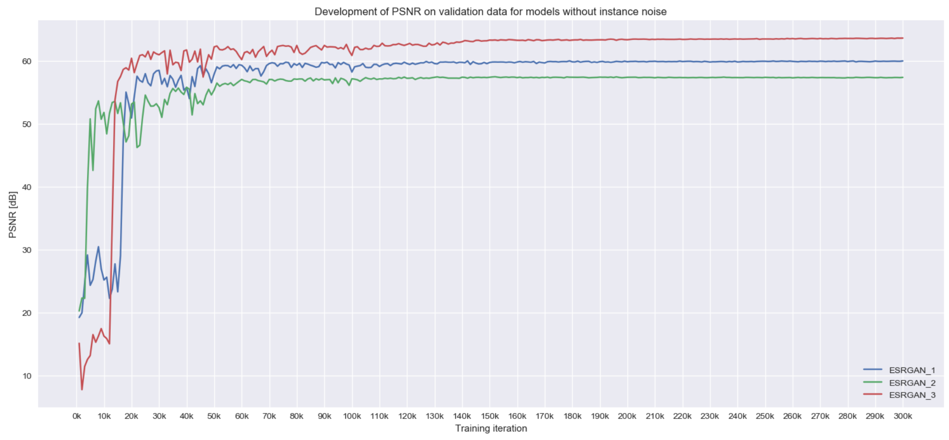
It is apparent from Figure 4.1a that the worse performing models all have a positive slope for PSNR at the end of training. Figure 4.2a illustrates that there is a chance for a poor model to improve at the end of training. Although retraining this model yielded promising results, Session 3 concluded that a multi-step learning rate schedule cycling is not effective enough considering the added complexity to the training algorithm. A common factor in all sessions is the significant variation of PSNR performance in the first 50k iterations of the training. Within the first 50k iterations, the multi-step learning rate schedule has already reached the minimum learning rate for the remainder of the training period. This session takes a step back, and explores what happens when the models are allowed to train for twice as many iterations, and the (original) multi-step learning rate schedule is extended accordingly. Moreover, to address the change in discriminator loss when noisy labels was disabled in Session 2, this experiment further reduces the training algorithm's stochasticity by disabling instance noise as well. This was briefly explored in Session 3, although no conclusions could be drawn. This session aims to further assess the impact of instance noise for this model. Table 4.6 shows how each hyperparameter is changed for this session. Three models were trained this time; the fourth model was prevented from executing due to a temporary exhaustion of the institute's computational quota on IDUN.

Table 4.6: Hyperparameter changes for training session 4.

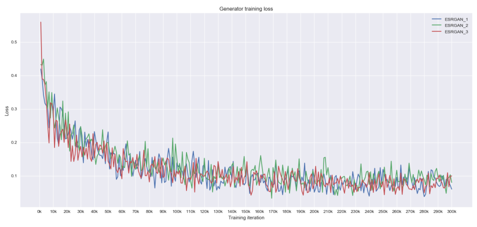
Changed hyperparameter(s)	Original value	New value
Label smoothing	Equation 3.4	Equation 3.3
Training iterations	150k	300k
Multi-step learning rate schedule	[10k, 20k, 30k, 40k]	[50k, 100k, 150k, 200k]
Instance noise	Equation 3.2	None

Figure 4.9 shows the results from this training session. Although the PSNR performances lie in the higher range compared to previous results, the discriminator validation losses behave differently. Figures 4.9d and 4.9e show how the discriminator losses converge towards the same value, and Figure 4.9a shows that the PSNR validation performance for all three models converge to values between 58 and 64 [dB]. While these PSNR performances are considerably more stable than in previous training sessions, the models converge to different values. Due to the small sample size (3 models), there is no guarantee that a fourth model would converge within the same range.

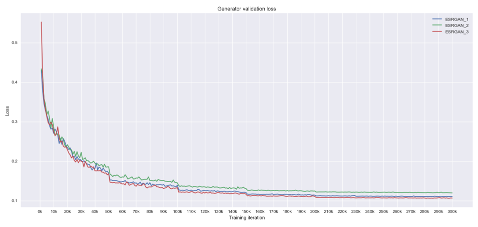
It is curious how the PSNR performances are unchanging when the generators' training losses are oscillating considerably even towards the end of training. While the learning rate schedule does decrease the models' learning rate throughout the training period, the PSNR performance remains stable after the second decrease in learning rate (100k iterations). Figure 4.10 shows the models' generator training losses decomposed into their terms as shown in Equation 2.41. Recall that the adversarial and pixel losses are weighted with a factor of $\lambda = 0.005$ and $\eta = 0.01$, respectively and are illustrated with this weighting. The unweighted pixel loss converges to a relatively low value, $L_1^{Pixel} = 0.0002 * 100 = 0.02$. In contrast, the weighted adversarial loss converges around a value 0.014; the corresponding unweighted value would be $\frac{1}{\lambda} L_G^{RaGAN} = 200 * 0.014 = 2.8$. Compared to the discriminator adversarial losses (Figure 4.9d), this is very high considering that RaGAN's loss functions are symmetrical. Hyperparameter changes performed so far have mostly targeted the discriminator, as most issues found in the literature



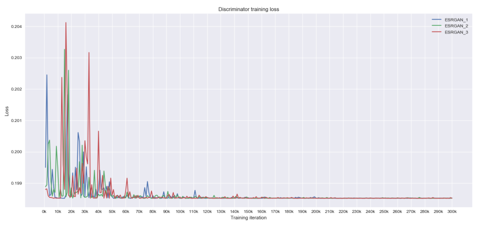
(a) PSNR development for training models with instance noise disabled.



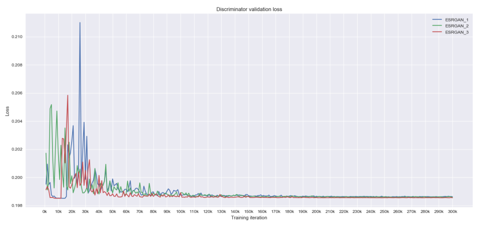
(b) Training loss for generators.



(c) Validation loss for generators.



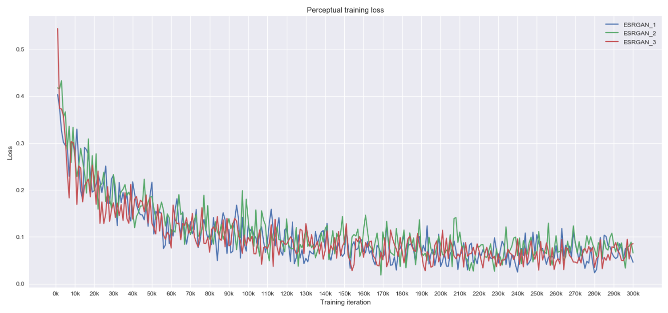
(d) Training loss for discriminators.



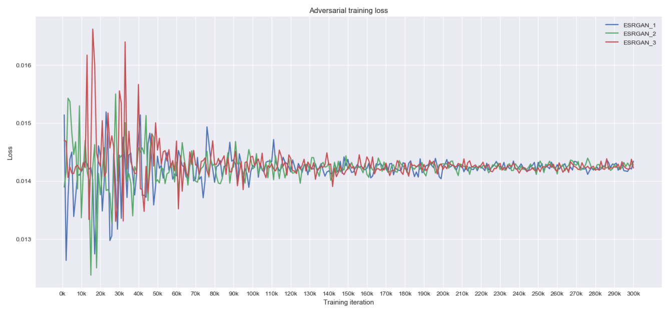
(e) Validation loss for discriminators.

Figure 4.9: Training session 4: Extended training to 300k iterations and disabled instance noise.

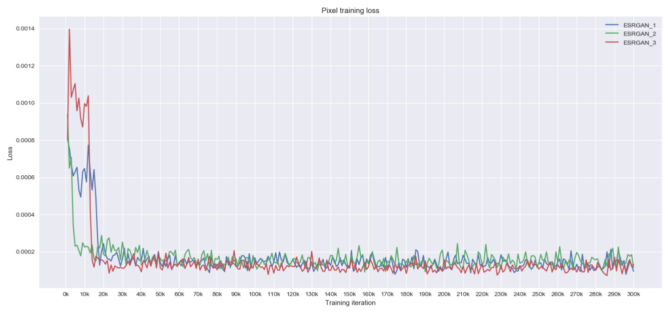
search related to GAN training is attributed to the discriminator [10, 42]. None of the training sessions in Experiment I has targeted the generator's training alone. Considering the heavily weighted perceptual loss, which greatly affects the generator of this model, it is worth looking more into detail of how the feature extractor impacts training.



(a) Perceptual loss component.



(b) Adversarial loss component.



(c) Pixel loss component.

Figure 4.10: Training session 4: Decomposing the generator training losses into their weighted components.

4.2.5 Summary and discussion of Experiment I.

This experiment has provided an overview of the training stability of the proposed model for super-resolving coarse-resolution airflow velocity fields using the perceptual-based SISR model, ESRGAN. Tuning different hyperparameters for the model has given insight into inhibiting factors such as noisy perceptual loss and discriminator overfitting. The conducted training sessions have not reached a conclusion for how to best stabilize the model.

From stochastic to static label smoothing

Session 2 showed an overall increase in PSNR validation performance and more consistent loss progressions after changing from stochastic to static label smoothing. The label smoothing method is still in conflict with arguments from Goodfellow, Salimans *et al.* who argued that smoothing fake labels will change the dynamics of the optimal discriminator in a Goodfellow GAN framework. This model implements RaGAN, which by design behaves quite differently from Goodfellow GAN. In comparison, RaGAN may be less sensitive to smoothing of fake labels. However, the larger weighting and values of the generators' perceptual loss component has shown to overshadow the discriminator feedback.

Multi-step learning rate schedule cycling

While the extended training of the worst performing model, "ESRGAN_4", in Session 2 performed close to the extended pre-trained model, the corresponding models in Session 3 using a novel multi-step learning rate schedule cycling method did not achieve similar results. Similar to this "ESRGAN_4", one of the models in Session 3 experienced the same issue with constant discriminator losses. Since this model also considered disabling instance noise, the results from Session 3 were inconclusive. By considering models in Session 4, it was found that disabling instance noise does not cause this issue. Therefore, it seems more likely that cycling the learning rate schedule is insufficient to "bump" poor models out of their local minima.

Longer training and disabling instance noise

Due to large variations in PSNR validation performance within the first 50k training iterations, Session 4 considered extending the number of training iterations and widen the multi-step learning rate schedule accordingly. Furthermore, instance noise was disabled for all models in order to obtain a greater understanding of its effects. Compared to the previous sessions, these models achieve a more stable PSNR validation performance towards the end of training, albeit only with a small margin. Furthermore, the models converged to different PSNR values. Interestingly, the discriminator validation losses appear significantly different from the previous sessions.

The impact of a faulty instance noise implementation

The noise-scaling bug in Equation 3.2 was undiscovered until after all experiments were concluded (including Experiment II and III) and there was no time left to fix it and redo the experiments. In the hindsight, it is obvious that the increasing amount of instance noise throughout the training impacts the discriminator validation losses in many of the preceding sessions of Experiment I. As described in Chapter 2.6, an annealing instance noise is intended to make the task of divergence minimization between the true and approximated distributions well-defined. The implementation error effectively reversed the scaling such that the samples were added more and more noise towards the end of training. This would likely be more problematic if the model's loss functions were purely adversarial. Due to the addition of content- and perceptual loss, the generators were not significantly affected. In fact, even though the discriminators in Session 2-4 managed to minimize their losses, the generators were virtually unaffected.

On the non-convergence of perceptual loss

As explained in Chapter 3, the applied ESRGAN model is heavily reliant on a perceptual loss. Considering that the generator losses share a similar development throughout all training sessions, the different hyperparameter changes has not affected the generators loss development significantly. There is one exception; in Session 1, the initial stochastic label smoothing method elicited large discriminator training losses compared to the subsequent training sessions. However, the discriminator losses showed no sign of converging, and there was large variations in the models' PSNR validation performances. With static label smoothing, the discriminator training losses consistently converged to a minimum value of 0.2. Recall that Chapter 2.6 described the expected discriminator loss development in a well-performing model to be low initially, with an increasing trend before convergence. This behaviour has not been observed for any of the trained models so far. The weighting of the generator's loss components (Equation 2.41) in combination with the large, noisy perceptual loss values is thought to overshadow the discriminators' feedback. In Session 2 and 4, the generators' losses were decomposed into their weighted components, and a discussion was started regarding the convergence of perceptual loss. The previous work did not justify whether or not it is feasible to apply such a perceptually driven model to super-resolve airflow data, nor how it can solve the governing equations of airflow (Chapter 2.1). Since the feature extractor is pre-trained for classification of ImageNet data, its activations relate to high-level features it is trained to look for in RGB image data. Therefore, the next step of this thesis starts an investigation on the use of a perceptual loss applied for super-resolution of airflow data, specifically.

4.3 Experiment II: Assessing the validity of applying a perceptual SISR method to airflow data.

The applied ESRGAN framework was developed for SISR on RGB image data, inheriting assumptions such as negligible psychovisual information loss and infinite plausible solutions for a reconstructed HR image. Whereas small changes in color may be negligible in SISR, the corresponding error in velocity fields can impact vector directions. Before investigating the feature extractor used for perceptual loss, an intuition for the differences and similarities between RGB and airflow velocity data is established by representing the airflow as RGB data.

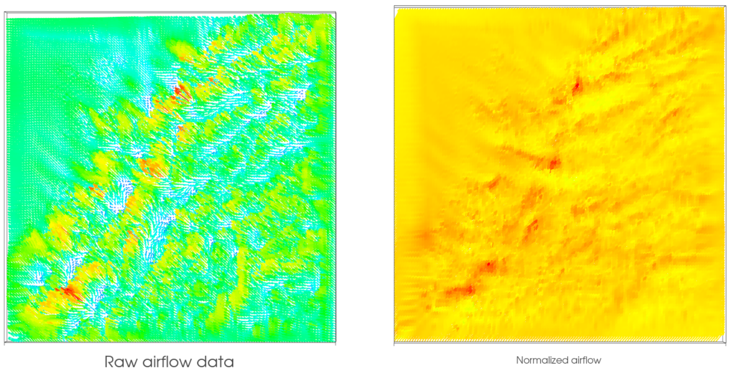
4.3.1 Experiment IIa: Visualizing velocity fields as RGB images

First, a random sample from the training dataset is converted into an RGB image as described in Chapter 3.5. The same normalization factors calculated for Experiment I in Table 4.2 are used to normalize the vector field. This way, the image is represented as seen by the feature extractor in Experiment I. Figure 4.11 shows each step of the transformation for a random sample in the bottom-layer training dataset. Recall that each velocity component is assigned its own color, i.e, $u \to$ red, $v \to$ green, $w \to$ blue. It is clear that green is the most dominant color in this sample, corresponding to the velocity component v along the y-axis in the SIMRA domain. This is not evident from the original vector representation in Figure 4.11a, and is likely due to a change in the dominant velocity components through normalization. While the RGB representation in Figure 4.11b is lacking in both brightness and contrast, the feature extractor is likely to register some of the geometric shapes in the image; Dodge & Karam [7] shows how VGG networks are resilient to contrast distortion, i.e., VGG networks have good classification performance on images with artificially low contrast.

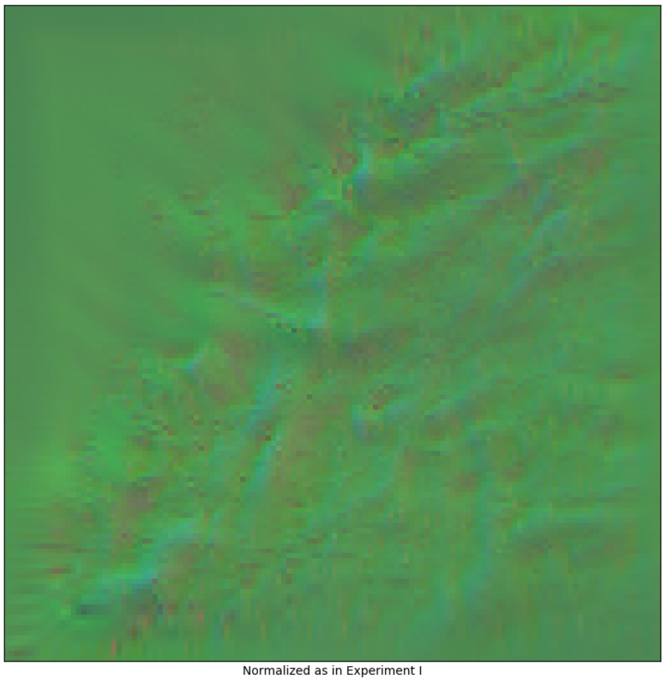
To gain a similar intuition for the whole training dataset, both the mean and variance of each spatial location of the dataset are calculated and represented as RGB images. To emphasize the results, these images are normalized channel-wise according to their own maxima and minima, thus utilizing the full range for each color. Figure 4.12 illustrates the normalized mean velocity field and its spatial variance. Due to the new normalization method, the mean velocity field shows the dominant velocity components at each spatial location in the domain. The image is still predominantly green, but areas containing higher ratios of red and blue can be seen. Color interaction illustrate how terrain impacts the airflow in the domain. The variance in Figure 4.12b shows how the airflow varies more over the sea compared to terrain. Moreover, the yellow color shows how the wind predominantly changes in x- and y- axes, but not in the vertical z-axis.

This short analysis has introduced some perceptual intuition for the ground-level airflow, and how information is conversed after the transformation to RGB image data. As mentioned in Chapter 3, two additional dataset were created for this experiment. The same analysis is performed on datasets “middle-layer” and “top-layer”. Two random samples, one from each dataset, are converted into RGB in the same manner as in Figure 4.11. These are shown in Figure 4.14. With less terrain-induced complexity, the image representations of velocity fields from higher altitudes are significantly harder to interpret.

Figures 4.13a-4.13b show the mean and variance for the dataset collected from the middle-layer dataset (middle of the domain), and Figures 4.13c-4.13d show the same for the top-layer dataset (top of the domain). Notice how both the means and variances become smoother as a result of the higher altitudes. Naturally, these

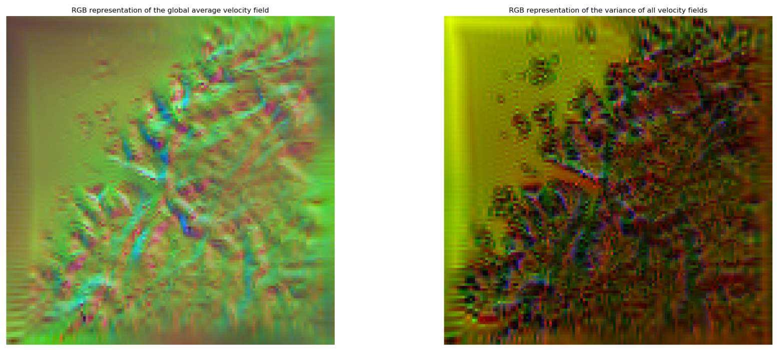


(a) Original and normalized vector field representation.



(b) RGB representation of a normalized vector field.

Figure 4.11: Arbitrary airflow velocity field normalized and translated to RGB.



(a) Mean velocity field. **(b)** Velocity field variance.

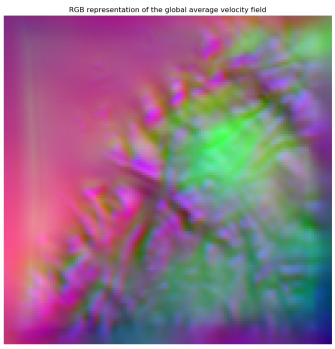
Figure 4.12: Visualizing the mean velocity field and the variance of each velocity component.

airflows are less affected by the terrain compared to Figure 4.12. For a feature extractor such as the one used in Experiment I, however, these smaller sample differences are likely to produce similar feature activations. This is investigated in the following section.

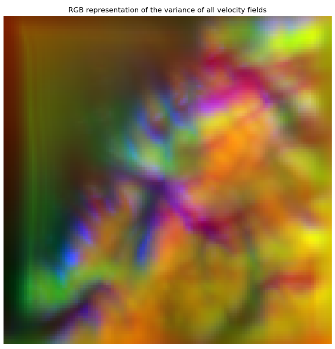
4.3.2 Experiment IIb: Investigating the effect of perceptual features for airflow data.

Despite the low interprebility of how ANNs make decisions in general, it is known that CNNs trained for image classification look for low-level details such as corners and edges [12]. While contrast is not easily quantified, it can easily be identified visually in images. Experiment IIa provided an intuition into how airflow data is perceived from the standpoint of the feature extractor. This experiment investigates how the convolutional feature extractor reacts to different datasets, and what kind of visual information is important for such a network to find meaningful features. Considering that the ESRGAN model in Experiment I weighs perceptual loss 200 times more than adversarial loss (Equation 2.41), it is safe to state that the generator is particularly sensitive to the feedback it receives from the feature extractor. It is important for the generative model's convergence that the feature extractor provides feedback that will improve the quality of its generated samples, i.e., move towards the wanted mapping from $Z \rightarrow T$ (Chapter 2.6).

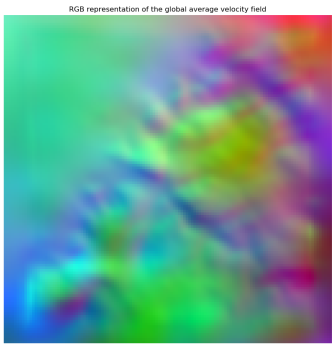
Before investigating airflow datasets, a downloaded subset RGB images from Flickr30k termed "Flickr15.6k" (Table 3.4) is presented to the feature extractor. The output is processed as explained in Chapter 3.5.2. Figure 4.15 shows the feature-space averaged mean and variance across each output of the feature extractor for this dataset. The mean activations (Figure 4.15a) varies between 2.81 and 7.95, and



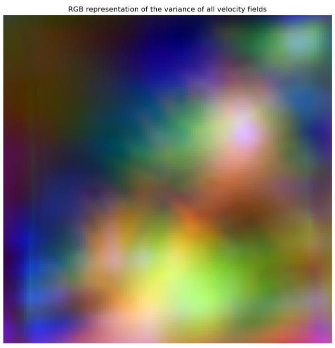
(a) Normalized mean velocity field in the middle-layer dataset.



(b) Normalized variance of the velocity fields in the middle-layer dataset.



(c) Normalized mean velocity field in the top-layer dataset.



(d) Normalized variance of the velocity fields in the top-layer dataset.

Figure 4.13: Visualizing the mean velocity field and the variance of each velocity component in higher-altitude training datasets.

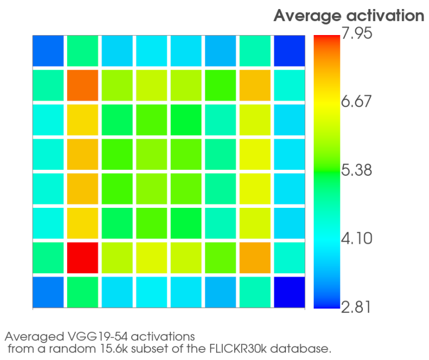
Figure 4.15b shows that these activations have a variance around 20 in the inner part of the image. Variance in the outer edges are smaller. While one could assume that this is attributed to the fact that the feature extractor was pre-trained on images with higher resolution (224x224), an auxiliary experiment using 5200 224x244 Flickr30k images (Figure D.1) shows that this is not the case. Recall zero-padding in convolutional layers shown in Figure 2.6. It is likely that the lower variations along the edges are a result of the local receptive fields passing over zero-padded edges in each layer. Nevertheless, these results provide an intuition for the behaviour of the feature extractor when presented data it is specifically trained to classify.

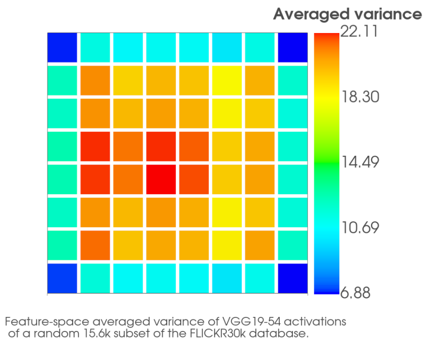
RGB representation of a random sample from Airflow3 dataset.

(a) RGB representation of a sample from the middle-layer dataset.

(b) RGB representation of a sample from the top-layer dataset.

Figure 4.14: Random airflow velocity fields from the middle- and top-layer training datasets are normalized as described for Experiment I and translated to RGB.



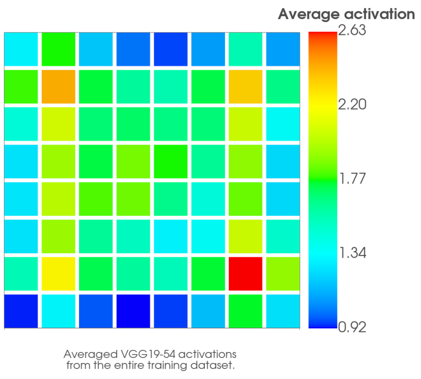


(a) Feature-space averaged mean activations.

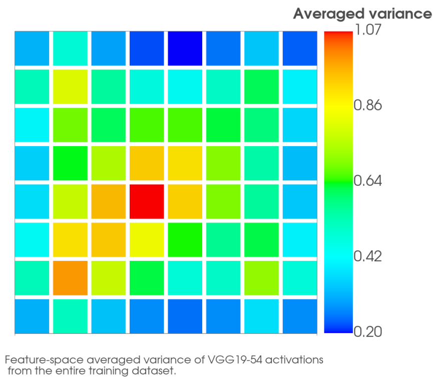
(b) Feature-space averaged variance of activations.

Figure 4.15: VGG19-54 feature extractor output using Flickr15.6k dataset as input.

The same procedure is done for the bottom-layer training dataset used in Experiment I. Figure 4.16 shows the corresponding results. Note that even the highest averaged activation in Figure 4.16a is smaller than the lowest activation in Figure 4.15a. Similarly, the feature-space averaged variances in Figure 4.16b are both smaller and more localized across the diagonal compared to Figure 4.15b. This could be interpreted as spatial areas within the domain where the airflow characteristics are significant enough to elicit richer activations from the feature extractor,



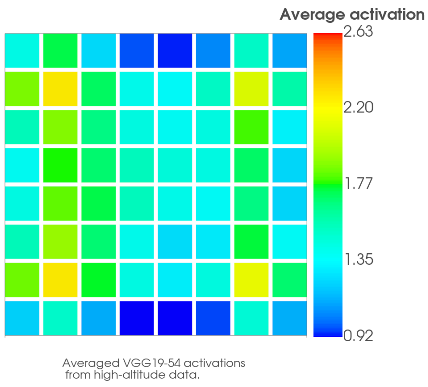
(a) Feature-space averaged mean activations.



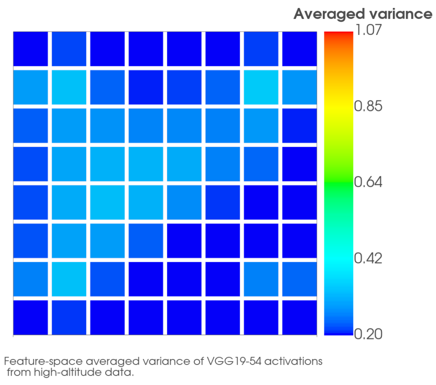
(b) Feature-space averaged variance of activations.

Figure 4.16: VGG19-54 feature extractor output using the bottom-layer dataset as input.

compared to other areas within the same domain.



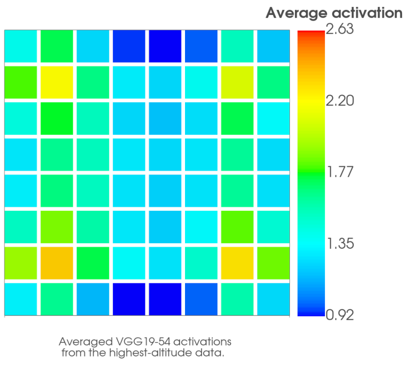
(a) Feature-space averaged mean activations. (Actual range is $[0.84, 2.26]$)



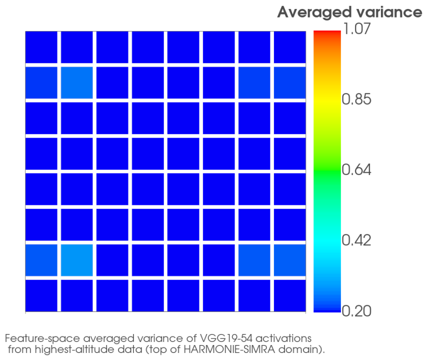
(b) Feature-space averaged variance of activations. (Actual range is $[0.06, 0.35]$)

Figure 4.17: VGG19-54 feature extractor output using middle-layer dataset as input.

When comparing these results with higher-altitude datasets, the colorbar range values from Figure 4.16 are kept to illustrate the differences induced by smoother velocity fields. Figures 4.17 and 4.18 shows the results using the middle- and top-layer training datasets as input, respectively. It is obvious that the smoother high-altitude velocity fields provide significantly less information for the feature extractor, resulting in smaller variations in its feature activations.



(a) Feature-space averaged mean activations. (Actual range is $[0.84, 2.35]$)



(b) Feature-space averaged variance of activations. (Actual range is $[0.03, 0.28]$)

Figure 4.18: VGG19-54 feature extractor output using top-layer dataset as input.

4.3.3 Summary and discussion of Experiment II.

Experiment IIa presented how airflow data used in Experiment I can be visualized in an RGB representation after normalization. This was done to give an intuition for how the normalized airflow data is presented to the feature extractor. Moreover, Figure 4.12 shows how this RGB representation can visualize statistical properties averaged across the normalized training dataset. The same properties were visualized for airflow data sampled from higher altitudes in the SIMRA domain, which were shown to contain significantly smoother spatial properties.[3]

Experiment IIb has shown how the variance in the feature extractor feedback is affected by the complexity in its input data. There is a clear correlation between the decreasing complexity in Figures 4.12 and 4.13 and the corresponding feature-space averaged variance shown in Figures 4.16, 4.17 and 4.18. Furthermore, it was shown that even ground-level airflow data yields significantly smaller variances in feature activations compared to an RGB image dataset. This relatively smaller range of variation is hypothesized to have affected the models in Experiment I to learn initially, but acts as a source of noise later in the training period. In other words, it is assumed that the generators become confused by the different perceptual characteristics present in the airflow data. Consider Figure 4.20a showing the development of perceptual loss in Session 4 of Experiment I. Although the trend decreases exponentially in early training, it is also evident that the oscillations around this trend don't diminish. After a certain point (around 40k iterations in Figure 4.20a), the generator is no longer able to make the super-resolved samples more

[3]These visualizations are also, in the author's opinion, rather mesmerizing.

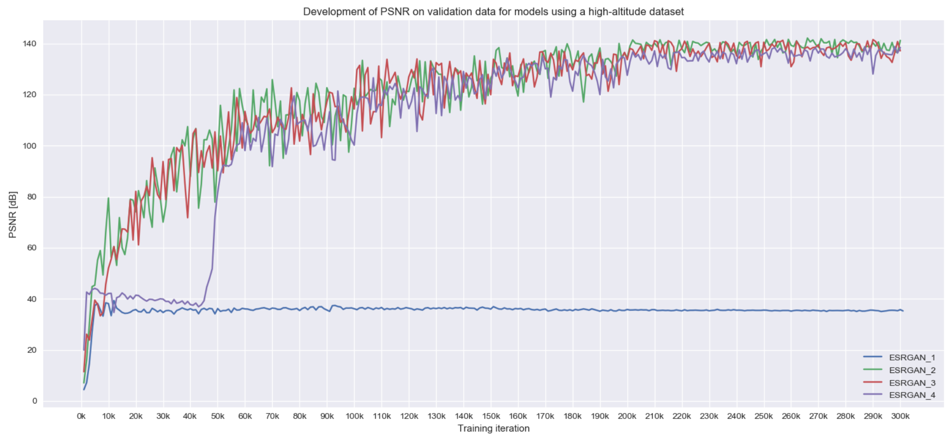
perceptually similar to their corresponding high-resolution velocity fields. Consequently, the perceptual loss acts as a noisy feedback for the generator, preventing it from fine-tuning its outputs.

However, it is not inconceivable that the models in Experiment I are able to reconstruct high-altitude velocity fields, even when using this perceptual loss. Consider that the perceptual loss for the highest-altitude HR data are essentially constant (Figure 4.18b) in comparison. Note that the captions describe the actual activation and variance ranges. As a result, the feature extractor will output similar activations for every training sample. If the generator produces a SR sample with different/significant perceptual characteristics, the resulting perceptual loss will drive the generator to produce samples with little or no such features. As the perceptual loss is driven towards zero this way, the adversarial and pixel/content losses will start to contribute more towards training the generator. It is arguably simpler for the generator to learn a constant perceptual characteristic rather than predicting different characteristics for each sample, especially since the feature activations are based on an entirely different type of data. This is investigated in the following experiment.

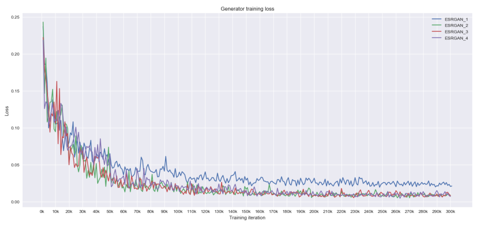
4.4 Experiment III: High-altitude airflow reconstruction with ESRGAN

Results in Experiment II show how the feature-space averaged variances of feature extractor activations become very small for high-altitude airflow data, even compared to the relatively low variance for ground-level data. By decomposing generator losses in Experiment I, it was shown how the perceptual loss acts as a noise towards the end of training. Hypothetically, it should be easier for the generator to ensure similar perceptual characteristics in its output rather than predicting and mimicking high-level features from an entirely different dataset, i.e., ImageNet. Moreover, the high-altitude velocity fields were shown to be significantly smoother due to the lack of interaction with terrain. While smoother velocity fields pose a simpler reconstruction task for the model, the traditional bicubic interpolation method is bound to perform better as well. Therefore, the criteria for the generative model to "succeed" should be more stringent than in Experiment I. Thus, the ESRGAN model is presented the highest-altitude training dataset and is set to reconstruct HR velocity fields in the same manner as in Experiment I. The hyperparameters were chosen based on the results from Session 4 in Experiment I, albeit with instance noise re-enabled (Table 3.5).

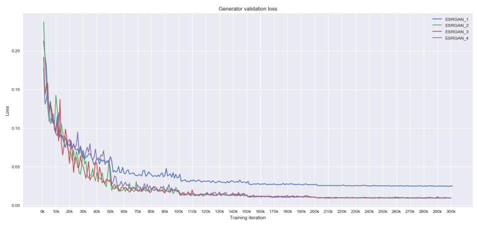
The results of this experiment is shown in Figure 4.19. Three of the models achieve PSNR validation performances of ≈ 140 [dB], which is significantly higher than any of the models in Experiment I. Given that this reconstruction task is less complex than in Experiment I, a higher PSNR performance was expected. Consider the discriminator and generator adversarial losses in Figures 4.19d and 4.20b, respectively. Compared to models in Experiment I, these losses more closely resemble the hypo-



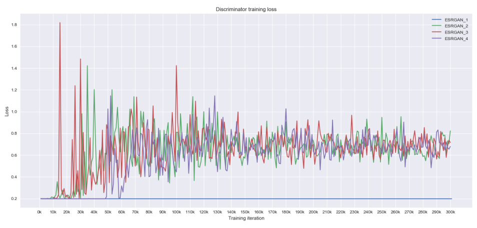
(a) PSNR validation performance.



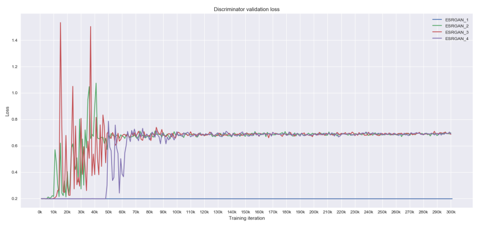
(b) Training loss for generators.



(c) Validation loss for generators.



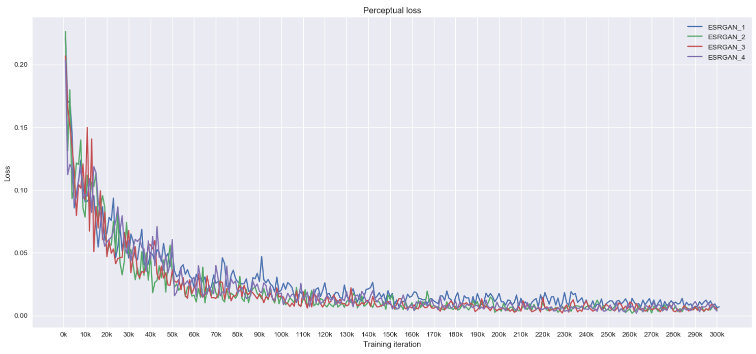
(d) Training loss for discriminators.



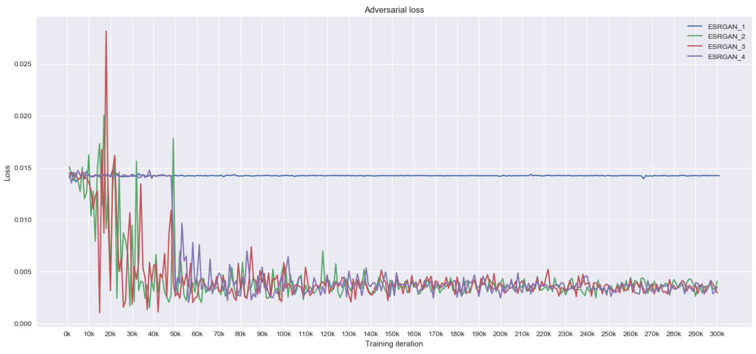
(e) Validation loss for discriminators.

Figure 4.19: Experiment III: Highest-altitude velocity field reconstruction.

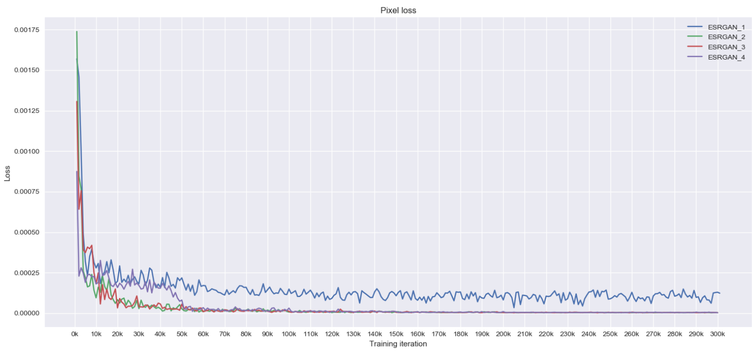
thetical loss development shown in Figure 2.11 in Chapter 2.6. The discriminators have an easy task in the first 20k iterations before the generator starts to produce believable reconstructions. Subsequently, the adversarial losses oscillate with gradually smaller amplitudes as both networks converge. Moreover, de-weighting the generators' adversarial losses, $\approx 0.004 * 200 = 0.8$, shows that the discriminator-generator adversarial losses are in fact balanced. One model, however, seem to have experienced the recurring issue of constant discriminator loss (e.g., "ESRGAN_4" in Session 2 and "LR_cycle@150k without instance noise" in Session 3.)



(a) Perceptual loss component.



(b) Adversarial loss component.



(c) Pixel loss component.

Figure 4.20: Experiment III: Decomposing the generator training loss into its weighted components.

Consider the perceptual component of the generator training loss in Figure 4.20a. It is clear that these models suffer less from noisy feature extractor feedback as in

Experiment I (4.10a). It was hypothesized in Experiment II that the significantly smaller variance in feature activations for this dataset could lead to an easier minimization task for the generative model. The perceptual loss development in these models strengthens this theory. As the generators manage to quickly reduce and stabilize their perceptual loss, the adversarial feedback from the discriminators has a larger impact. As a result, the generators are able to balance minimizing their perceptual and adversarial losses.

4.5 Performance evaluation of models in Experiment I and III.

Up until this point, all models have been evaluated based on their PSNR performance on validation datasets. As all experiments are concluded, the models will be presented their test datasets to finalize the performance evaluation. Given that this model is quite expensive to train, both with respect to time and computational resources, it should perform considerably better than the traditional bicubic interpolation method to consider applying it in real-world applications. In addition to trained models, the original pre-trained model and two downloaded variants of the original ESRGAN (by Wang *et al.* [50]) are tested and used as references. However, the applied evaluation metrics are designed for use on image data. How can the quality of reconstructed airflow data be evaluated using such metrics? Recall that PSNR is notorious for yielding large variations between indistinguishable images. In image compression, for instance, the goal is to remove psychovisual redundant information without removing perceptually important information. Therefore, PSNR is non-ideal for such tasks. In contrast, airflow data contains no redundant information. Consequently, any difference in PSNR, or the MSE as it is based on, represents valid differences in the compared flows. As explained in Chapter 3.7, the calculation of PSNR for denormalized testing data is different due to the change of scale in denormalized data. Using the calculated normalization factors in Table 4.2 yields the following R^2 (simulated max fluctuation) terms:

$$\begin{aligned} R^2_{Ex1} &= (18.99 - (-12.54))^2 &&= 994.14 \\ R^2_{Ex3} &= (36.09 - (-33.64))^2 &&= 4862.27 \end{aligned}$$

where R^2_{Ex1} and R^2_{Ex3} are applied for models in Experiment I and III, respectively.

The LPIPS metric is introduced to quantify a perceptual performance. As explained in Chapter 3.7, the agreement between this metric and PSNR is used to identify whether models tend to ignore psychovisual redundant information rather than learning the governing equations of airflow. Considering the unconventional application of super-resolving fluid data using a perceptual approach, no existing literature has investigated nor justified whether minimizing a perceptual distance is synonymous to learning the relevant fluid dynamics. The spatial property of the LPIPS metric is of particular interest. While differences in MSE and PSNR indicate meaningful errors, they cannot relate the error spatially. Bicubic interpolation is expected to fail for high-frequency details in the airflow, but at least it will fail predictably. Therefore, the spatial LPIPS evaluation aims to find whether the trained models produce errors in a predictable manner.

4.5.1 Test set PSNR and LPIPS agreement evaluation

The test set performances of the 22 trained models in Experiment I and 4 models trained in Experiment III are presented in Table 4.7. Models exceeding the performance of the bicubic interpolation method are marked in **bold** letters. Since PSNR is based on MSE, this is included as well. Overall, few of the trained models in Experiment I exceed bicubic interpolation in PSNR performance. In Experiment III, however, three of the four models significantly exceed bicubic interpolation. Consider Equation 3.7 for calculating PSNR. When values for MSE becomes small, the function starts to increase exponentially. Without an upper bound on the PSNR metric, and no comparable literature due to data-driven normalization of unconventional data, it is hard to estimate the actual quality of the super-resolved velocity fields.

Recall that a perfect reconstruction will yield an LPIPS value of zero, i.e., lower LPIPS is better. The LPIPS metric yields more optimistic results for the most trained models compared to bicubic interpolation. Even the worst performing model from Session 2, which had significantly worse PSNR validation performance during training compared to the other models, yield a smaller mean perceptual difference for its generated samples compared to bicubic interpolation. As explained in Chapter 3.7, LPIPS is included in the performance evaluation to find potential disagreements with the PSNR metric. It was stated that the metrics should agree when the reconstruction is good, and disagree when the model disregards psychovisually redundant information. Several of the models achieve better LPIPS scores than the bicubic interpolation method and simulaneously scores worse for PSNR. This disagreement indicates that these models attempt to minimize the perceptual difference between their super-resolved samples and the corresponding high-resolution velocity field rather than reconstructing the airflow according to the governing equations. LPIPS and PSNR values relating to Experiment III suggest that three of the models achieve near-perfect reconstruction of all test samples. Considering the findings for high-altitude data in Experiment II, the LPIPS metric might not be well suited for this particular dataset. Notice how "ESRGAN1" in Experiment III has a relatively large MSE of 6.14 ± 4.84, while the corresponding LPIPS metric still yields a small value.

4.5.2 Visual inspection of the best, average and worst performing models

Visual inspection of each velocity component can uncover any discrepancies that are not discovered from the applied metrics. Filled contour plots with a discrete color-map was thought to give an easily interpretable representation. Ground-level outputs from the best, an average and the worst performing models according to Table 4.7 are presented to show a range of possible outputs from the model. The outputs were sampled using a random number generator to avoid “cherry picking” the results. These are presented in Figures 4.21, 4.22 and **??**, respectively. Models in Experiment III achieved near-perfect reconstruction capabilities according to MSE and LPIPS. Figure 4.23 shows that the bicubic interpolation arguably reconstructs the high-resolution velocity field better than the generative model despite achieving worse quality metrics. Thus, the applied metrics become inaccurate for near-perfect reconstruction.

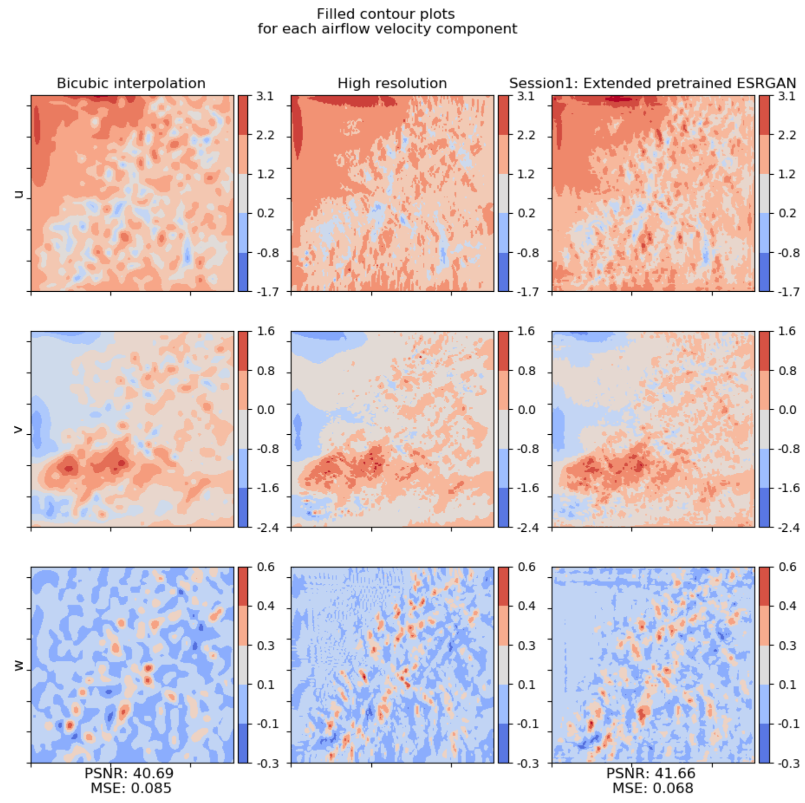


Figure 4.21: Filled contour plot comparison for each velocity component. Left column: Bicubic interpolation, Middle column: High-resolution reference. Right column: Best model from Session 1. Each row represents the different velocity components, u, v and w. The generative model is mostly able to reconstruct general flow structures, but not for high-frequency details.

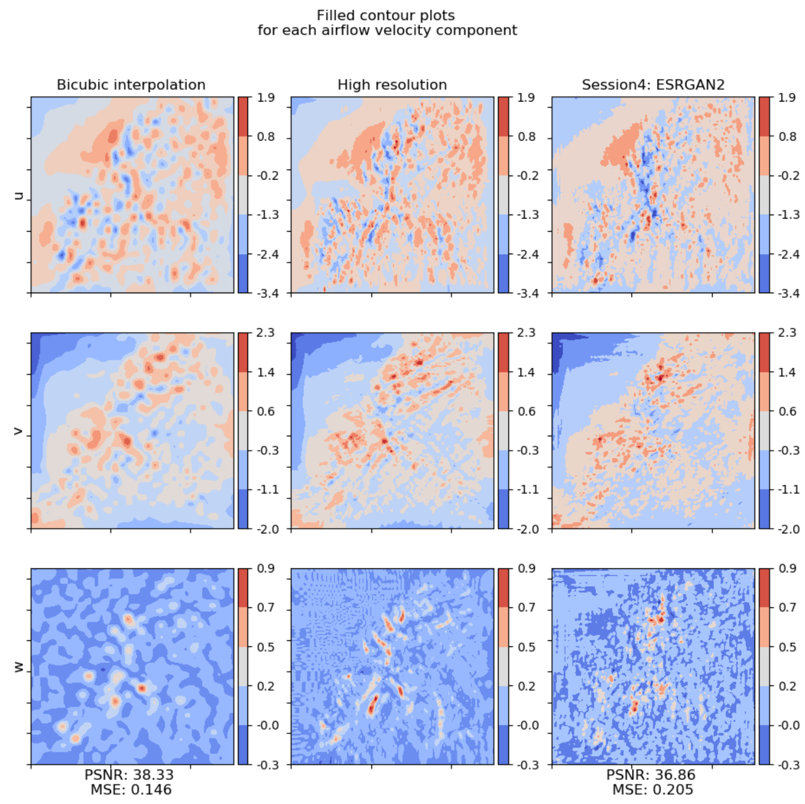


Figure 4.22: Filled contour plot comparison for each velocity component. Left column: Bicubic interpolation, Middle column: High-resolution reference. Right column: Average model from Session 4. Each row represents the different velocity components, u, v and w. The generative model recreates general flow structures but fails to do so accurately.

4.5.3 Spatial LPIPS performance

Chapter 2.7 described that *localized* perceptual differences can be found by removing the spatial averaging in Equation 2.35. Calculating spatial LPIPS for the full test set and averaging sample-wise can uncover areas within the domain where a model consistently fails to match the corresponding high-resolution velocity field. To visually compare the models, the color-map is scaled with respect to the maximum LPIPS value found this way for the bicubic interpolation method, which is presented in Figure 4.24. Similar to Table 4.7, lower spatial LPIPS values mean smaller differences between super-resolved and high-resolution. This figure shows that the bicubic interpolation method tends to produce the most perceptually different airflow characteristics in areas where the terrain is more complex. Consider that the method does not take any domain knowledge into consideration. These

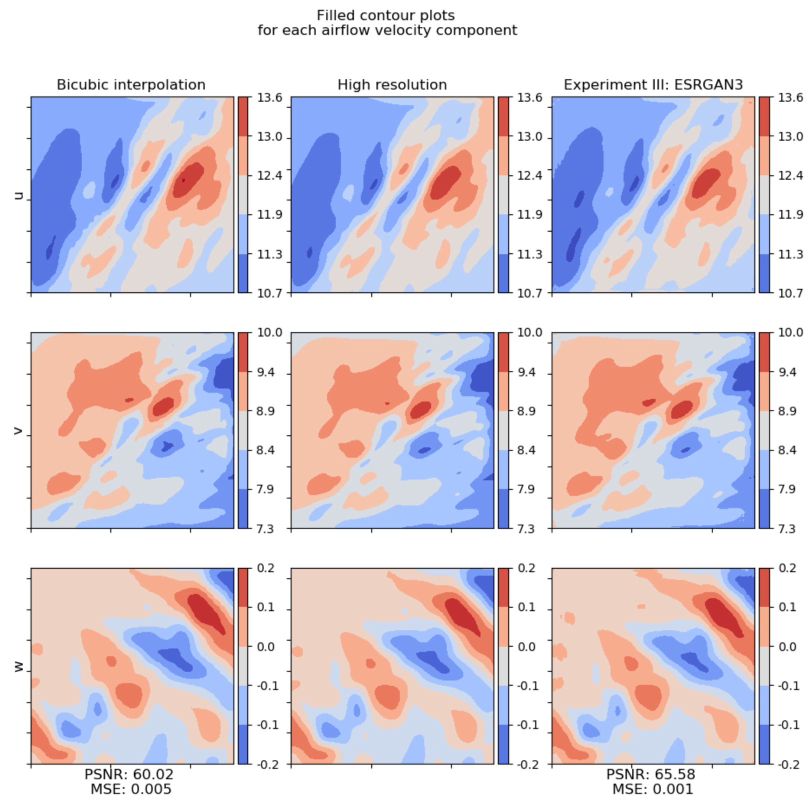
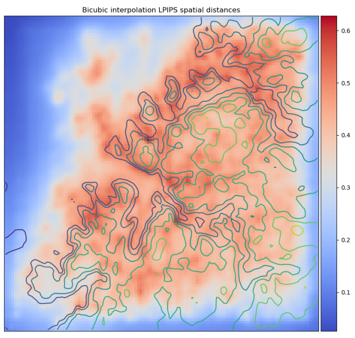
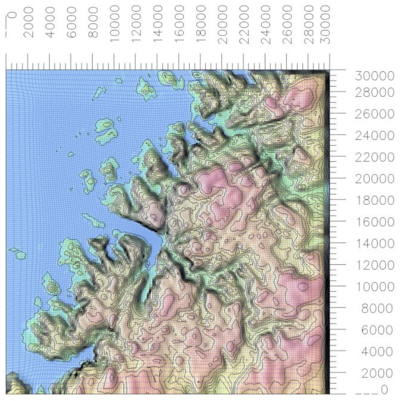


Figure 4.23: Filled contour plot comparison for each velocity component. Left column: Bicubic interpolation, Middle column: High-resolution reference. Right column: Best model from Experiment III (top-level airflow). Each row represents the different velocity components, u, v and w. Although bicubic interpolation scores worse wrt. quality metrics, its reconstructed airflow more closely match the high-resolution reference compared to the generative model.

results show that the bicubic smoothing of turbulent and complex flow elicit perceptual differences compared to the true high-resolution airflow.

The best and worst performing models with respect to LPIPS in Experiment I and III, including the other reference models, are tested in the same way as bicubic interpolation. Figure 4.25 shows the spatial LPIPS differences for these models in Experiment I, where the best model (left) yields significantly smaller and less localized perceptual differences compared to bicubic interpolation (right). In contrast, the corresponding errors for the worst model (middle) are less correlated with terrain complexity and generally larger. These representations provide valuable supplementary information to the mean LPIPS values in Table 4.7.

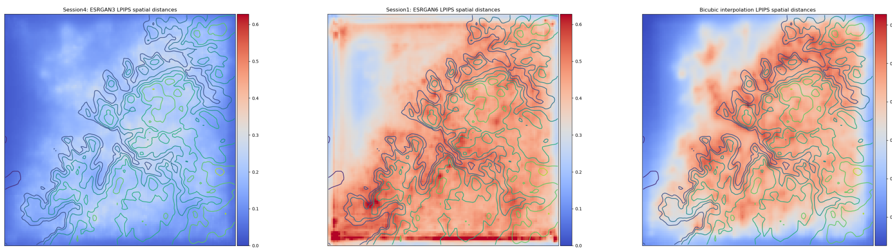




(a) Localized errors.

(b) SIMRA domain

Figure 4.24: Spatial LPIPS with terrain overlay for the bicubic interpolation method. Added SIMRA domain for reference.



(a) Best: Session 4 - ESRGAN3.

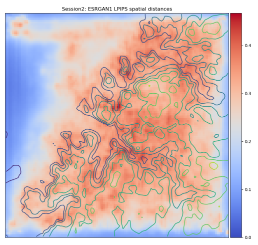
(b) Worst: Session 1 - ESRGAN6.

(c) Reference: Bicubic interpolation.

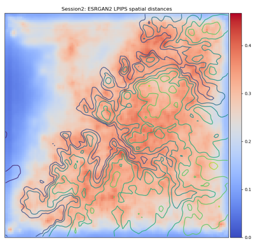
Figure 4.25: Spatial LPIPS with terrain overlay: Best and worst models of Experiment I vs. BC.

For Session 2, models “ESRGAN1”, “ESRGAN2” and “ESRGAN3” achieved similar LPIPS scores in Table 4.7. Figure 4.26 compares the errors of these models in the spatial domain. Note that the color-map for these models are scaled relative to the models’ own maximum error. In other words, this figure is not meant to compare against bicubic interpolation. Although their performances differ slightly, the spatial errors appear consistent between the models. This shows that when independent models achieve similar LPIPS performances, their corresponding spatial errors are consistent.

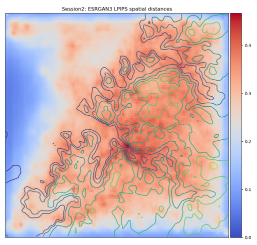
Corresponding representations are created for models in Experiment III using the highest-altitude test dataset. Note that the altitude considered for these models



(a) ESRGAN1.



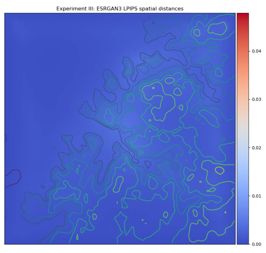
(b) ESRGAN2.



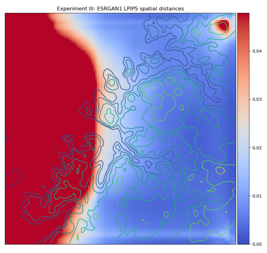
(c) ESRGAN3.

Figure 4.26: Spatial LPIPS: Spatial error differences between similar models in Session 2.

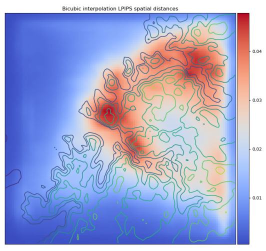
don't have an actual contour for terrain, but the ground-level contour is included to represent the domain below. Figure 4.27 illustrates how bicubic interpolation (right) fails in a similar manner as for ground-level airflows, i.e, surrounding complex terrain, though with considerably lower error rates. In contrast, super-resolved velocity fields from the best model (left) yields almost no perceptual difference. The worst model (middle), however, yields a rather strange result. Recall from Experiment III that this model experienced constant discriminator losses and a constant PSNR validation performance throughout training. Interestingly, the model's perceptual loss (Figure 4.20a) converged along with the other models, indicating that minimizing the perceptual loss is not necessarily synonymous to achieving accurate super-resolved velocity fields.



(a) Best: ESRGAN3.



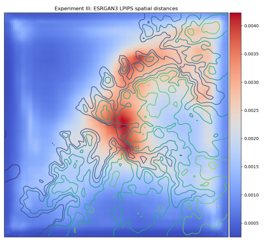
(b) Worst: ESRGAN1.



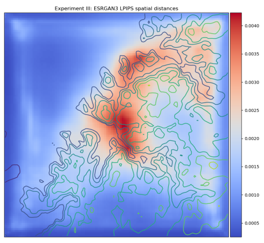
(c) Reference: Bicubic interpolation.

Figure 4.27: Spatial LPIPS with terrain overlay: Best and worst models of Experiment III vs. BC.

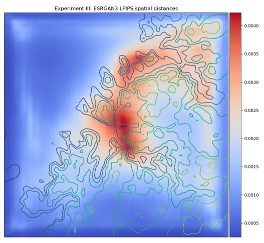
Furthermore, consider that the LPIPS performance for three of the models in Experiment III are identical in Table 4.7 and even close to representing perfect reconstruction capabilities. Figure 4.28 illustrates that the small errors are consistent in the spatial domain. Compared to the corresponding comparison for models in Experiment I (Figure 4.26), these models have an even closer correlation. This shows great promise for the model's ability to consistently reproduce its results when its LPIPS scores become small. Some error patterns can be seen along the edges of the domain. Nevertheless, the associated errors are relatively small, but should still be considered in future work.



(a) ESRGAN2.



(b) ESRGAN3.



(c) ESRGAN4.

Figure 4.28: Spatial LPIPS: Consistent spatial errors between similar models in Experiment III.

Table 4.7: PSNR and LPIPS agreement evaluation for all trained models in Experiment I and III.

Session	Model	PSNR [dB]		MSE		VGG LPIPS	
		mean	±std	mean	±std	mean	±std
Reference models	**Bicubic interp.**	**38.21**	**3.20**	**0.20**	**0.16**	**0.35**	**0.14**
	Pretrained model	38.01	3.01	0.20	0.17	0.26	0.10
	Original ESRGAN	27.21	3.77	2.77	2.77	0.52	0.17
	PSNR ESRGAN	36.17	3.40	0.33	0.29	**0.34**	**0.15**
Session 1	ESRGAN1	29.27	1.31	1.24	0.46	0.38	0.11
	ESRGAN2	37.68	3.02	0.22	0.18	**0.25**	**0.09**
	ESRGAN3	29.04	1.05	1.28	0.37	**0.33**	**0.09**
	ESRGAN4	28.74	0.94	1.36	0.35	0.40	0.11
	ESRGAN5	37.10	2.93	0.25	0.20	**0.29**	**0.11**
	ESRGAN6	26.72	1.00	2.18	0.57	0.43	0.11
	ESRGAN7	32.67	1.70	0.59	0.29	0.35	0.09
	ESRGAN8	27.98	1.11	1.64	0.50	0.42	0.11
	Ext. pretrained	**39.21**	**3.01**	**0.16**	**0.13**	**0.17**	**0.07**
Session 2	ESRGAN1	37.12	2.96	0.25	0.19	**0.27**	**0.10**
	ESRGAN2	37.35	2.98	0.23	0.19	**0.26**	**0.10**
	ESRGAN3	37.30	2.89	0.23	0.18	**0.27**	**0.10**
	ESRGAN4	35.48	2.65	0.35	0.26	**0.31**	**0.10**
	Ext. pretrained	**39.03**	**3.05**	**0.16**	**0.14**	**0.18**	**0.08**
	Ext. worst noisy	**38.39**	**2.96**	**0.19**	**0.15**	**0.21**	**0.09**
	Ext. worst	**38.59**	**2.99**	**0.18**	**0.14**	**0.21**	**0.09**
Session 3	Cycle 50k, hard	35.76	2.94	0.34	0.27	**0.28**	**0.11**
	Cycle 50k, soft	36.40	2.92	0.29	0.23	**0.26**	**0.11**
	Cycle 150k, no noise	34.15	1.89	0.43	0.23	**0.30**	**0.09**
	Cycle 150k	36.41	3.01	0.30	0.24	**0.29**	**0.11**
Session 4	ESRGAN1	37.52	3.27	0.24	0.21	**0.20**	**0.09**
	ESRGAN2	36.47	3.16	0.29	0.24	**0.25**	**0.12**
	ESRGAN3	**39.02**	**3.25**	**0.17**	**0.15**	**0.16**	**0.08**
Ex. III	**Bicubic interp.**	**58.40**	**4.24**	**0.012**	**0.017**	**0.017**	**0.022**
	ESRGAN1	35.14	11.38	6.14	4.84	0.075	0.156
	ESRGAN2	**69.08**	**3.18**	**0.001**	**0.001**	**0.002**	**0.004**
	ESRGAN3	**67.08**	**2.55**	**0.001**	**0.001**	**0.002**	**0.003**
	ESRGAN4	**68.48**	**3.17**	**0.001**	**0.001**	**0.002**	**0.004**

Chapter 5

Conclusions and further work

5.1 Answering the research questions

The answer to each of the following research questions concludes the work done in this thesis, and proposals for further work are presented.

Why does the previously proposed GAN-based super-resolving model consistently fail to reproduce its results?
Experiment I conducted a stability analysis of the previously proposed model for super-resolving low-resolution velocity fields using the perceptually driven SISR model, ESRGAN. Through hyperparameter tuning, learning rate cycling, and doubling the number of training iterations, it was found that the perceptual component in the generator loss function consistently failed to converge for all models.

What is the fundamental issue with applying the pre-trained feature extractor to airflow data?
The previous work failed to justify whether a perceptual feature extractor pre-trained on ImageNet applies to the airflow velocity data considered for this model. Experiment II investigated the difference in perceptual characteristics between RGB image data in Flickr30k, ground-level airflow training data in Experiment I, and two high-altitude training datasets created for the same date range. Translating the airflow velocity data to RGB provided an intuition for the visual complexity of airflow data as presented to the feature extractor. Additionally, the training datasets were averaged and normalized independently to visualize the spatial correlation and variance between samples using the RGB representation. It was shown that the ground-level airflow data yielded significantly less variance in feature activations compared to the RGB dataset.

Furthermore, the higher-altitude training datasets yielded considerably less variance in feature activations, even near-constant. Experiment III considered the same model in Experiment I applied to the highest-altitude dataset to investigate whether rich variations in perceptual feature activations are beneficial or inhibiting for the generative model. Compared to models in Experiment I, the models in Experiment III achieved a significantly higher PSNR performance, discriminator loss progressions corresponded to the hypothetical minimax game illustration (Figure 2.11) and a converging perceptual loss. Empirical evidence suggests that higher variance in the perceptual feature activations makes a more difficult task for the generator to solve when there is a significant difference between the dataset used to train the feature extractor versus the dataset available to the generative model.

Is the generative model's task of minimizing a perceptual difference synonymous to learning the governing equations of airflow in the relevant domain?
Even though the PSNR validation performances in Experiment III were promising, the values alone are hard to interpret. The performance evaluation considered an agreement comparison between the PSNR and LPIPS metrics. The criterion to be met was based on the agreement between a higher PSNR than bicubic interpolation, and a corresponding lower LPIPS score. It was shown that most models in Experiment I failed this agreement criterion, except for models with extended training, i.e., started their training from the end state of a pre-trained generator. Considering that the original ESRGAN model [50] performs initial pre-training of their generative model before introducing the adversarial framework, this addition may benefit the model. However, these "extended" models still performed only *marginally* better than bicubic interpolation. Spatial LPIPS errors averaged sample-wise over test data for the bicubic interpolation method were illustrated, and it was shown that perceptual distance error values correspond to areas of more complex terrain. Comparing trained models with similar LPIPS scores showed that localized errors are spatially correlated. This shows promise for the consistency of the model output, despite the imperfect reconstruction quality. Furthermore, models in Experiment III resulted in near-perfect reconstruction quality both with respect to LPIPS and MSE, with corresponding near-perfect spatial correlation in LPIPS error. Through visual inspection, it was shown that the applied metrics become inaccurate when the airflow reconstruction is near-perfect.

5.2 Further work

Ultimately, this thesis concludes that the proposed perceptually driven generative-adversarial framework is unsuited for reconstructing airflow in complex terrain. Empirical evidence suggests that the model is capable of reconstructing smoother airflow based on the fact that the output from the feature extractor is near-constant. Therefore, it is possible that the network architecture itself is sufficient, and the fundamental issue is attributed to the RGB data used to train the feature extractor. Since airflow data cannot be used to form a supervised learning problem, a feature extractor explicitly trained for airflow cannot be created in the same way as for RGB image data. There is, to the author's knowledge, one exception; Chu & Thuerey [5] utilizes intermediate convolutional features extracted from the *discriminator* in their proposed generative model for super-resolving coarse fluid simulations with rich turbulence details. Consider that the discriminator essentially solves a supervised learning task, i.e., classifies between "real" and "fake" flow fields, the resulting convolutional network is similar both in structure and task as the pre-trained feature extractor in this thesis. Assuming that the generative-adversarial model avoids the typical issues such as discriminator overfitting, mode collapse, and convergence failure, the resulting discriminator should become an expert in discovering high-level features attributed to real flow, i.e., it learns the governing equations. Using the convolutional feature extractor part of the discriminator can provide the generative model with expert knowledge directly related to the airflow data, rather than the corresponding RGB-related high-level features found by the feature extractor used in this thesis.

Sanchez-Gonzalez *et al.* [43] propose a particle-based "Graph Network-based Simulator" (GNS) that can learn complex fluid physics for different viscosities and simulates realistic fluid interaction with a presented environment. Adopting this model using airflow and including the SIMRA terrain data can be considered by future researchers.

Bibliography

1. Arjovsky, M. & Bottou, L. Towards Principled Methods for Training Generative Adversarial Networks. *stat* **1050** (Jan. 2017).
2. Arjovsky, M., Chintala, S. & Bottou, L. Wasserstein GAN (Jan. 2017).
3. Bellman, R. & Corporation, R. *Adaptive Control Processes: a Guided Tour: R-350* (Rand Corporation, 1961).
4. Blau, Y., Mechrez, R., Timofte, R., Michaeli, T. & Zelnik-Manor, L. *The 2018 PIRM Challenge on Perceptual Image Super-resolution* 2018. eprint: `1809.07517`.
5. Chu, M. & Thuerey, N. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics* **36,** 1–14. ISSN: 1557-7368. `http://dx.doi.org/10.1145/3072959.3073643` (July 2017).
6. Deng, J. *et al. Imagenet: A large-scale hierarchical image database* in *2009 IEEE conference on computer vision and pattern recognition* (2009), 248–255.
7. Dodge, S. & Karam, L. *Understanding how image quality affects deep neural networks* in (June 2016), 1–6.
8. Dong, C., Loy, C. C., He, K. & Tang, X. *Image Super-Resolution Using Deep Convolutional Networks* 2014.
9. Dumoulin, V. & Visin, F. A guide to convolution arithmetic for deep learning (Mar. 2016).
10. Goodfellow, I. *NIPS 2016 Tutorial: Generative Adversarial Networks* 2016. arXiv: `1701.00160 [cs.LG]`.
11. Goodfellow, I. J. *et al. Generative Adversarial Networks* 2014. eprint: `1406.2661`.
12. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* `http://www.deeplearningbook.org` (MIT Press, 2016).
13. Google. *Machine Learning Course: Generative Adversarial Networks* (Accessed: 05.2020). Apr. 2019. `https://developers.google.com/machine-learning/gan/training`.
14. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. & Courville, A. *Improved Training of Wasserstein GANs* 2017. arXiv: `1704.00028 [cs.LG]`.
15. *HARMONIE-SIMRA Database* (Accessed 04.2020). `https://thredds.met.no/thredds/fileServer/opwind/`.

16. Hastie, T., Tibshirani, R. & Friedman, J. *The Elements of Statistical Learning* ISBN: 9780387848587 (Springer, 2009).
17. He, K., Zhang, X., Ren, S. & Sun, J. *Deep Residual Learning for Image Recognition* in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016), 770–778.
18. Iandola, F. N. *et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size* 2016. arXiv: `1602.07360 [cs.CV]`.
19. Irani, M. & Peleg, S. Improving resolution by image registration. *CVGIP: Graphical Models and Image Processing* **53,** 231–239. ISSN: 1049-9652 (1991).
20. Johnson, J., Alahi, A. & Fei-Fei, L. *Perceptual Losses for Real-Time Style Transfer and Super-Resolution* 2016. eprint: `1603.08155`.
21. Jolicoeur-Martineau, A. *The relativistic discriminator: a key element missing from standard GAN* 2018. eprint: `1807.00734`.
22. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2014. arXiv: `1412.6980 [cs.LG]`.
23. Krizhevsky, A., Sutskever, I. & Hinton, G. E. in *Advances in Neural Information Processing Systems 25* 1097–1105 (Curran Associates, Inc., 2012).
24. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* **60,** 84–90. ISSN: 0001-0782. `https://doi.org/10.1145/3065386` (May 2017).
25. Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* **86,** 2278–2324 (Dec. 1998).
26. LeCun, Y. & Cortes, C. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/` (2010).
27. Ledig, C. *et al. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network* 2016.
28. Leong, W. J. & Horgan, H. J. DeepBedMap: Using a deep neural network to better resolve the bed topography of Antarctica. *The Cryosphere Discussions* **2020,** 1–27. `https://www.the-cryosphere-discuss.net/tc-2020-74/` (2020).
29. Mescheder, L., Geiger, A. & Nowozin, S. *Which Training Methods for GANs do actually Converge?* 2018. arXiv: `1801.04406 [cs.LG]`.
30. Mirza, M. & Osindero, S. Conditional Generative Adversarial Nets (Nov. 2014).
31. Nagarajan, V. & Kolter, J. Z. *Gradient descent GAN optimization is locally stable* 2017. arXiv: `1706.04156 [cs.LG]`.
32. Nielsen, M. *Neural Networks and Deep Learning* (accessed 20.04.2020). `http://neuralnetworksanddeeplearning.com/index.html` (Determination Press, 2015).
33. Nocedal, J. & Wright, S. J. *Numerical Optimization* ISBN: 0387303030 (Springer, 2006).
34. Paszke, A. *et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library* 2019. arXiv: `1912.01703`.

35. Purves, D. *et al. Neuroscience* ISBN: 9780878936953 (Oxford University Press, 2012).
36. Radford, A., Metz, L. & Chintala, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks (Nov. 2015).
37. Ramachandran, P. & Varoquaux, G. Mayavi: 3D Visualization of Scientific Data. *Computing in Science & Engineering* **13,** 40–51. ISSN: 1521-9615 (2011).
38. Rasheed, A., Süld, J. & Kvamsdal, T. A Multiscale Wind and Power Forecast System for Wind Farms. *Energy Procedia* **53** (Dec. 2014).
39. Rasheed, A., Tabib, M. & Kristiansen, J. *Wind Farm Modeling in a Realistic Environment Using a Multiscale Approach* June 2017. https://doi.org/10.1115/OMAE2017-61686.
40. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning Representations by Back-propagating Errors. *Nature* **323,** 533–536. http://www.nature.com/articles/323533a0 (1986).
41. Russell, S. & Norvig, P. *Artificial Intelligence: A Modern Approach* 3rd. ISBN: 0136042597, 9780136042594 (Prentice Hall Press, Upper Saddle River, NJ, USA, 2009).
42. Salimans, T. *et al. Improved Techniques for Training GANs* 2016. arXiv: 1606.03498 [cs.LG].
43. Sanchez-Gonzalez, A. *et al. Learning to Simulate Complex Physics with Graph Networks* 2020. arXiv: 2002.09405 [cs.LG].
44. Simonyan, K. & Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556* (Sept. 2014).
45. Själander, M., Jahre, M., Tufte, G. & Reissmann, N. *EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure* 2019. arXiv: 1912.05848 [cs.DC].
46. Sønderby, C. K., Caballero, J., Theis, L., Shi, W. & Huszár, F. *Amortised MAP Inference for Image Super-resolution* 2016. arXiv: 1610.04490 [cs.CV].
47. Tran, D. T. *et al. GANs enabled super-resolution reconstruction of wind field* (Unpublished). 2019.
48. Tyagi, V. *Understanding Digital Image Processing* ISBN: 9781351342667 (CRC Press, 2018).
49. Vesterkjær, E. E. *GAN implementation for single image super resolution.* Accessed (01.2020). 2019. %7Bhttps://github.com/eirikeve/esrdgan%7D.
50. Wang, X. *et al. ESRGAN: Enhanced super-resolution generative adversarial networks* in *The European Conference on Computer Vision Workshops (ECCVW)* (Sept. 2018).
51. Young, P., Lai, A., Hodosh, M. & Hockenmaier, J. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *TACL* **2,** 67–78 (2014).
52. Zhang, R., Isola, P., Efros, A. A., Shechtman, E. & Wang, O. *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric* in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), 586–595.

Appendix A

Overview of appendices

The appendix contains additional information considered excessive to include in the main part of the thesis. Its contents are meant to provide additional information for interested researchers.

Appendix B

Accessing the contents of netCDF data files

This appendix explains some additional information regarding the NetCDF files generated by the HARMONIE-SIMRA system used to form the datasets in this thesis. Each file has the naming convention "simra_BESSAKER_[YYYYMMDD]T[HH]Z.nc", where [YYYYMMDD] is replaced with a past date, and [HH] is replaced with "00" or "12" depending on whether the file contains data from 00:00-12:00 or 12:00-24:00, respectively. These filed predates back to 04.08.2017, and are continuously generated at the time of writing. Table B.1 contains all variables available in each file. Using the netcdf4 library for Python3, the file is loaded into a netcdf4.Dataset data structure. Variables can then be accessed from the netcdf4.Dataset.

Table B.1: Available data generated by the coupled HARMONIE-SIMRA system

Variable	Description
time(time)	Time of day
forecast_reference_time()	NA
l(l)	Vertical separator for horizontal planes
rotated_latitude_longitude()	NA
x(x)	x-axis
y(y)	y-axis
longitude(y,x)	NA
latitude(y,x)	NA
geopotential_height_ml(time,l,y,x)	Geopotential height
surface_altitude(y,x)	NA
air_potential_temperature_ml(time,l,y,x)	NA
x_wind_ml(time,l,y,x)	Wind velocity along x-axis (u)
y_wind_ml(time,l,y,x)	Wind velocity along y-axis (v)
upward_air_velocity_ml(time,l,y,x)	Wind velocity along z-axis (w)
x_wind_10m(time,y,x)	NA
y_wind_10m(time,y,x)	NA
air_pressure_ml(time,l,y,x)	Pressure
surface_air_pressure(time,y,x)	Pressure at surface-level
turbulence_index_ml(time,l,y,x)	NA
turbulence_dissipation_ml(time,l,y,x)	Kinetic energy loss due to turbulence

Dimensions(sizes): x(135), y(136), l(41), time(13)

Appendix C

GAN-related distances, divergences and algorithms

This appendix provides mathematical descriptions of different distances and divergences used in GAN context, as described by Arjovsky *et al.* [2].

$$\delta(T,T') = \sup_{A\in\sum} |T(A) - T'(A)| \qquad \text{Total Variation (TV) distance} \tag{C.1}$$

$$KL(T||T') = \int \log\left(\frac{T(x)}{T'(x)}\right) T(x)d\mu(x) \quad \text{Kullback-Leibler (KL) divergence} \tag{C.2}$$

$$JS(T,T') = KL(T||T_m) + KL(T'||T_m) \qquad \text{Jensen-Shannon (JS) divergence} \tag{C.3}$$

$$W(T,T') = \inf_{\gamma\in\prod(T,T')} \mathbb{E}_{(x,y)} \sim \gamma\left[||x-y||\right] \qquad \frac{\text{Earth Mover's Distance (EMD)}}{\text{or Wasserstein-1}} \tag{C.4}$$

Algorithm 4: Training algorithm for WGAN. Rephrased from its definition in [2].

```
1  Hyperparameters : $\alpha$, the learning rate. $c$, the weight clipping
     parameter. $m$, the batch size. $n_{\text{critic}}$, the number of iterations of the critic
     per generator iteration.
2  Require : $\theta_{C_0}$, initial critic parameters. $\theta_{G_0}$, initial generator parameters.
3  while $\theta_G$ has not converged do
4      for $t = 0, ..., n_{critic}$ do
5          • Sample batch of $m$ noise samples $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from prior $Z$.
6          • Sample batch of $m$ examples $\{\mathbf{t}^{(1)}, ..., \mathbf{t}^{(m)}\}$ from data generating,
             *true*, distribution $T$.
7          • Update the critic using RMSProp:
8              $\Delta_C \leftarrow \nabla_{\theta_C} \left[\frac{1}{m}\sum_{i=1}^{m} C\left(\mathbf{t}^{(i)}\right) - \frac{1}{m}\sum_{i=1}^{m} C\left(G\left(\mathbf{z}^{(i)}\right)\right)\right]$
9              $\theta_C \leftarrow \theta_C + \alpha \cdot \text{RMSProp}(\theta_C, \Delta_C)$
10             $\theta_C \leftarrow \text{clip}(\theta_C, -c, c)$
11     end
12     • Sample batch of $m$ noise samples $\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}\}$ from prior $Z$.
13     • Update the generator using RMSProp:
14         $\Delta_G \leftarrow -\nabla_{\theta_G} \frac{1}{m}\sum_{i=1}^{m} \left[C\left(G\left(\mathbf{z}^{(i)}\right)\right)\right]$
15         $\theta_G \leftarrow \theta_G - \alpha \cdot \text{RMSProp}(\theta_G, \Delta_G)$
16 end
```

Appendix D

Auxilliary feature extractor experiment

Experiment IIb raised a suspicion regarding smaller variances along the outer spatial axes of VGG19-54 feature activations. Since the applied VGG19 network was pretrained on 224x244 images, an additional Flickr30k subset was created. Using 5200 224x244 images, the feature-space averaged variance was found in the same manner as in Experiment IIb. The results are shown in Figure D.1.

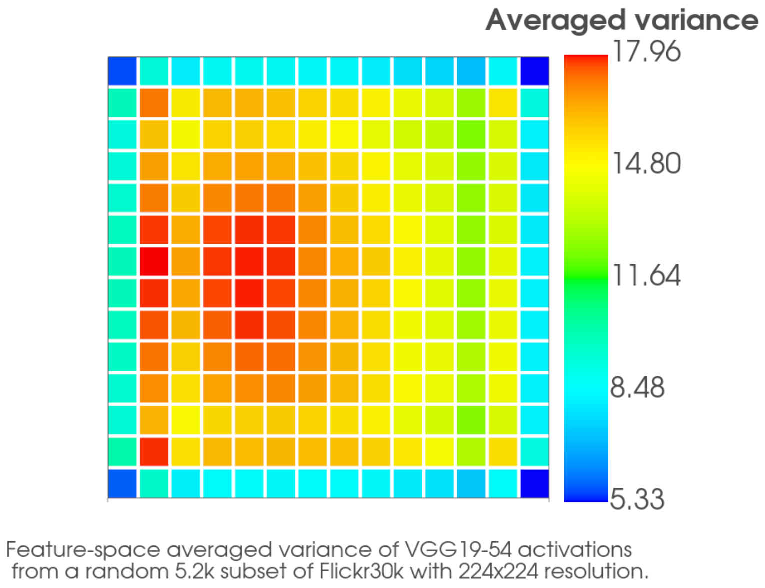


Figure D.1: Feature-space averaged variance of VGG19-54 activations using a 224x224 resolution subset of Flick30k.

Appendix E

Software requirements

This appendix contains the core software modules and Python libraries used in this thesis.

Table E.1: Core software modules used on the IDUN HPC Cluster.

Software module	Version
GCC	8.2.0-2.31.1
CUDA	10.1.105
Python	3.7.2

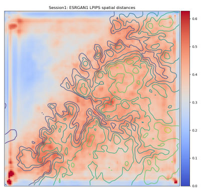
Table E.2: Python 3.7.2 - Software requirements

Python library	Version
cycler	0.10.0
joblib	0.14.1
kiwisolver	1.1.0
matplotlib	3.1.3
numpy	1.18.1
opencv-python	4.2.0.32
Pillow	7.0.0
progressbar2	3.47.0
protobuf	3.11.3
psutil	5.7.0
pyparsing	2.4.6
python-dateutil	2.8.1
python-utils	2.3.0
scikit-learn	0.22.1
scipy	1.4.1
six	1.14.0
tensorboardX	2.0
torch	1.4.0
torchvision	0.5.0

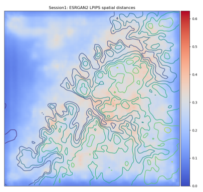
Appendix F

Spatial LPIPS evaluation for all models

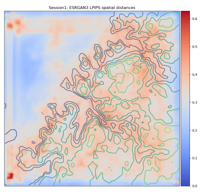
Only a few figures illustrating the sample-mean spatial LPIPS differences were presented in Chapter 4 to avoid clutter. Corresponding figures for all models are presented here.



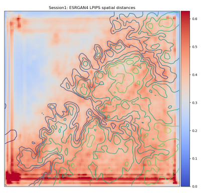
(a) ESRGAN1.



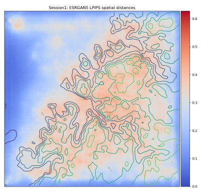
(b) ESRGAN2.



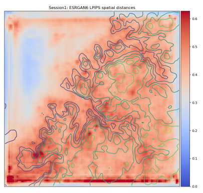
(c) ESRGAN3.



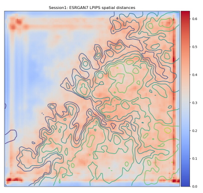
(d) ESRGAN4.



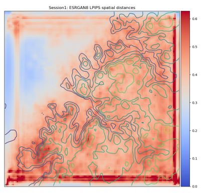
(e) ESRGAN5.



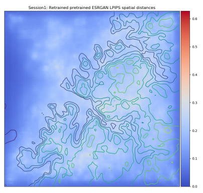
(f) ESRGAN6.



(g) ESRGAN7.

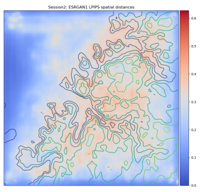


(h) ESRGAN8.

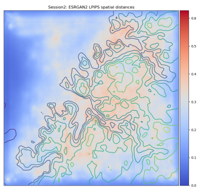


(i) Ext. pretrained.

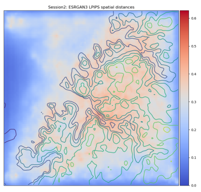
Figure F.1: Spatial LPIPS for all models in Session 1.



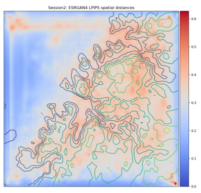
(a) ESRGAN1.



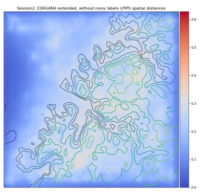
(b) ESRGAN2.



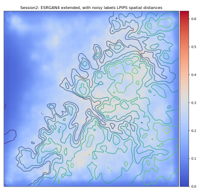
(c) ESRGAN3.



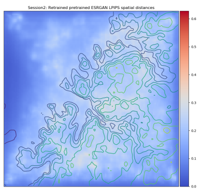
(d) ESRGAN4.



(e) Ext. ESRGAN4 with noise.

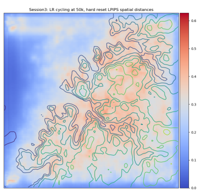


(f) Ext. ESRGAN4.

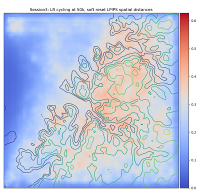


(g) Ext. pretrained.

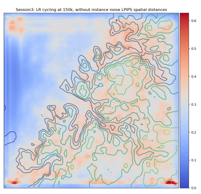
Figure F.2: Spatial LPIPS for all models in Session 2.



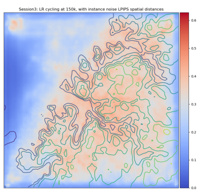
(a) LR cycle 50k, hard reset.



(b) LR cycle 50k, soft reset.

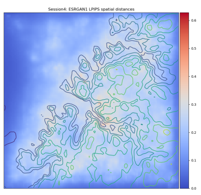


(c) LR cycle 150l w/o instance noise.

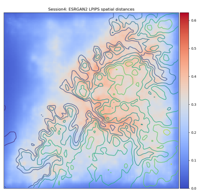


(d) LR cycle 150k.

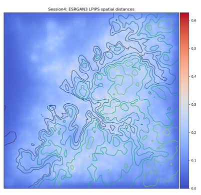
Figure F.3: Spatial LPIPS for all models in Session 3.



(a) ESRGAN1.

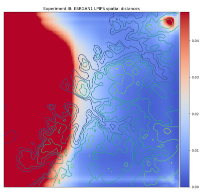


(b) ESRGAN2.

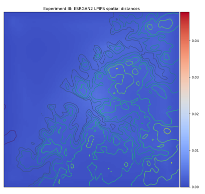


(c) ESRGAN3.

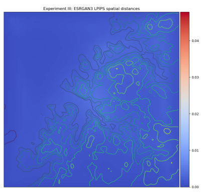
Figure F.4: Spatial LPIPS for all models in Session 4.



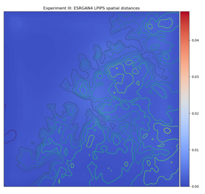
(a) ESRGAN1.



(b) ESRGAN2.



(c) ESRGAN3.



(d) ESRGAN4.

Figure F.5: Spatial LPIPS for all models in Experiment III.