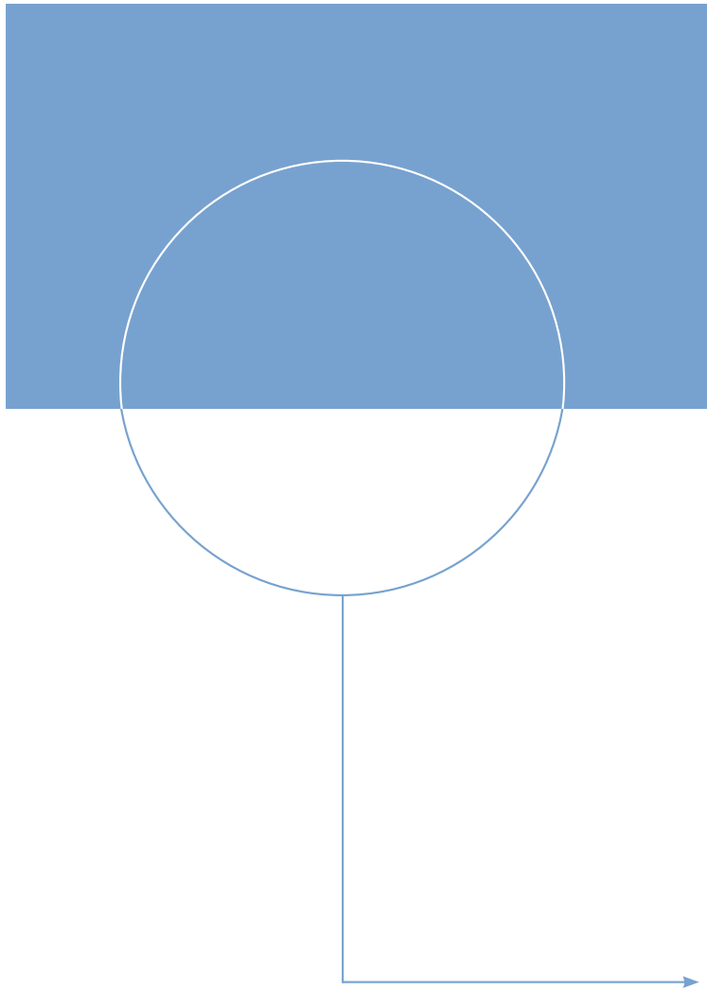


Master's thesis

2020

Duy Tan Huynh Tran

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

Duy Tan Huynh Tran

Convolutional Neural Network and Generative Adversarial Networks Enabled Resolution Enhancement of Numerical Simulations

June 2020

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of
Science and Technology

Convolutional Neural Network and Generative Adversarial Networks Enabled Resolution Enhancement of Numerical Simulations

Duy Tan Huynh Tran

MTTK
Submission date: June 2020
Supervisor: Professor Adil Rasheed

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Abstract

Fluid flows, like atmospheric flows around terrains in wind farms, are governed by a broad variety of spatio-temporal turbulent scales, thus making their real-time numerical modeling computationally unmanageable owing to higher resolution required to capture all the scales. Hence, in this work, we demonstrate a novel approach to address this issue through a combination of fast coarse scale physics-based simulator and a family of advanced machine learning algorithms like convolutional neural networks (CNNs) and generative adversarial networks (GANs). The physics-based simulator generates a coarse wind field in a real wind farm located on a complex terrain and then machine learning models enhance these results to a much finer resolution. The results from machine learning methods are compared with each other and against state-of-the-art interpolation methods with respect to ground truth, which shows the superiority of the approach. We also investigate intermediate results within both deep learning models, gaining insight into how it reconstructs the fully-resolved 3D velocity fields from coarser scale while respecting the local terrain.

Sammendrag

Optimal vindmølleplassering og prognoser av vindmøllers kraftproduksjon krever nøyaktig kunnskap om vindfeltet. Generelt blir målekampanjer foretatt for å innhente informasjon om de rådende vindforholdene i et bestemt område. Disse målekampanjene er ofte kostbare, og gir vinddata med meget grove oppløsninger. Et attraktivt alternativ til målekampanjene er numeriske simuleringer, men de er begrenset av stor regnetid.

Vi presenterer en løsning gjennom en innovativ kombinasjon av tradisjonelle numeriske løsere (numerisk fluiddynamikk kode) og avanserte maskinlæringsalgoritmer som Convolutional Neural Networks (CNNs) og Generative Adversarial Networks (GANs). En tradisjonell numerisk løser basert på bevaringslovene til masse og bevegelsesmengde brukes til å generere et grovt vindfelt, og deretter brukes maskinlæringsmodellene til å forfine oppløsningen. Til slutt presenterer vi eksperimentelle resultater som reflekterer muligheten til å forfine oppløsningen til et vindfelt og rekonstruere det originale vindfeltet ved bruk av maskinlæringsalgoritmene. Det er ikke blitt gjort funn av tidligere presenterte løsninger som rekonstruerer høyoppløste vindfelt i et ekte komplekst terreng.

Preface

This thesis marks the finalization of my Master's degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU), and is written under the supervision of Professor Adil Rasheed, who did an outstanding job of guiding and assisting me along the way - not only in terms of technical guidance, but also by encouraging me to submit a poster to the 17^{th} Deep Sea Offshore Wind R&D Conference, Deep Wind 2020.

Furthermore, I am thankful to PhD student Haakon Robinson at the department of Engineering Cybernetics for the guidance and support at the beginning of the project. I am also grateful to the HPC group of NTNU for providing me the necessary hardware in order to test the model, and I acknowledge the financial support from the Norwegian Research Council and the industrial partners of the OPWIND: Operational Control for Wind Power Plants project (Grant No.: 268044/E20). I'd also like to thank my family and friends for all the kind support throughout these five years.

All of the presented algorithms and experiments were implemented in Python using the open source software library PyTorch. The PyTorch library was especially useful for its data loading utility.

All results were generated using "Idun Cluster", which is a project among the faculties of NTNU and the IT division with the objective of providing a cluster for rapid testing and prototyping of HPC software. At this time, Idun Cluster consists of 68 nodes. The code is mainly run on two 14-core Intel Xeon E5-2650 v4 $(2,2Ghz)$ processor with 128 GB memory, and an NVIDIA Tesla P100 GPU.

Most of the relevant notation and theory on which the project is based on will be presented accordingly, but the reader is expected to be familiar with the fundamental principles of computational fluid dynamics.

Duy Tan Huynh Tran
Trondheim, June 01, 2020

Table of Contents

List of Tables

List of Figures

Abbreviations

CNN	=	Convolutional Neural Network
GAN	=	Generative Adversarial Network
OPWIND	=	Operational Control for Wind Power Plants
CFD	=	Computational Fluid Dynamics
ML	=	Machine Learning
AI	=	Artificial Intelligence
DL	=	Deep Learning
DNN	=	Deep Neural Network
HARMONIE	=	Hirlam Aladin Regional Mesoscale Operational Numerical prediction in Europe
SIMRA	=	Semi Implicit Method for Reynolds Averaged Navier Stokes Equations
NN	=	Nearest Neighbor
BC	=	Bicubic
ANN	=	Artificial Neural Network
ReLU	=	Rectified Linear Unit
SISR	=	Single-Image Super-Resolution
SRCNN	=	Super-Resolution Convolutional Neural Network
SRGAN	=	Super-Resolution Generative Adversarial Network
ESRGAN	=	Enhanced Super-Resolution Generative Adversarial Network
RRDB	=	Residual-in-Residual Dense Block
HR	=	High-Resolution
LR	=	Low-Resolution
SR	=	Super-Resolution
PCA	=	Principal Component Analysis
PSNR	=	Peak signal-to-noise ratio

Chapter 1

Introduction

Optimal wind turbine siting and power production forecasting in wind farms require accurate knowledge of local wind fields. Generally, measurement campaigns are undertaken to obtain an insight into the prevailing wind conditions at a particular site. These campaigns are expensive, and yield very coarse resolution wind data. Numerical simulation is therefore an attractive alternative to the measurement campaigns. However, high-resolution numerical simulation is computationally intractable. In this master thesis, we will address this issue through an innovative combination of traditional numerical solvers (computational fluid dynamics codes) and advanced machine learning algorithms.

1.1 Background and Motivation

In the context of upcoming technologies like digital twin (DT), internet of things (IoT) and autonomous systems, the need for real-time simulation approaches are growing [1]. In these contexts computational fluid dynamics (CFD) simulations are considered some of the most expensive enablers. To complicate things further, the cost of these simulations scale rapidly with increasing geometric complexity and Reynolds numbers. There are strict constraints on the resolution of the computational mesh that can be utilized to resolve the physics of interest.

In wind engineering applications, one is generally interested in predicting terrain induced flow features like flow channeling, mountain waves, rotors and hydraulic jump [2]. This requires that the computational mesh has sufficiently fine resolution to resolve the terrain accurately. This requirement makes real-time predictions computationally intractable with the current computational infrastructure. There is, therefore, a need to resolve this issue. Solutions to this problem range from model simplification to parameterization. Intrusive reduced order models have been proposed for improving the computational efficiency of

such models [3]. Still, these models tend to be unstable for turbulent flows, and their effectiveness has only been demonstrated on toy problems. To address the instability issues with these models, non-intrusive reduced order models have been proposed [4, 5]. Nevertheless, even these approaches have been limited to academic experiments.

Recent breakthroughs in artificial intelligence (AI) and machine learning (ML) open up new possibilities [6, 7, 8]. Deep Neural Networks (DNN) have been used to learn the dynamics of systems involving fluids [9, 10, 11]. Likewise, Reinforcement Learning (RL) has been used to solve control problems related to fluid [12, 13]. Even so, traditional DNNs fail to learn the dynamics. Lately, a new family of machine learning algorithms called Generative Adversarial Networks (GANs) has achieved human level performance in creative tasks like filling missing pixels, converting black and white images into colored images, generating art, and converting one music genre into another, all without the need of explicit programming. One of the achievements of GANs has been in increasing the resolution of images. The concept was also demonstrated, in the context of fluid mechanics, to reconstruct high-resolution turbulence fields using coarse scale fields [14]. The demonstration was once again for flow around cylinders.

GAN architectures were successfully applied to upscale the Particle Image Velocimetry (PIV) measurements, which were limited to low spatial resolution [15]. A need was felt to develop the GAN-based methodology further with different parameters and architectures for more complex flows. Recently, GANs are used to generate new solutions of PDE-governed systems by training on existing datasets. It is shown that turbulent flow realizations generated from GANs are able to capture several statistical constraints of turbulent flows such as Kolmogorov's $-5/3$ law and small scale intermittency of turbulence [16]. Furthermore, to improve the performance and stability of GANs, temporal coherence was applied to GANs to generate super-resolution realizations of turbulent flows [17]. Governing physical laws in the form of stochastic differential equations were encoded into the architecture of GANs [18].

Inspired by dynamical systems, augmenting the discriminator inputs by using residuals and noise were introduced to training data [19]. Physical constraints such as conservation laws and statistical constraints derived from data distribution were embedded into the generator to improve the generalization capability of the GAN-based physical system emulator [20]. Realistic inflow boundary conditions for turbulent channel flow were produced by combining recurrent neural networks (RNN) with GANs. The combination of RNN and GAN architecture was able to generate fully developed time-varying flow for a long time, and was able to maintain spatio-temporal correlations for generated flow close to those of direct numerical simulations (DNS) [21].

Bode *et al.* [22] presented a physics-informed enhanced super-resolution GAN (PIESRGAN) framework for subgrid scale modeling turbulent reactive flows. Their framework included a loss function based on the continuity equation to enforce the physics into the network. They illustrated the effective performance and extrapolation capability of PIESRGAN framework for decaying turbulence and LES of reactive spray in combustion pro-

cess. Lee *et al.* [23] applied GANs for predicting the unsteady shedding of vortices behind a cylinder. They trained their GAN for two different Reynolds numbers and showed the capability of GAN to produce accurate results at interpolatory condition. In addition, they demonstrated the performance of GAN for predicting flow fields with larger time step interval compared to the time step employed for training.

Lee at al. [24] employed conditional GANs (cGANs) for predicting small eddies in a three-dimensional turbulent mixing-layer. The cGANs are different from GANs in a way that it learns the mapping of input features and randomly generated noise to the output. Werhahn *et al.* [25] proposed the Multi-Pass GAN framework for super-resolution of three-dimensional fluid flows. Their method decomposes generative problems on Cartesian field functions into multiple smaller problems that can be learned effectively using two separate GANs. Specifically, first GAN upscales slices parallel to the XY-plane and the second one refines the whole volume along the Z-axis working on slices in the YZ-plane. This approach leads to shorter and more robust training runs.

It is important to note that all the work we discussed above were shown to work for academic problems. In the current work we apply the approach to reconstruct high-resolution wind field in a real complex terrain. It is demonstrated that the model learned flow behaviour in complex terrain dominated by valleys, hills and fjords. The GANs reconstructed field is compared with state-of-the art interpolation techniques, which are generally employed for finding wind field at any particular site from coarse scale wind field, and also a convolutional neural network (CNN). We demonstrate that the GANs outperform both interpolation techniques and CNN, and provide a powerful alternative to achieve the task of generating high-resolution wind field from inaccurate coarse scale wind field without the need of solving complex equations in real time.

1.2 Objective

The main objective of the current thesis is to explore the possibility of replacing computationally expensive high-resolution simulations with a combination of coarse scale simulation and advanced machine learning algorithms like CNNs and GANs.

1.3 Contributions

We propose a novel approach through an innovative combination of physics-based computational fluid dynamics simulator and GANs, that generates high-resolution wind field in complex terrain. A traditional numerical solver based on mass and momentum conservation principles is used to generate a very coarse scale wind field, and then a pre-trained GAN is used to refine the resolution. Finally, our model is evaluated against state-of-the-art upsampling methods and a CNN.

In summary, the main contributions of this thesis include:

- We provide a physics-based simulator that consists of two different models operating at different spatial resolutions and coupled together to make the realistic wind flow modelling computationally manageable. This generates a coarse wind field in a real wind farm.
- We propose a novel combination of fast coarse scale physics-based simulator and GANs to generate high-resolution wind field in complex terrain. The GAN-based artificial intelligence framework learns the main characteristics of the flow in complex terrain.
- We present an extensive quantitative and qualitative evaluation of the generated images and our model's capability to learn the main characteristics of the flow in complex terrain. Further, we demonstrate how it outperforms common state-of-the-art techniques.

1.4 Thesis Structure

In the following chapter, Chapter 2, we give a brief high level understanding of the numerical solver and different interpolation methods. Further, we present in-depth deep learning fundamentals, before we move towards CNN and GANs. In Chapter 3, we present the description of the data, software and hardware framework, and discuss the hyperparameter choices of our model. Chapter 4 presents the results of state-of-the-art methods, our proposed models on the dataset and insights into the inner working of the models. Finally, in Chapter 5, we will conclude and discuss further work.

Chapter 2

Theory

In the first part of this section, a brief overview of the governing equations, numerical codes utilized and their capability is given. Wherever possible, the articles which describe the tools in more detail are referred to. In the second part of this chapter, we give a more in-depth explanation of each interpolation method utilized. In the third part, we first start with a simple deep neural network before we go more in-depth into the evolution of convolutional neural networks (CNNs) and the fundamentals of generative adversarial networks (GANs). Finally, we combine the theory mentioned so far for our purpose.

2.1 Atmospheric Models for Data Generation

Atmospheric flows are governed by mass, momentum and energy conservation principles given by Equations 2.1, 2.2 and 2.3 respectively.

$$\nabla \cdot (\rho_s \mathbf{u}) = 0 \tag{2.1}$$

$$\frac{D\mathbf{u}}{Dt} = -\nabla \left(\frac{p_d}{\rho_s} \right) + \mathbf{g}\frac{\theta_d}{\theta_s} + \frac{1}{\rho_s}\nabla \cdot \mathbf{R} + \mathbf{f} \tag{2.2}$$

$$\frac{D\theta}{Dt} = \nabla \cdot (\gamma_T \nabla \theta) + q \tag{2.3}$$

where $\mathbf{u}, \rho, p, \theta, \mathbf{R}, \mathbf{f}$ represent velocity, density, pressure, potential temperature, stress tensor and sink/source term (e.g. Coriolis force) respectively. Furthermore, $\mathbf{g}$, γ_T and q denote acceleration due to gravity, thermal diffusivity and temperature source term. γ_T can be used to model radiative heating of the atmosphere in a mesoscale modeling context. As for the subscripts, s signify hydrostatic values, while subscript d indicates the deviation from this value. In mathematical terms this equals to $p = p_s + p_d$, $\theta = \theta_s + \theta_d$, $\rho = \rho_s + \rho_d$ where the hydrostatic relation is given by $\partial p_s / \partial z = -g\rho_s$ and $\rho_s = p_s / R\theta (p_o/p_s)^{R_g/C_p}$, where C_p represents the specific heat at constant pressure while R_g being the gas constant.

Again from [26], $\mathbf{R}, P_k, G_\theta$ are given by Equations 2.4, 2.5.

$$R_{ij} = \nu_T \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} k \delta_{ij} \tag{2.4}$$

$$P_k = \nu_T \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \frac{\partial u_i}{\partial x_j}, \quad G_\theta = -\frac{g}{\theta} \frac{\nu_T}{\sigma_T} \frac{\partial \theta}{\partial z} \tag{2.5}$$

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

The turbulent viscosity ν_T given by Equation 2.6 is computed from the turbulent kinetic energy (k) and dissipation (ϵ) given by Equation 2.7, 2.8.

$$\frac{Dk}{Dt} = \nabla \cdot (\nu_T \nabla k) + P_k + G_\theta - \epsilon \tag{2.7}$$

$$\frac{D\epsilon}{Dt} = \nabla \cdot \left(\frac{\nu_T}{\sigma_e} \nabla \epsilon \right) + (C_1 P_k + C_3 G_\theta) \frac{\epsilon}{k} - C_2 \frac{\epsilon^2}{k} \tag{2.8}$$

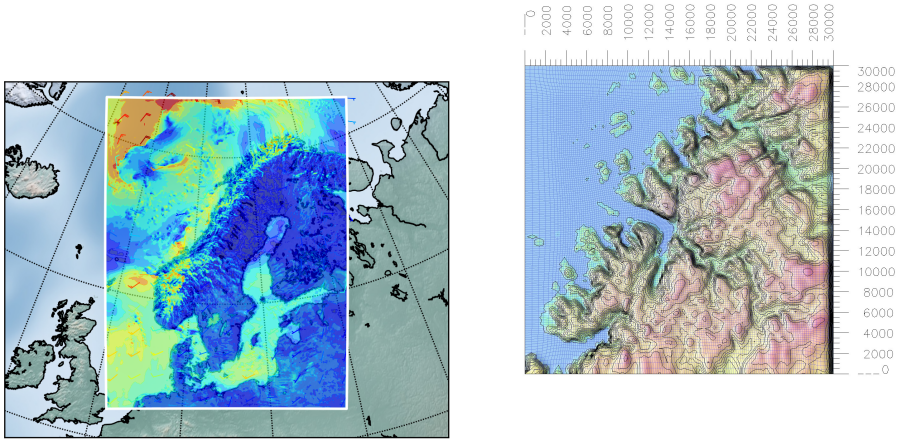
In the current work we have used two different models operating at different spatial resolutions and coupled together to make this realistic wind flow modelling computationally tractable. The large scale model is called HARMONIE and is used as a weather forecast model in Norway. The wind field available from this model is at a horizontal resolution of $2.5\,\text{km} \times 2.5\,\text{km}$. The resolution of the wind field is improved to $200\,\text{m} \times 200\,\text{m}$ using another model called SIMRA. Both these models are essentially based on the equations presented above. One major difference between the two models is in the way turbulence is modelled. In SIMRA a two equation turbulence model (one for turbulent kinetic energy, i.e. Equation 2.7 and another for dissipation i.e. Equation 2.8) is used, while in HARMONIE, a one equation model given by Equation 2.7 is employed. Further, the turbulent dissipation is estimated from $\epsilon = (C_\mu^{1/2} K)^{3/2} / \ell_t$, with ℓ_t computed by applying the relationship

$$\ell_t \approx \frac{\min(\kappa z, 200\,\text{m})}{1 + 5Ri} \tag{2.9}$$

where

$$Ri = \frac{(g/\theta)\partial\theta/\partial z}{(\partial u/\partial z)^2} \approx -\frac{G}{P} \tag{2.10}$$

The stability correction $(1 + 5Ri)$ is replaced by $(1 - 40Ri)^{-1/3}$ in convective conditions and the gradient Richardson number Ri is expected to be less than $1/4$. At last, the coefficients are $(C_\mu, C_1, C_2, C_3) = (0.09, 1.92, 1.43, 1)$ and the coefficients $(\kappa, \sigma_K, \sigma_\epsilon)$ are $(0.4, 1, 1.3)$, respectively [27]. The domain and mesh can be seen in Fig. 2.1b, and at this microscale, the Coriolis effect is neglected.



(a) HARMONIE domain **(b)** SIMRA domain and mesh

Figure 2.1: HARMONIE-SIMRA COUPLING

2.2 Interpolation

With regards to image upscaling of digital images, there are two commonly used scaling algorithms. The first one is the nearest neighbor (NN) interpolation technique, which is the fastest and simplest interpolation algorithm to implement. When upsampling an image, the algorithm chooses the value of the nearest neighboring pixel, and determines the intensity value of it. An example can be seen from Fig. 2.2, where we have an image region of 2×2 green pixels. During the upscaling phase to 3×3, five additional pixels are created, which have no color associated with the original image. When utilizing NN, the algorithm only utilizes the color of the green pixel to assign to the new pixels. This can again lead to a huge problem such as introducing aliasing or jagged edges, and bicubic interpolation is therefore more often preferred.

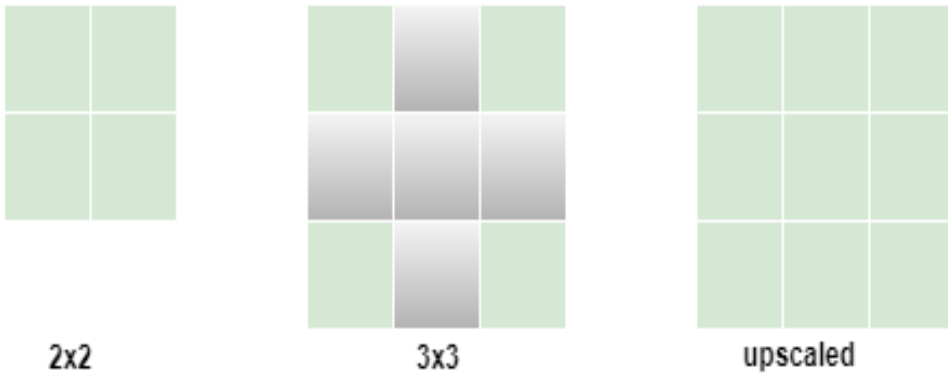


Figure 2.2: Example of nearest neighbor interpolation.

The bicubic interpolation technique interpolates the digital image on a 2D grid of pixels. Compared to bilinear interpolation, which only considers 2×2 pixels, bicubic takes 4×4 pixels into consideration and performs a cubic interpolation on each of the two dimensions of the image. This results in smoother looking images and having less interpolation artifacts. It is a simple algorithm, which adds more pixels in between the ones we already have, and appropriately fills each pixel up based on the colors of the pixels directly surrounding it. A simple example can be seen from Fig. 2.3.

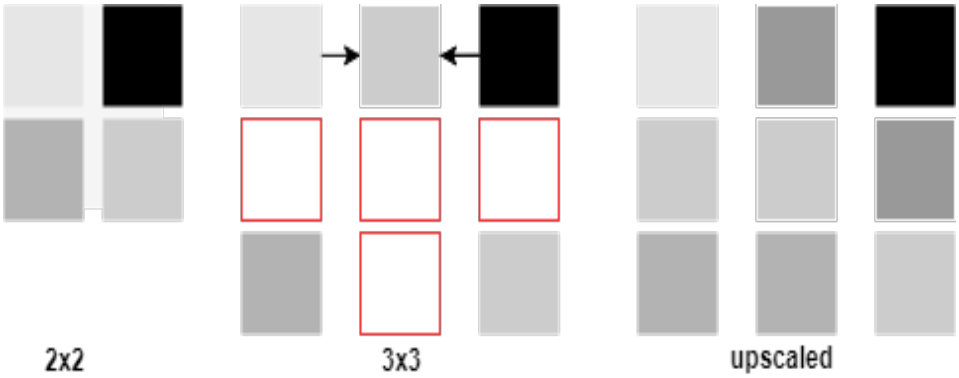


Figure 2.3: Example of bicubic interpolation.

2.3 Neural Networks

Artificial Neuron

The artificial neuron is the fundamental building block of neural networks. It was devised as a computational model of the biological neurons of the brain. The *neurons* form the fundamentals of the network, i.e. an artificial neural network (ANN). Typically, they are modelled as seen in Fig. 2.4. The inputs to the neuron is shown on the left side as the vector x, which are all weighted separately by the vector w and summed up together with the bias term b. Next, the sum is injected into an activation function ϕ to estimate the output y. The activation function is primarily used to saturate the range of the neurons. In mathematical terms, a neuron is simply a multivariable function given as

$$y = \phi\left(\sum_i x_i w_i + b\right) \tag{2.11}$$

Network Architecture

By combining layers of neurons a network is then developed. The first layer of the network represents the inputs, which are then fed throughout the network reaching the output layer as the end destination. The individual neurons in the input layer are related with a feature in the input data, e.g. the intensity value at a particular pixel position, and for every

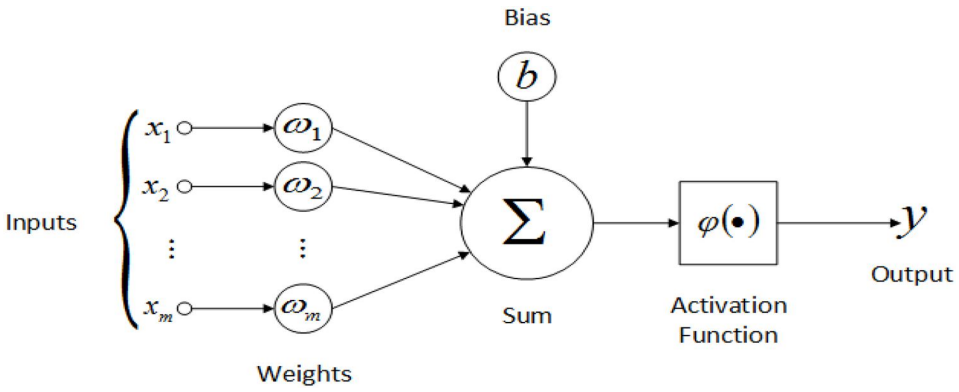


Figure 2.4: The structure of an artificial neuron [28]

sample they are simply allocated values in the data. Between the input and output layers, one can define one or more layers as hidden layers because they are internal to the network and are usually abstracted away. In particular, given the values of the input layer and by applying Eq. 2.11 for each individual neuron, the activations of each subsequent layer can then be calculated. This process of connecting the input signal across the network is called *forward propagation* and enables us to calculate the values of the output neurons in the final output layer. For a specified network with fixed biases and weights, the output values depend solely on the inputs.

Fig. 2.5 shows an example of an ANN composed of an input layer with three neurons, one hidden layer with four neurons and the final output layer with two neurons. The amount of neurons is defined as the size of each layer, i.e. input layer is of size two, hidden layer of size three and final layer of size two respectively. Furthermore, a typical implementation problem of deep learning is the difficulty of deciding the size of each layer. While the size of the input layer is straightforward, it is particularly harder to define the optimal configuration of hidden layers by intuition. The amount and depth of neurons depend on the complexity of the problem, which again will affect the runtime of the network and performance. Prior work with neural networks utilized only a simple hidden layer, and theoretical studies demonstrate that simple, single-layer networks can depict any function with arbitrary precision [30]. Certain novel, high-performing DL architectures such as ResNet-152 take advantage of beyond 100 layers, with approximately 100 million parameters, i.e. neuron weights. Between various network architectures there is a substantial variation in classification accuracy, even for networks with similar computational complexity [31]. A representational ability of a network is not the appropriate limitation, but rather to learn a satisfactory representation during the training phase.

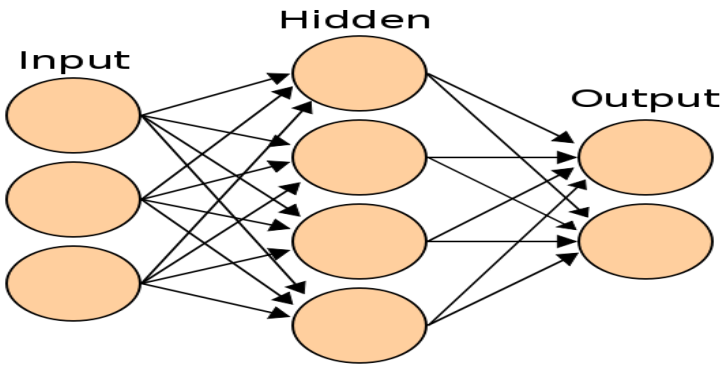


Figure 2.5: An artificial neural network where each node represents a node as depicted in Fig. 2.4 [29]

Gradient Descent and Backpropagation

The training phase is defined as the process whereby the parameters of the network, i.e. the biases and weights of the individual neurons, are modified in order for the network to generate the true output values. The concept of a desired output is defined by means of a given cost function, e.g. the mean squared error [30]:

$$C(w,b) = \frac{1}{2n} \left\| \sum_x y(x) - a(x,w,b) \right\|^2 \tag{2.12}$$

where C is the cost defined as half the average of the square error for each particular sample x. a and y signify the neural network output and the correct output for the given sample respectively. From a quick observation, since the output a is dependent on the set of all neuron biases and weights, i.e. b and w, the cost C is a function of these network parameters, in addition to the values of y and x, which are provided by the dataset and not subject to optimization. Thus, the equivalency of training the neural network is by minimizing the cost function C w.r.t w and b, where the optimal neural network that ideally matches the correct values achieve a minimal cost of zero. Even with a small neural network that depicts a toy problem scenario, this optimization problem is analytically unmanageable in a large dimension space. Instead, the common method is to approximate the minimum by means of gradient descent, which is a straightforward iterative algorithm utilizing first derivatives in the following way [30]:

$$p_i = p_{i-1} - \eta \nabla C \tag{2.13}$$

where p_i is the parameter set (the biases and weights, combined) given by shifting from the previous parameter set, p_{i-1}, in the opposite direction of the gradient of the cost function. The gradient, by composition, is a vector that points in the direction of largest rate of increase for the function, and intuitively for the objectives of minimization, the greatest negative gradient can be found in the opposite direction. Furthermore, η is defined as the

step-size given as a constant. In the context of machine learning, this constant is commonly denoted as the learning rate, since it adjusts what degree each update will change the network parameters while training.

Intuitively, gradient descent deems considerably computationally expensive, since it is an iterative algorithm implying possibly millions of partial derivative calculations for each parameter. However, there exist an effective common approach of gradient descent in DL, a method named *backpropagation*, resulting in computationally acceptable runtimes when training neural networks. This method has also been redeemed several times [32].

By utilizing the previously mentioned forward- and backpropagation, we can now calculate the output of a network for a specific input and a cost function, e.g. Eq. 2.12 aids as an estimate for the distance between the ideal and actual outputs. For whichever neuron in the output layer, its value is given by Eq. 2.11. During forward propagation the values of these inputs of the neuron is calculated in such a way that the partial derivatives w.r.t. the bias and weights corresponding to this particular neuron can be calculated. Additionally, the activation function of the neuron and the cost function require differentiation, and provided that the gradient of the whole training set can be composed as the average of gradients for each individual sample [30].

From this point on, reproducing the way the biases and weights of neurons in every layer enables us to forward propagate an input throughout a network to determine its output by the utilization of the chain rule [32]. Then, the cost can be backpropagated in the opposite direction of the network, beginning at the output layer and traversing backwards. Thus, the gradient of the cost function w.r.t. all biases and weights can be calculated due to backpropagation, and is as computationally effective as calculating the output of the network for a given input. Through this process, the network can start with randomly initialized biases and weights and iteratively search for better values such that the cost decreases and approximate an ideal mapping from inputs to outputs. This stage-by-stage process is named training the network and is where the self-learning takes place. However, there is no guarantee that the parameter p will converge to the optimal value during gradient descent.

Two failure modes common from single-variable calculus are being stuck in a local minimum, resulting in a suboptimal solution, or frequently overshooting the minimum, which slows down the convergence due to dampened oscillations, or exploding oscillations leading to catastrophic divergence. In practice, the first failure mode appears to be of limited practical significance, partly due to sparsity of local minima in large spaces. In addition, neural networks have a tendency to reach convergence towards resembling trained states no matter initial values. A quote from a 2015 review article [32], this is formulated as:

> In practice, poor local minima are rarely a problem with large networks. Regardless of the initial conditions, the system nearly always reaches solutions of very similar quality. Recent theoretical and empirical results strongly suggest that local minima are not a serious issue in general. Instead, the landscape is packed with a combinatorially large number of saddle points where the gra-

> dient is zero, and the surface curves up in most dimensions and curves down in the remainder. The analysis seems to show that saddle points with only a few downward curving directions are present in very large numbers, but almost all of them have very similar values of the objective function. Hence, it does not much matter which of these saddle points the algorithm gets stuck at. (Yann LeCun *et al.*, Nature, 2015)

There have been extensive studies of the second failure mode, involving several challenges that appear in larger dimensions such as intensely different gradient magnitudes in various dimensions and saddle points. Thus, the gradient descent technique in Eq. 2.13 can first be refined to include a momentum term, in such a way that our update technique utilizes the record of previous update steps [30]:

$$m_i = \mu m_{i-1} - \eta \nabla C \tag{2.14}$$

$$p_i = p_{i-1} + m_i \tag{2.15}$$

where μ is a constant that regulates to which degree we preserve the previous values of the update steps. Notice that for $\mu = 0$ this set of equations is identical to Eq. 2.13, i.e. the technique has no memory of its previous update steps, and while μ is approaching 1, the dynamics are controlled by the record of updates. The motivation behind this modified gradient descent is how it enables the updates to build up momentum, i.e. advancing with greater steps, in a dimension space where the gradient continuously points in the corresponding directions, whilst steps ought to be relatively smaller in a dimension where the gradient continues to change direction.

A refined optimization technique was introduced in 2014 denoted Adaptive Moment Estimator (Adam) [33], and is one of the most commonly used implementations of gradient descent for neural networks. It utilizes both first order momentum, as presented above, along with bias corrections and second order momentum. Moreover, it is considered a notably robust algorithm where trial and error is best practice for determining an appropriate learning rate. Even though the more refined optimizers are in general more stable, one must also take into account the extra parameters that may need tuning, e.g. the exponential decay rates of the moments.

2.4 Convolutional Neural Networks

With the increase of hidden layers in deep neural networks, the individual neurons and its connections in the network becomes infeasible to design by hand. Instead, when we express the activation functions in terms of convolutional kernels, a robust class of networks arises, specifically convolutional neural networks (CNNs), with outstanding success in image and pattern recognition [34]. In the later sections we will make use of more advanced networks consisting of *residual blocks*, which utilize layers such as *convolutions*, *rectified linear units* and *shortcuts*. In this section, these concepts will be elaborated in detail within the context of a residual network.

Convolution Layer

From [35] a CNN is defined as:

> Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. (Goodfellow *et al.*, Deep Learning, 2016)

In other words, a CNN is a form of ANN that makes use of at least one convolutional layer in its architecture. It can also be traced all the way back to Fukushima and his Neocognitron [36], where he presented a hierarchical multilayered neural network conducting robust visual pattern recognition. In mathematical terms, a convolution with regards to the neural network can be defined as

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.16}$$

where $(f * g)(t)$ is a completely new function based on $f(t)$ and $g(t)$. It can be noticed as the weighted function of $f(\tau)$ at time instant t where the weighting is given by $g(t-\tau)$. In other terms, the convolution operator defines the output with regard to the input. In relation to neural network terminology we can further write this theory as

$$s(t) = (x * \omega)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)(t-\tau), \tag{2.17}$$

where the output $s(t)$ is often defined as the feature map. Furthermore, the inputs x and ω are often defined as the input and kernel, respectively. Notice that Eq. 2.17 is the discrete version of the continuous convolution operator from Eq. 2.16.

In this work we will work with wind fields depicted as images, and it would therefore be more appropriate to introduce multidimensional arrays. If an input image $\mathbf{I}$ is represented in two dimensions, then the kernel $\mathbf{K}$ should be represented in two dimensions as well. Thus, we modify Eq. 2.17 into a two-dimensional version

$$\mathbf{S}(i,j) = (\mathbf{I} * \mathbf{K})(i,j) = \sum_{m}\sum_{n} \mathbf{I}(m,n)\mathbf{K}(i-m, j-n) \tag{2.18}$$

We can now interpret discrete convolution as matrix multiplication, and for this reason the kernels are often interpreted as small matrices that retrieve the desired data from the input. In a CNN, a known kernel is the edge detection kernel, which is applied to images in order to detect edges. Furthermore, the first level of convolution usually represents the existence or lack of edges at specific locations and orientations in the image [32].

Fig. 2.6 shows the mathematical procedure where an image represented in matrix form is multiplied with an edge detection kernel. A 3×3 kernel will thus decrease a $n \times m$ matrix

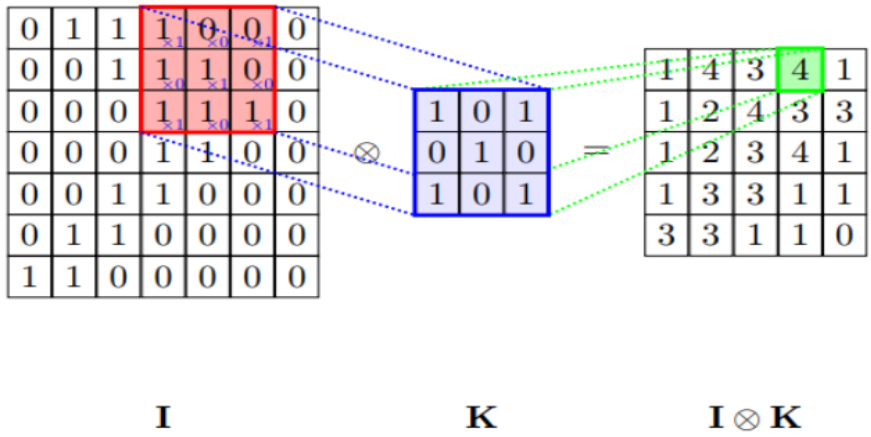


Figure 2.6: Convolution from a matrix point of view [37]

into a dimension of $(n-2) \times (m-2)$, and the resulting feature map depicts the requested information. However, the use of convolution operator introduces a problem when we are close to the edge of the image or if the size of the image is too small. With several convolution layers, this problem can result in an unwanted small feature map. Nonetheless, this problem is tackled by introducing padding, where additional zero-values are added at the start and end of the input matrix such that the size of the resulting feature map will not decrease. This technique can be shown as

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \xrightarrow{padding} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.19}$$

Rectified Linear Units

We introduced in Eq. 2.11 the activation function f that is utilized when we calculate the output from a neuron. LeCun *et al.* [32] denotes the rectified linear unit (ReLU) as the most commonly used activation function in deep learning applications defined as

$$f(z) = \max(z, 0) \tag{2.20}$$

where z is the pre-activation neuron output. From Glorot *et al.* [38], ReLU has been proven to outperform activation functions such as *tanh* and *softplus* in deep learning applications related to image data. There have also been proposed various ReLU modifications, and the one used in this work is the leaky rectified linear unit (LeakyReLU)

$$f(z) = \max(z, -\alpha z) \tag{2.21}$$

where the non-negative constant α is of order 10^{-1}. The reasoning behind the choice of this activation function is to avoid *vanishing gradients*, which happens when the gradient becomes truly small and will thus not update the weights significantly.

Shortcuts

Traditionally, deeper convolutional neural networks notoriously suffer from the degradation problem, i.e. accuracy reduction with increasing depth of the neural network after arriving at a maxima. However, Liu *et al.* [39] achieved undoubtedly a milestone in deep learning with the introduction of Residual Network (ResNet) utilizing shortcut connections. With the introduction of convolutional layers, which take into account deeper networks that despite everything have a sensible number of parameters, the pattern has been for networks to turn out to be even more deeper [31].

In any event, putting aside computational confinements, training networks with many layers is troublesome. Generally, the issue is that the parameters in any specified layer cannot be optimized autonomously for the remainder of the network, and the training strategy fails to work adequately for any specified layer while a large number of the layers in the network chain are poorly tuned. Several neat tricks have been uncovered to tackle this problem, e.g. increasing the depth of networks to learn more complex relationships with the use of unsupervised pre-training [38]. In relation to the residual blocks used in GANs, another approach is to pre-train the network with less layers, until placing extra layers and re-training the deeper, modified network. This kind of incremental bootstrapping method eases a significant signal to propagate throughout the deeper neural network such that the gradient descent optimization function properly.

In our deep learning frameworks, the residual blocks utilize shortcuts, or more elegantly *additive identity mappings*. The input is forwarded through two sequential convolutional layers in a residual block. Even so, rather outputting just this double convolution result, it outputs instead the non-modified input and the sum of this double convolution result [40]. The reason for this is to ensure that a significant signal propagates throughout the network regardless of poorly tuned hyperparameters, resulting in a steadier training strategy even for really deep neural networks. He *et al.* [40] has shown significantly better performances with networks utilizing shortcut connections without computational load or any extra parameters introduced, and with increased depth the results are actually getting better.

2.5 Generative Adversarial Network Fundamentals

Generative adversarial networks (GANs) were first described by Goodwell [41]. A generator network **G** takes the data distribution as input (i.e. coarse wind field) and generates a synthetic example (i.e. fine wind field). A discriminator network **D** then attempts to classify the synthetic example as either real or fake. When the two networks are trained simultaneously both tries to outperform the other resulting in a generator that can generate realistic output which are indistinguishable from the fake ones.

A simple analogy to describe GANs is to represent the generator as a counterfeiter who walks into a store with counterfeited bills, while the discriminator represents a cashier who has the knowledge of the difference between real and counterfeited bills. In the first iteration the counterfeiter, i.e. the generator, brings a drawing of e.g. a 10 dollar bill. The fake 10 dollar bill is definitely rejected by the cashier, i.e. the discriminator, but the counterfeiter learns from this mistake, and in the next iteration the counterfeiter tries monopoly money. As there is an evident difference between monopoly and real money, the cashier rejects the counterfeiter, and the counterfeiter now *learns* that monopoly money is insufficient. Thus, for the next iteration, the counterfeiter will try to create more realistic-looking 10 dollar bills. This back-and-forth process continues until the counterfeiter is able to generate really high-quality bills.

In the most optimal conditions, after adequate epochs of training, the generator network is substantially capable of capturing the real data distribution, while the "smart" discriminator network is incapable of distinguishing the generated images from the ground truth. This whole process can simply be seen as a two-player minimax game, which in mathematical terms can be described with the subsequent value function $V(D, G)$ [41]:

$$\min_G \max_D V(D, G) = \mathbb{E}_{I \sim p_{\text{data}}(I)}[\log D(I)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.22}$$

where I is the real image sample (i.e. fine wind field) from the ground truth, $p_{\text{data}}(I)$ represents the probability distribution of the fine wind field, and $D(I)$ is the probability that I derived from the real images (i.e. fine wind field) instead of the generated images (i.e. realistic-looking wind field). z is the random noise of input generator network G, $G(z)$ is the generated fake image (i.e. realistic-looking wind field), and $D(G(z))$ is the probability of determining whether $G(z)$ derived from the real images or not. Throughout the entire training process, the generator network G desires to generate the value of $D(G(z))$ as big as possible, which again will diminish the value of $V(D, G)$. As for the discriminator network D, it attempts to increase the $D(I)$ and reduce the $D(G(z))$, resulting in an increase of $V(D, G)$. Hence, the value function $V(D, G)$ tries to modify the parameters of G to minimize $[\log(1 - D(G(z)))]$ and modify the parameters of D to maximize $[\log D(I)]$. This capability has been used to perform super-resolution, which increases the resolution of an input image without introducing obvious artifacts.

2.6 SRCNN: Super-Resolution Convolutional Neural Network

In Single-Image Super-Resolution (SISR) the goal is to estimate a high-resolution, super-resolved image I^{SR} from a low-resolution input image I^{LR}. The low-resolution images I^{LR} are obtained by applying a downsampling operation to I^{HR} with a tuneable downsampling factor. SRCNN is the first deep learning method for this purpose, which can directly learn an end-to-end mapping between the low/high-resolution images.

Fig. 2.7 shows the simple network structure layout. It is a simple CNN containing three layers, where each layer consists of a convolution layer with an activation function. The bicubic interpolation of a low-resolution image is the input image of the network, with equivalent size as the output high-resolution image. From the figure, the first layer primarily extracts representations and patches of low-resolution images, with a convolution of 9×9 filter size of 64 number of feature maps and three channels. The second layer maps the n_1- *dimensional* representations, i.e. feature vectors, of various patches into an n_2- dimensional one, resulting in a non-linear mapping. For each mapping operation the number of patches relies on the kernel size of the second convolution layer. In the figure this is seen as a convolution with 5×5 filter size of 32 number of feature maps. Finally, the last layer reconstructs the high-resolution image [42]. We will later show that the results from SRCNN are quite good, but as mentioned earlier, the GANs have proven better results in terms of image quality than CNN.

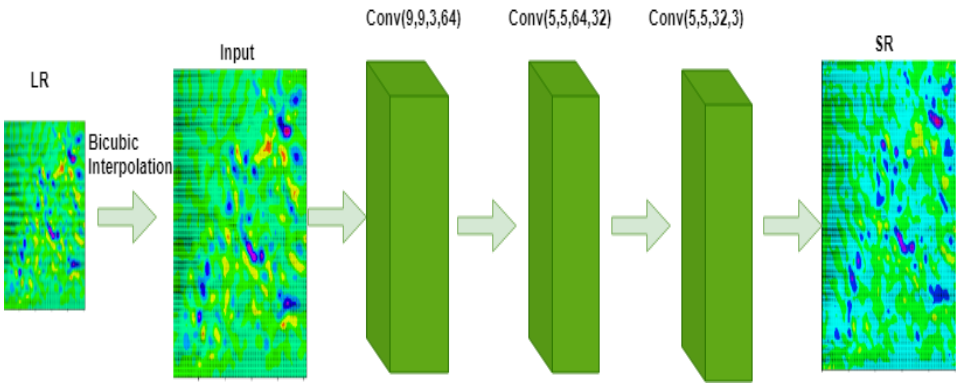


Figure 2.7: Network architecture of SRCNN.

2.7 ESRGAN: Enhanced Super-Resolution Generative Adversarial Network

Ledig *et al.* [43] introduced SRGAN, which uses a *perceptual loss* function based on high-level features extracted from a pre-trained image classification model [44]. The work used the VGG19 network [45], which was trained on over a million examples from the ImageNet dataset [46]. This greatly improved the perceptual quality of the generated images, but was observed to introduce high-frequency artifacts for deeper networks. A simple high-level block diagram of super-resolution utilizing GANs can be shown in Fig. 2.8, with the velocity components *(u, v, w)* of the generated data concatenated and used as input.

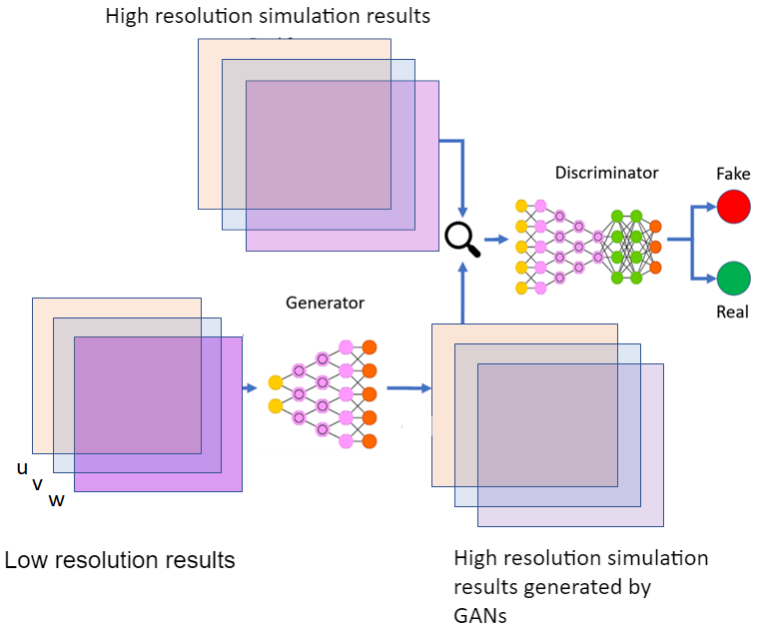


Figure 2.8: High-level block diagram of super-resolution using GANs.

To enhance the output quality of SRGAN, Wang *et al.* [47] modified the network architecture by changing the basic network building block to the Residual-in-Residual Dense Block (RRDB) (see the red block in Fig. 2.9), calling the resulting model the *enhanced SRGAN* (ESRGAN), which is the model that this work is based on. Each RRDB consists of four convolution layers $F = [f_{c1}, f_{c2}, f_{c3}, f_{c4}]$ where the first three layers are concatenated [48] such that the third convolution layer will have $\times 3$ output feature maps, while the last transition layer will squeeze the input feature map to the output channeled feature map. Then, the shortcut connection, which is described in Sec. 2.4, is established between the input and the feature map of the last convolution layer in the RRDB, i.e. $\mathbf{x} + f_{c4}\left(f_{c3}\left(f_{c2}\left(f_{c1}(\mathbf{x})\right)\right)\right)$.

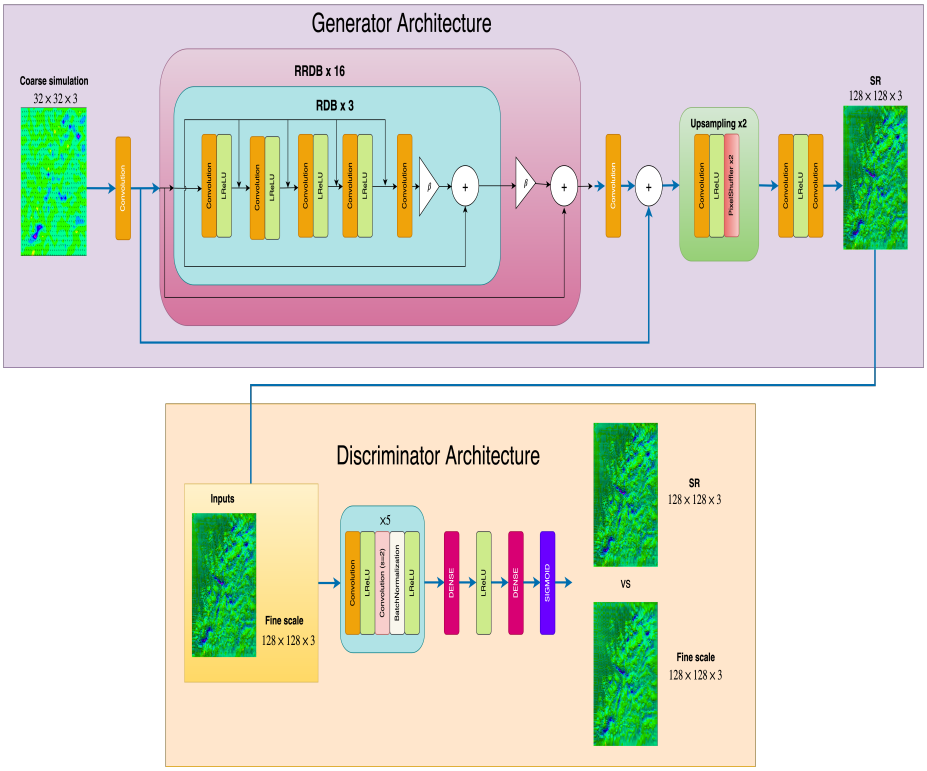


Figure 2.9: Architecture of the ESRGAN model. The generator network consists of two convolutional layers (3x3 kernels, 64 feature maps, and LeakyReLU activation), residual skip connections (scaled by $\beta = 0.2$) and two upsampling layers (two sub-pixel convolutional layers). The discriminator consists of five convolutional layers, two dense layers, and a sigmoid output. The convolutional layers have an increasing number of 3×3 filter kernels (scaling by a factor of 2 from 64 to 512 kernels), and strided convolutions are applied after each one. Zero-padding is used to control the output shape, as is common practice.

Two sub-pixel convolution layers (see the green block in Fig. 2.9) [49] are used to upsample the feature maps by accumulating feature responses at different channels. Furthermore, ESRGAN utilizes the Relativistic Average Discriminator (D_{R}) [50], allowing the generator network to be trained on the *relative realness* of its output, rather than a hard binary classification. This was reported to yield more consistent performance both during and after training, as well as better looking images containing detailed textures and sharper edges compared to previous work [47]. The relative realness of a synthetic image relative to the original can be formulated as:

$$D_{\mathrm{R}}(x_r, x_f) = \sigma(C(x_r) - \mathbb{E}_{x_f}\left[C(x_f)\right]) \tag{2.23}$$

where x_r, x_f are real and synthetic examples respectively, σ the sigmoid function, $C(x)$ is the non-transformed discriminator output, and $\mathbb{E}_{x_f}[\cdot]$ represents the average over all the

synthetic images in the current mini-batch. Based on this, the adversarial losses for the generator and discriminator networks ($L_{\mathrm{G}}^{\mathrm{R}}$ and $L_{\mathrm{D}}^{\mathrm{R}}$, respectively) are defined as:

$$L_{\mathrm{G}}^{\mathrm{R}} = -\mathbb{E}_{x_r}\left[\ln(1 - D_{\mathrm{R}}(x_r, x_f))\right] - \mathbb{E}_{x_f}\left[\ln(D_{\mathrm{R}}(x_f, x_r))\right] \tag{2.24}$$

$$L_{\mathrm{D}}^{\mathrm{R}} = -\mathbb{E}_{x_r}\left[\ln(D_{\mathrm{R}}(x_r, x_f))\right] - \mathbb{E}_{x_f}\left[\ln(1 - D_{\mathrm{R}}(x_f, x_r))\right] \tag{2.25}$$

where $x_f = G(x_i)$ and x_i as the input low-resolution image. The total loss for the generator and discriminator networks are then:

$$L_{\mathrm{G}} = L_{\mathrm{percep}} + \lambda L_{\mathrm{G}}^{\mathrm{R}} + \eta L_1 \tag{2.26}$$

$$L_{\mathrm{D}} = L_{\mathrm{D}}^{\mathrm{R}} \tag{2.27}$$

where L_{percep} is the perceptual loss term from SRGAN [43], $L_1 = \mathbb{E}_{x_i}\left\|G(x_i) - y\right\|_1$ is the 1-norm distance between the ground truth image y and the generated image x_i, and the coefficients (λ, η) are separate learning rates for the adversarial and L_1 losses, and may be varied during the training process. In their original work on ESRGAN, Wang *et al.* [47] initialised these rates as $\lambda = 5 \cdot 10^{-3}$ and $\eta = 10^{-2}$, and further reduced λ by a factor of two every 50k iterations.

2.8 Principal Component Analysis

Principal component analysis (PCA) is a method for pattern identification in data, and expressing the data in such a way as to highlight their similarities and differences [51]. It is most commonly used as a dimensionality reduction method [52]. Essentially, the idea is to depict a dataset using fewer variables than the original dataset, while keeping as much information as possible. PCA is a straightforward five step procedure:

1. Get a dataset
2. Subtract the mean from the dataset
3. Calculate the covariance matrix
4. Calculate the eigenvectors and eigenvalues of the covariance matrix
5. Choose components and form a feature vector

In Sec. 4.2 PCA is applied to a set of images. There are numerous ways to do this. A precise description of how this is accomplished will be presented here. We have two objectives we wish to achieve:

1. Without presenting every single one of the images, we want to visualize as much of the information as possible
2. To make the images necessary for the neural network, we want to determine if they contain truly distinct data

An image is therefore treated as a variable and the image height and width as samples. Images are obviously two dimensional and thereby can not exactly be considered as samples. Hence, this data is unpacked into a single dimension. This is done by fetching one row at a time from an image and attaching it to the next row. This is demonstrated in Eq. 2.28

$$\begin{aligned} I &= \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \\ &= \begin{bmatrix} c_{00} & c_{01} & c_{10} & c_{11} \end{bmatrix} \end{aligned} \tag{2.28}$$

If we had 128 images of size 32×32 we gather this together such that we have a matrix of dimensions $128 \times (32 \cdot 32) = 128 \times 1024$, and apply PCA onto this matrix. As a result, we end up with a list of components that include the variance in the original dataset. There are two main results worth noticing. Firstly, when one component describes all the variance in the dataset. In such a case the images hold a clear pattern, e.g. the images are all the same. Secondly, when all components describe corresponding levels of variance, indicating no clear pattern in the images, e.g. the images are all different.

Chapter 3

Set-up

In this section we present our computational set-up. First, we will describe the data generation and pre-processing. Next, we will outline the software and hardware frameworks used in this work. Then, we will discuss our choice of hyperparameters for both neural network frameworks and quantitative evaluation metrics. Finally, we outline the complete training workflow with respect to the GAN-based artificial intelligence framework.

3.1 Data Generation

The HARMONIE-SIMRA coupled system is utilized to generate the data used in this work. As mentioned earlier the $2.5\,\mathrm{km} \times 2.5\,\mathrm{km}$ horizontal resolution wind forecast data from the HARMONIE model was used to force the SIMRA model which in turn generated a wind field at a fine horizontal resolution of $200\,\mathrm{m} \times 200\,\mathrm{m}$ over a domain of $30\,\mathrm{km} \times 30\,\mathrm{km} \times 3\,\mathrm{km}$. The model is operational since 1^{st} July 2017, generating an hourly stream of three dimensional wind field, pressure, turbulent kinetic energy and dissipation rate. For the current work the data corresponding to the period 1^{st} July 2017 to 1^{st} July 2019 was utilized. The duration corresponded to $2 \times 365 \times 24 = 17520$ data points.

3.2 Data Pre-processing

Due to the enormous amount of data and limits of the available computational resource, we demonstrate our approach in a two dimensional setting only. Two dimensional terrain-following planes lying $40\,\mathrm{m}$ above the terrain surface were extracted and treated as the high-resolution data ($200\,\mathrm{m} \times 200\,\mathrm{m}$) representing the ground truth. The downsampled coarse scale data ($800\,\mathrm{m} \times 800\,\mathrm{m}$) was obtained using the nearest neighbor algorithm. The downsampled data was used as the input to the generator in the GAN and the original fine scale data was treated as the corresponding target. Furthermore, each velocity component

was normalized using the respective min-max value of the components according to

$$z_i = \frac{x_i - min(x_i)}{max(x_i) - min(x_i)} \tag{3.1}$$

This step scales the values in the range of $[0, 1]$. The normalization is important in order to make the training less sensitive to the feature scales, leading again to stable convergence. In addition, several multiplication operations occur as the input passes through the layers of the neural network, thus keeping the inputs between 0 and 1 averts these values from getting too huge during the training. This problem is also known as the exploding gradient problem. With regards to our dataset, the velocity components (u, v, w) can be seen as three separate input channels, just like an image has the three color channels RGB respectively. A simple illustration of velocity components *u* and *v* in the respective grid is given in Fig 3.1. The details of the robustness of the operational model are explained further in [53].

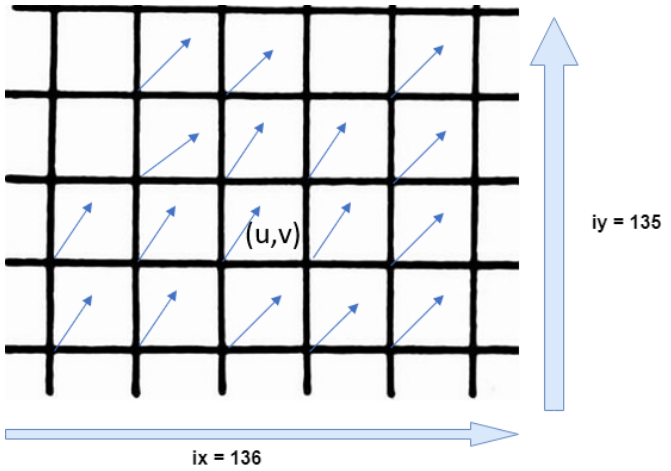


Figure 3.1: Illustration of velocity components u and v on the grid space.

The dataset was further split into training, validation and test set in the ratio $80 : 10 : 10$. This translated into 14016, 1752, 1752 data points for training, validation and test, respectively. The training set was used to train both the CNN and GAN models or in other words, to find the optimal values of the model parameters (also called the weights). The performance of the model was continuously evaluated during the training phase on the validation set. This helped in tuning the model by adjusting the hyperparameters and avoiding overfitting. Finally, the accuracy and performance of the model was tested on the unseen test set.

3.2.1 Downsampling and Resolution Enhancement Algorithms

Two interpolation techniques have been employed in this work: nearest neighbor and bicubic interpolation. While the former is used for creating a coarse scale wind field from the high-resolution field as input to the generator network in the GAN-based framework, the latter is used for simply enhancing the resolution of the coarse scale (generated using nearest neighbor algorithm) for comparison with the GANs reconstructed high-resolution wind field. SRCNN also takes a bicubic interpolated enlarged image as input and learns the mapping relationship between the bicubic interpolated image and the original HR image. Furthermore, the reason for choosing NN during downsampling is to create the most foggy wind field as possible, in order to further strengthening the ability of the generator network of recreating the high-resolution wind field. Fig. 3.2 depicts a sample image.

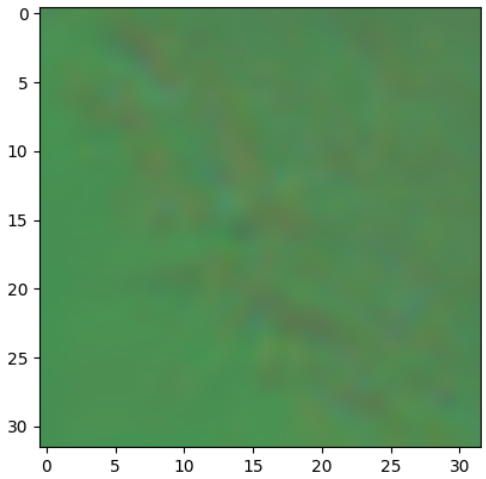


Figure 3.2: Sample image of the input of the network.

For a new point in the coarse mesh, the nearest neighbor algorithm selects the value of the point (from the high-resolution mesh) nearest to it and does not consider the values of other neighboring points at all, yielding a piecewise constant interpolant. Thus this method is very rapid, and creates low quality blocky results.

In the current work we have compared the CNN and GANs generated high-resolution wind field with that obtained using a bicubic interpolation. Even though the resolution increases, a big disadvantage is artificial smoothing of the field, due to the filtering process being based on low-pass characteristics.

Table 3.1: Details of the computational models, number of CPU, domain extent [km], number of mesh elements [million] and total simulation time [minutes].

Model	CORES	Domain	N	Time
HARMONIE	1840	$1875 \times 2400 \times 26$	46	87
SIMRA	48	$30 \times 30 \times 2.5$	1.6	13

3.3 Software and Hardware Framework

All the data employed in this work was available in a NetCDF (Network Common Data Form) file format through an OpenDap server. NetCDF library was utilized for processing the data. All the code for the CNN- and GAN-based frameworks is developed in Python 3.7.2 using the PyTorch 1.2.0 library [54], which is an open source software library developed by the AI group of Facebook with main focus on the implementation of various neural network architectures.

The HARMONIE-SIMRA codes were carried out on the supercomputing facility "Vilje", which is an SGI Altix ICE X distributed memory system that contains 1440 nodes interconnected with a high-bandwidth low-latency switch network (FDR Infiniband). Each node has two 8-core Intel Sandy Bridge ($2.6Ghz$) and $32GB$ memory, providing the total number of cores to 23040. The system is suitable and designed for large scale parallel MPI (Message Passing Interface) applications. The results are transformed into NetCDF [55] and realized through an OPeNDAP server. The utilization of OPeNDAP (Open-source Project for a Network Data Access Protocol) [56] excludes the redundant copying of the result files on several machines for post-processing. A set of Python routines are implemented to read and post-process the hosted files on the fly.

Table 3.1 presents a brief overview of the computational set-up. The HARMONIE model operates on 1840 cores and to perform a 48 hours forecast requires a duration of approximately 87 minutes. SIMRA on the other hand, operating on 48 cores, requires a duration of 13 minutes to complete one hourly averaged simulation each for the next 12 hours. As for the neural network models utilized in this work, the code was run on "Idun Cluster" [57], which is a project among the faculties of NTNU and the IT division with the objective of providing a cluster for rapid testing and prototyping of HPC software. At this time, Idun Cluster consists of 68 nodes. The code is mainly run on two 14-core Intel Xeon E5-2650 v4 $(2, 2Ghz)$ processor with 128 GB memory, and an NVIDIA Tesla P100 GPU.

3.4 Choice of Hyperparameters for SRCNN

SRCNN [58] architectures are CNN based learning algorithms that learn an end-to-end mapping between the low/high-resolution images for enabling single-image super-resolution, and here the mapping is represented by a deep convolutional neural network [59]. The in-built CNN layers are able to successfully capture the spatial and temporal dependencies in an image through the application of relevant filters (such as convolution). The parameters of SRCNN used in this work are shown in Table. 3.2, which are optimized to obtain the best performance of SRCNN. Due to gradient vanishing, the performance of SRCNN cannot be improved by increasing the number of network layers.

Parameter	**Value**
Input	Bicubic interpolation of LR images
Number of layers	3
Parameters of 1^{st} layer	$9 \times 9 \times 1 \times 64$
Parameters of 2^{nd} layer	$5 \times 5 \times 64 \times 32$
Parameters of 3^{rd} layer	$5 \times 5 \times 32 \times 1$
Learning rate	1×10^{-4}

Table 3.2: Table of SRCNN hyperparameters.

3.5 Choice of Hyperparameters for ESRGAN

Due to correlation in terms of tensor operations, our model is based on ESRGAN, thus developing a strong candidate for reconstruction of coarser scale. Table 3.3 yields the most important hyperparameters used in this work. The first hyperparameter "scale" is the factor by which we want to enhance the resolution. We also tried to adjust the depth of the network architecture, i.e. the number of RRDB. Even though the original ESRGAN had great results with 23 RRDB, we experienced far better results with smaller depth and a wider network, i.e. increasing the number of filters (features). The sharpness of the generated images was visually more pleasing, but the number of parameters increased immensely. A useful tool applied was the local feature fusion with kernel size of 1 at the end of the residual dense block, which resulted in almost 50% reduction of the number of weights, with no loss in performance. Hence, the training phase was much faster. One should also note that a too wide network will cause a GPU memory explosion.

After some experimentation, $150k$ iterations was observed to be sufficient for convergence, which is less than what was reported for ESRGAN [47]. To avoid excessive hyperparameter tuning, the learning rates (λ, η) were also chosen to be the same as for ESRGAN, and the decay intervals were reduced proportionately to the reduction in training time. Finally, we should mention that in typical ML areas, the grid search algorithm is commonly used for hyperparameter tuning. However, this process takes a huge toll on GANs since the objective function has additional added costs and thereby resulting in extra hyperparameters to tune.

Parameter	Value
Scale	4
Base no. of Features (G)	128
Base no. of Features (D)	128
Kernel size (G)	5×5
Local Feature fusion (G)	1×1
No. of iterations	150k
L_1 learning rate η	10^{-2}
Initial λ	$5 \cdot 10^{-3}$
λ decay	$\lambda \leftarrow 0.5\lambda$ at it. [10k, 20k, 30k, 40k]

Table 3.3: Table of ESRGAN hyperparameters.

3.5.1 Training Tricks

During training we experienced *mode collapse*, which is when the discriminator loss progressively decreases to zero. This non-convergence occurs when the generator maps multiple inputs to the same output. Thus, the outputs of the generator share many similar features and the generator will accordingly learn to generate just one type of examples rather than to generate all types. This happens due to the hard labels, i.e. Generated Images = 0 and Real Images = 1, causing the discriminator loss to approach zero rapidly. Despite the fact that there is no suitable theoretical foundation as to how to design and train GANs, there is a convincingly proven literature of heuristics, i.e. ”hacks”, that have empirically shown satisfactory results in practice [60]. We utilized *one-sided label smoothing*, which is the idea of replacing the hard label of real images with a value slightly less than 1, in this case 0.9, and thereby prevents the excessive extrapolation behaviour in the discriminator. Another hack utilized was training the discriminator twice as much as the generator to circumvent the mode collapse, and flip the labels the other way around to assist the gradient flow in the early iterations.

3.6 Quantitative Evaluation Metrics

The peak signal-to-noise ratio (PSNR) is usually used to measure the reconstruction quality of lossy transformation (e.g. image inpainting, image compression). Higher is better when measuring with PSNR. In the current context it is defined via the mean squared error (MSE) computed from the original wind field and its noisy (interpolation methods, CNN or GAN) approximations. Mathematically:

$$PSNR = 20 \cdot log_{10}(MAX_1) - 10 \cdot log_{10}(MSE) \tag{3.2}$$

where MAX_1 is the maximum possible wind magnitude. The PSNR is however, only related to the pixel-level MSE between images, only considering about the difference between the pixel values at the same positions instead of how realistic the image looks, i.e. human visual perception. This draws to the poor performance of PSNR in describing the quality of the super-resolved images in real life. Nonetheless, due to the deficiency of completely accurate perceptual metrics and the need to compare performance with literature works, PSNR is currently the most commonly used evaluation criteria for these kinds of problems.

The L2-norm loss function is commonly known as the least squares error (LSE), and is a strict measure of difference. The method simply minimizes the sum of the square of the differences (S) between the target value y_i and the estimated values $f(x_i)$:

$$S = \sum_{i=1}^{n} \left(y_i - f\left(x_i\right)\right)^2 \tag{3.3}$$

3.7 Overview of the Method

The training workflow of the super-resolution reconstruction of turbulent velocity fields using the GAN-based approach is shown in Fig. 3.3. The dataset used in this work was generated using a unidirectionally coupled HARMONIE-SIMRA multiscale system. HARMONIE is a meteorological program used for weather forecasting in Norway, operating at a horizontal resolution of $2.5km \times 2.5km$. SIMRA is a program specially designed to model terrain-induced wind and turbulence in complex terrain at high horizontal spatial resolution of $200m \times 200m$. Both these programs are based on the mass, momentum and energy conservation principles of fluid mechanics. Furthermore, the dataset, i.e. the three channel turbulent velocity fields, can be seen as these three colorful blocks. Next, the input to the generator is created by downsampling the high-resolution data using nearest neighbor by a factor of 4, creating coarser scale wind field and can be represented as these transparent blocks. The output from the generator is a realistic synthetic example of the training set, i.e. the fine scale wind field, and can be seen as these mid-transparent blocks at the bottom left corner. The discriminator then evaluates if the output generated by the generator appears realistic or fake by calculating the probabilities of the input images and returning the training loss to adjust the biases and weights of G and D. Both G and D networks are differential networks, and by using backpropagation algorithms as described in Sec. 2.3, the error gradients can be obtained. These two competing networks are trained simultaneously, each of them trying to outperform the other.

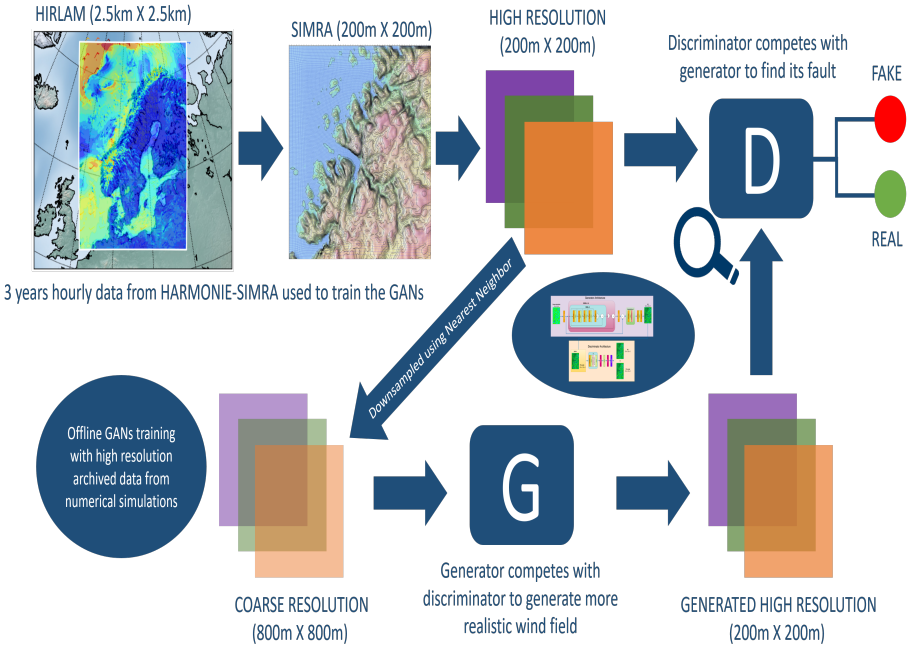


Figure 3.3: Training workflow for GANs for enhancing wind field estimations in complex terrain.

Chapter 4

Results and Discussions

In this section we will outline the results we generated. First, we will present and discuss the results of various upscaling methods on new unseen data. Then, we will go more in-depth into the training results of ESRGAN and provide a detailed comparison against the other upscaling methods. Finally, we will investigate the intermediate layers of both SRCNN and ESRGAN models and apply PCA to gain a statistical understanding.

4.1 Upscaling Methods

The goal of this work is to reconstruct high-resolution wind field in complex terrain without conducting computationally expensive high-resolution numerical simulations. We will first present the results from common interpolation methods and discuss the findings. Then we shift our focus towards deep learning, more specifically the SRCNN and ESRGAN frameworks, and evaluate the proposed models both qualitatively and quantitatively.

4.1.1 Nearest Neighbor Interpolation

As mentioned in Sec. 2.2, nearest neighbor interpolation is very fast and the simplest one to implement. Due to its simplicity and naive solution through duplicating the nearest neighboring pixel, we can clearly see from Fig. 4.1 that NN is nowhere close to the ground truth in both details and sharpness. Instead, NN produces high aliasing effect that results in jagged edges and unpleasant artifacts. Artifacts are most frequently from aliasing, which is often caused by non-linear mixing effects generating high-frequency components prior to sampling. As a result, the reproduced wind field is unnatural and blurry. Even though the method is very fast, it will nonetheless produce blocky results of low quality. We therefore turn our focus towards bicubic interpolation.

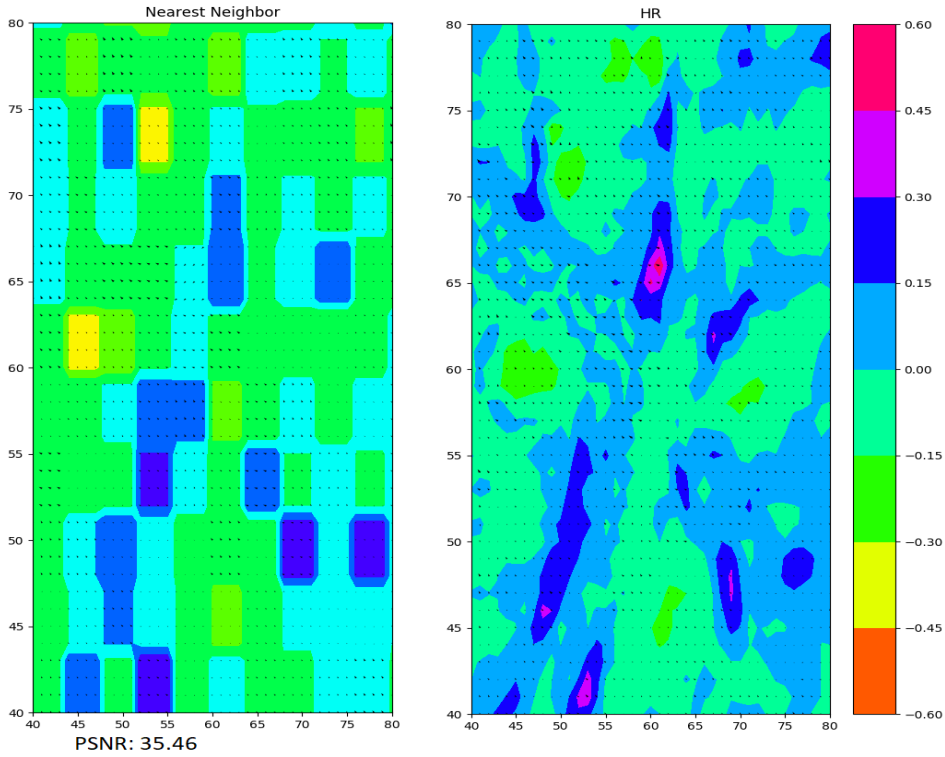


Figure 4.1: Zoomed in (40×40) $\times 4$ enhancement qualitative results between nearest neighbor interpolation and high-resolution fields.

4.1.2 Bicubic Interpolation

Bicubic interpolation improves the resolution exclusively based on its own contents, thus information lost from the downsampling phase is never recovered. This is evident from Fig. 4.2, and it can be seen how bicubic produces blurring, a moderate aliasing and an edge halo effect. It does simply not bring any more information to the table. Instead, it introduces some side effects such as noise amplification, computational complexity and blurring results. In addition, it cannot recover the HR velocity fields consisting of high-frequency information as it is basically a smoothing method. Visually, bicubic interpolation exceeds nearest neighbor in terms of preserving smooth contours in the wind field, and is somewhat closer to the ground truth.

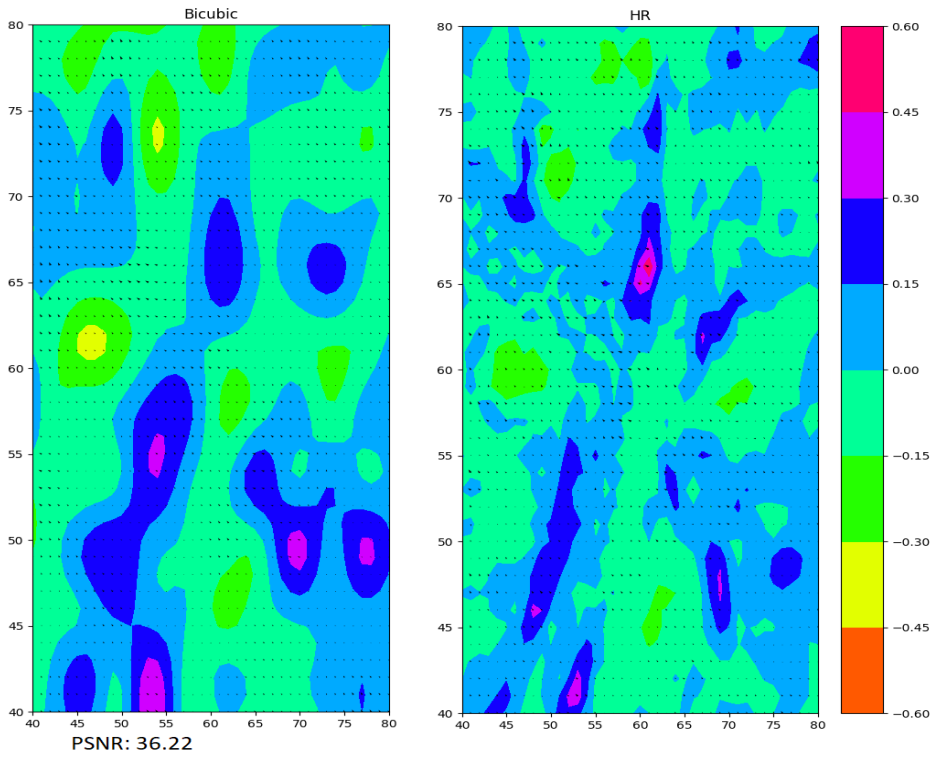


Figure 4.2: Zoomed in (40×40) $\times 4$ enhancement qualitative results between bicubic interpolation and high-resolution fields.

For a quantitative evaluation, the peak signal-to-noise ratio (PSNR), i.e. Eq. 3.2, is also presented in the figure. As expected, bicubic interpolation results in a higher PSNR value of 36.22 compared to NN which resulted in 35.46. Fig. 4.3 presents the L2-norm, i.e. Eq. 3.3, computed from the high-resolution wind field and the reconstructed fields using nearest neighbor and bicubic interpolation for 1800 high-resolution images. One can see that the nearest neighbor interpolation results in significantly greater errors which is in agreement with the observation from Fig. 4.1. Nonetheless, bicubic interpolation is still not a suitable technique for generating finer scale wind field from coarser scale due to the filtering process being based on low-pass characteristics. We therefore shift our focus now to neural networks.

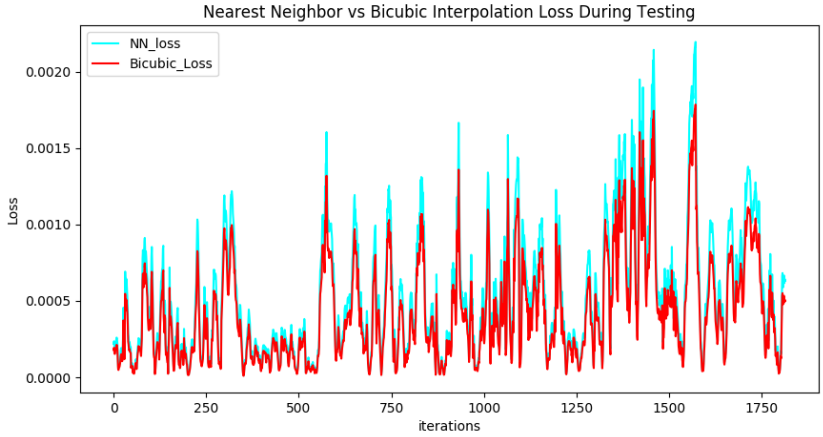


Figure 4.3: L2-norm error comparison of nearest neighbor (NN) and bicubic interpolation over part of the test set. The samples were taken from the September-October 2019 period. Each iteration corresponds to one hour.

4.1.3 SRCNN

Fig. 4.4 gives a comparison of the SRCNN approach, nearest neighbor and bicubic interpolation with respect to the ground truth. In the figure, the wind field in the first image corresponds to the high-resolution wind field obtained by applying nearest neighbor. Second image corresponds to the high-resolution wind field obtained by applying bicubic interpolation on the subsampled wind field. The third image corresponds to the high-resolution field obtained from the application of the trained SRCNN on the coarse field. At first glance, both the bicubic and SRCNN model yield good visual aspects while nearest neighbor is outperformed as expected. However, the difference is more evident from the PSNR given in the respective figure, with a whole $3.5dB$ difference between SRCNN and bicubic interpolation.

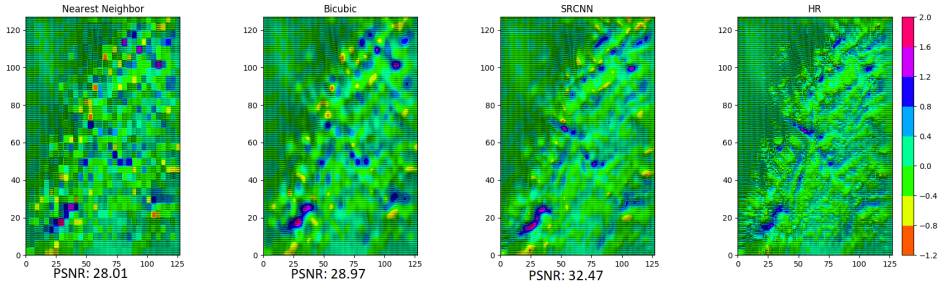


Figure 4.4: Comparisons of the $\times 4$ (from left to right) nearest neighbor, bicubic interpolation, SRCNN and high-resolution fields.

If we further zoom into the bicubic interpolated, SRCNN and HR images as shown in Fig. 4.5, we can observe that the SRCNN model is somewhat closer to the ground truth in both sharpness and details than bicubic interpolation. One of the reasons for the success of SRCNN is due to the ability of the convolutional neural network to *learn*, thus reconstructing more detailed textures than the two other interpolation methods. Bicubic interpolation tend to produce wind fields closer to ground truth, but introduce blurring and noise amplifying in the process. On the other hand, the SRCNN framework bring better perceptual quality resulting in a more realistic wind field, but also introduce unpleasant artifacts such as meaningless noise.

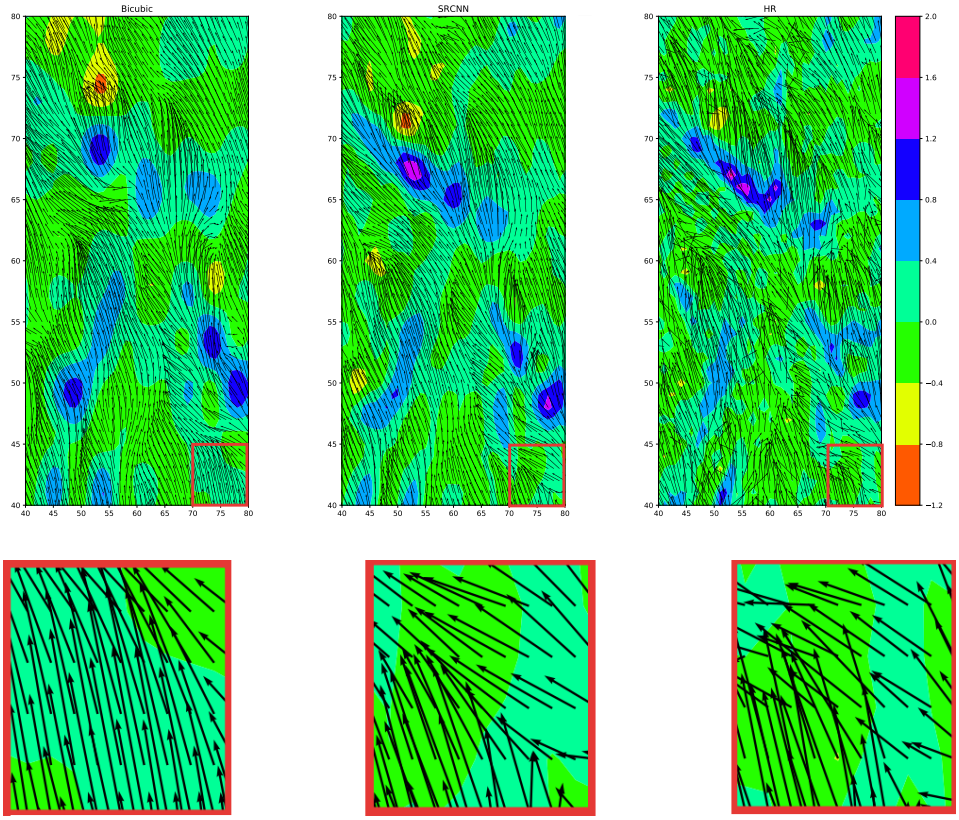


Figure 4.5: Zoomed in (10×10) and $\times 4$ upscaling qualitative results (from left to right) of the bicubic interpolation, SRCNN and high-resolution fields.

Fig. 4.6 presents the L2-norm computed from the high-resolution wind fields and the reconstructed fields using nearest neighbor, bicubic interpolation and SRCNN with respect to the 1800 high-resolution images. We can immediately see that the nearest neighbor method results in significantly greater error than the two other methods, while on the

other hand, SRCNN has consistently smaller error than bicubic interpolation. This is in agreement with the observation from Fig. 4.5. SRCNN has learned to significantly increase the sharpness of the coarser scale wind field and to remove some of the aliasing effects generated from bicubic interpolation. However, it fails to fully reconstruct realistic wind field, leading to a perceptually implausible result.

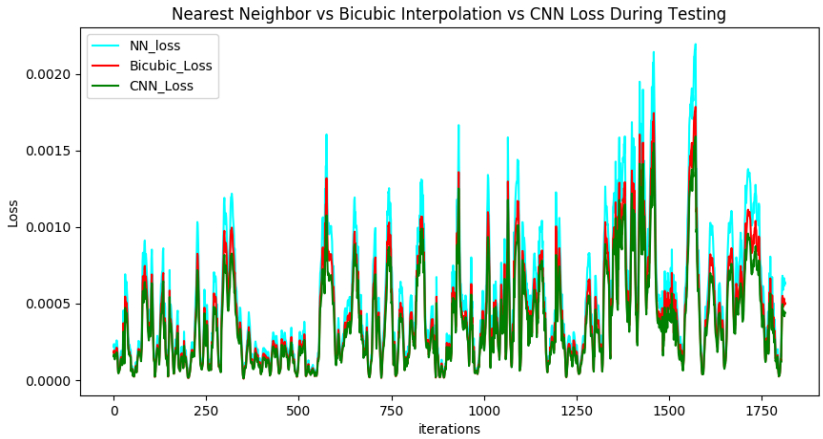


Figure 4.6: L2-norm error comparison of nearest neighbor (NN), bicubic interpolation and SRCNN over part of the test set. The samples were taken from the September-October 2019 period. Each iteration corresponds to one hour.

4.1.4 ESRGAN

This section presents the performance of the trained network on new unseen data set. Fig. 4.7 gives a comparison of the bicubic interpolation, SRCNN and ESRGAN approaches with respect to the ground truth. In the figure, the wind field in the first image corresponds to the high-resolution wind field obtained by applying bicubic interpolation on the subsampled wind field. Second image corresponds to the high-resolution wind field obtained by applying SRCNN on the subsampled wind field. The third image corresponds to the high-resolution field obtained from the application of the trained ESRGAN on the coarse field. It is very clear that the ESRGAN model is able to successfully reconstruct the high-resolution field and is closest to the original high-resolution wind field in both details and sharpness. Indeed, if we further zoom in as shown in Fig. 4.8, we can clearly see the qualitatively better field predicted by the ESRGAN. It is shown how SRCNN over-smooths the wind field, while on the other hand, the framework of ESRGAN successfully *learns* the main characteristics of the flow in complex terrain. Moreover, unpleasant artifacts are nonexistent leading to more natural results, which have been a common problem in typical GANs due to the use of batch normalization layers such as the SRGAN.

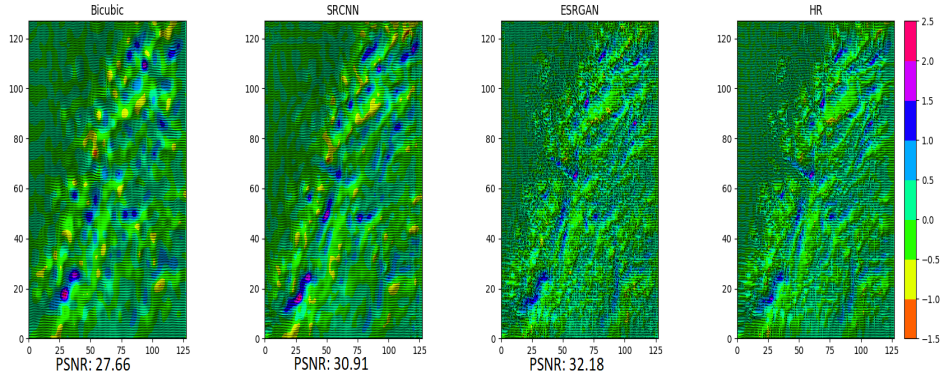


Figure 4.7: Comparisons of the $\times 4$ (from left to right) bicubic interpolation, SRCNN, ESRGAN and high-resolution fields.

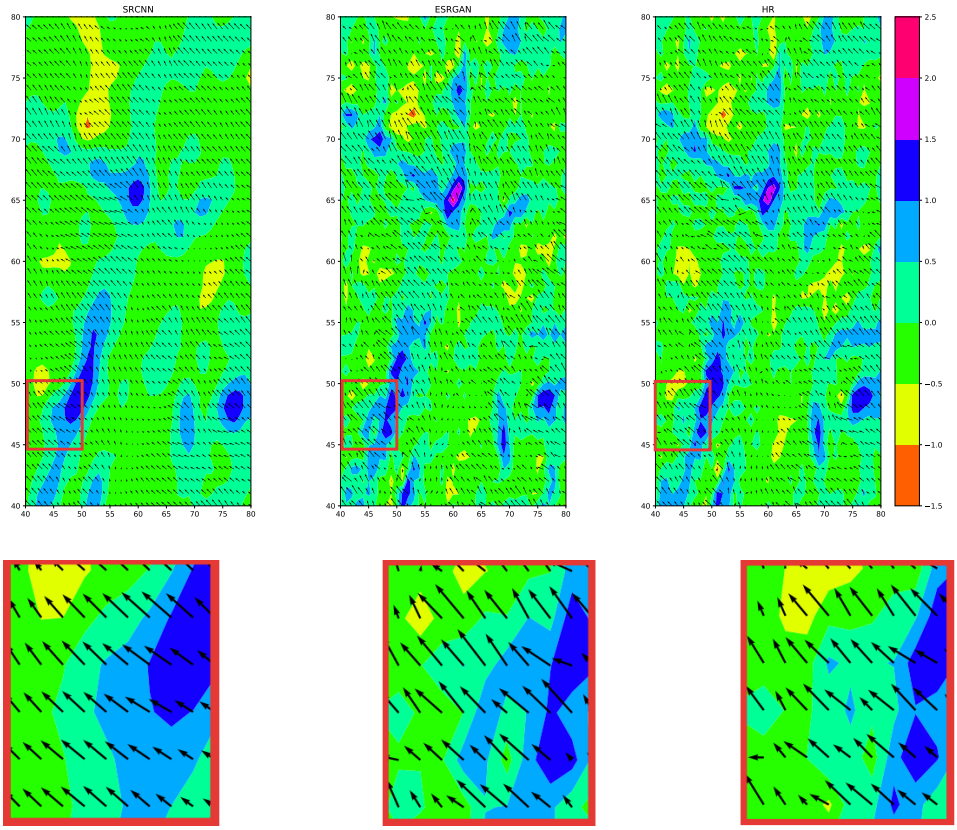


Figure 4.8: Zoomed in (10×5) and $\times 4$ upscaling qualitative results (from left to right) of the SRCNN, ESRGAN and high-resolution fields.

The reason SRCNN results in poor visual quality, even if the network appears to be performing well, is due to the choice of loss function. When we utilize a traditional loss function for measuring how accurate the generated image is compared to the real image, it measures how *mathematically* adjacent, i.e. the Euclidean distance, as opposed to how *visually* close the generated is to the real image. Furthermore, the main disadvantage of using the MSE as loss function in applications such as super-resolution is that it is computed pixel-wise, i.e. it only measures the difference between two corresponding pixels in the generated and the real images. This motivates detecting pixel-wise averages of plausible solutions which are usually overly-smooth and thus have inadequate perceptual quality.

The smooth averages of color in an area can be seen in Fig. 4.8 for SRCNN respectively. The ESRGAN counteracts this problem with the perceptual loss function, which measures the visual clarity, and thus resulting in finer details. This perceptual loss, i.e. the first term in Eq. 2.26, is the sum of the adversarial loss and content loss. One of the great benefits of using a GAN rather than a CNN is the utilization of the adversarial loss to motivate outputs to appear natural. This happens due to the fundamental nature of GANs: to detect data that looks like it does not belong. The content loss compares the fine details in images by sending the generated and original images through the feature maps of the neural network and calculating the loss on the outputs, i.e. how realistic do these generated images look.

The first reason for the ability of ESRGAN to fully reconstruct the wind field compared to the SRCNN model, is the replacement of simple convolution blocks with residual blocks, thus increasing the accuracy significantly. The second reason is the incorporation of transposed convolution for upsampling. In the SRCNN model, the images are first upsampled using bicubic interpolation and then fed as input to a simple CNN. Since nearest neighbor and bicubic interpolation are *unlearnable* methods, i.e. they can only be used before or after the neural network architecture and not in between, the accuracy and speed decreases significantly. On the other hand, ESRGAN uses two sub-pixel CNN layers for upscaling, as seen in the last part of the generator in Fig. 2.9. Thus, the speed and accuracy increases significantly.

As a matter of fact, the basis for the success of ESRGAN is due to the discriminator utilizing the relativistic average GAN, which learns to identify whether one image is more realistic than another. Hence, the generator is able to recover more detailed textures. Accordingly, Deng *et al.* [14] has shown blurry edges at the recirculation zone reconstructed by SRGAN, which becomes more evident with the increase in upscaling factor. Another reason is the utilization of features before activation, which ensures stronger supervision and thus the ability to recover more realistic textures and accurate brightness. Clearly, the discriminator network search for the high frequency information that differentiates LR and HR images, thus forcing the ESRGAN output to contain far more high frequency details than the output of the SRCNN.

The results are very promising, and the difference between SRCNN and ESRGAN is further strengthened from Fig. 4.9 and Fig. 4.10, where we notice the smoothness and lack of details in the SRCNN images.

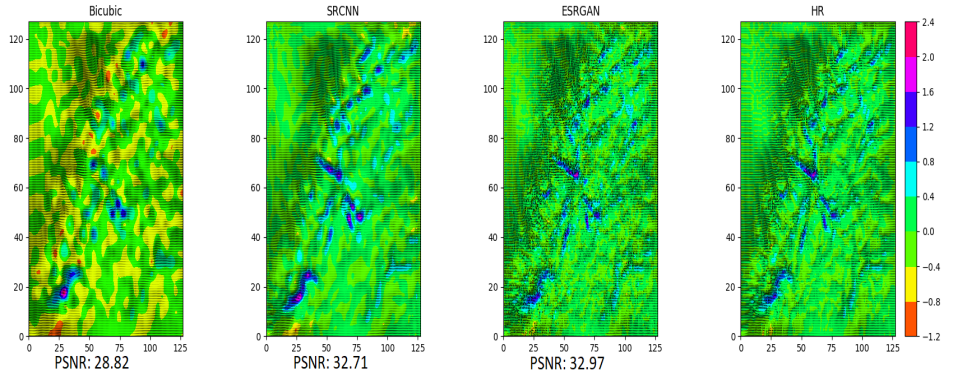


Figure 4.9: Comparisons of the $\times 4$ (from left to right) bicubic interpolation, SRCNN, ESRGAN and high-resolution fields.

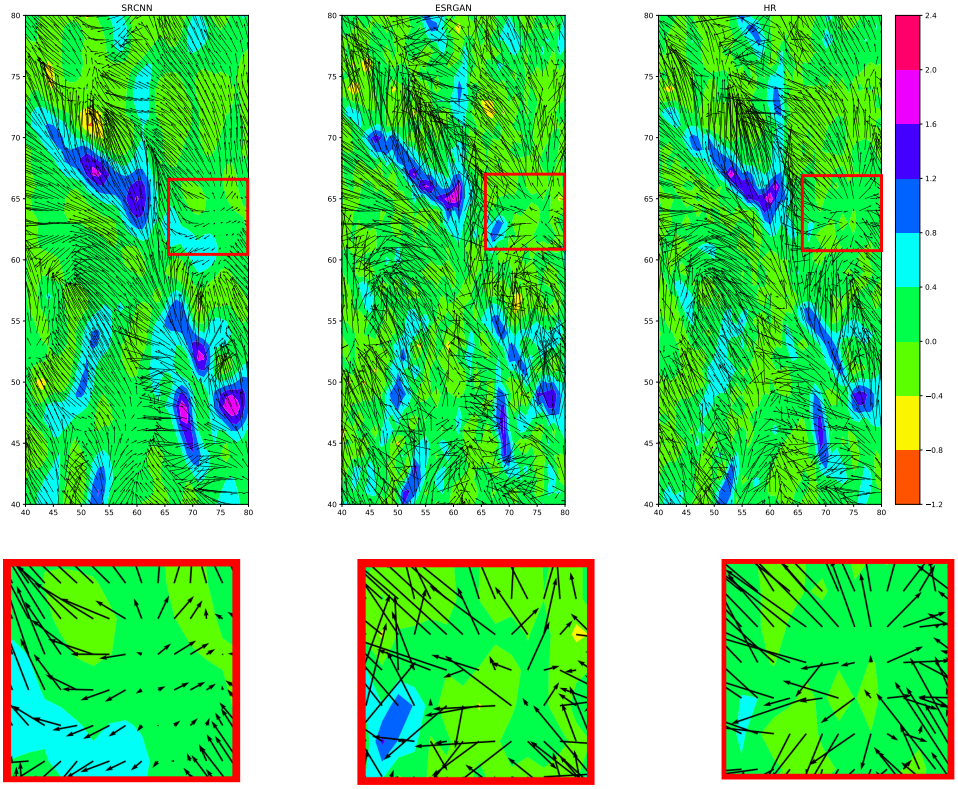


Figure 4.10: More zoomed in (10×5) and $\times 4$ upscaling qualitative results (from left to right) of the SRCNN, ESRGAN and high-resolution fields.

It is very clear that the ESRGAN is able to successfully reconstruct the high-resolution field consistently. The bicubic interpolated images are consistently worse than the two other models, since information lost from the downsampling phase is never recovered. Furthermore, SRCNN still over-smooths, but in a smaller degree than bicubic interpolation. On the other hand, ESRGAN learns the features of the wind flow and utilizes that information to recover the loss during the downsampling phase.

For a quantitative evaluation, the peak signal-to-noise ratio (PSNR) is also presented in Fig. 4.7 and Fig. 4.9. It can be clearly seen the GAN-based approach has consistently higher PSNR compared to the bicubic interpolation and CNN model. Fig. 4.11 presents the L2-norm computed from the high-resolution wind field and the reconstructed fields using nearest neighbor, bicubic interpolation, SRCNN and ESRGAN with respect to the 1800 high-resolution images. One can see that the SRCNN model results in significantly greater errors than ESRGAN, which is in agreement with the observation from Fig. 4.10.

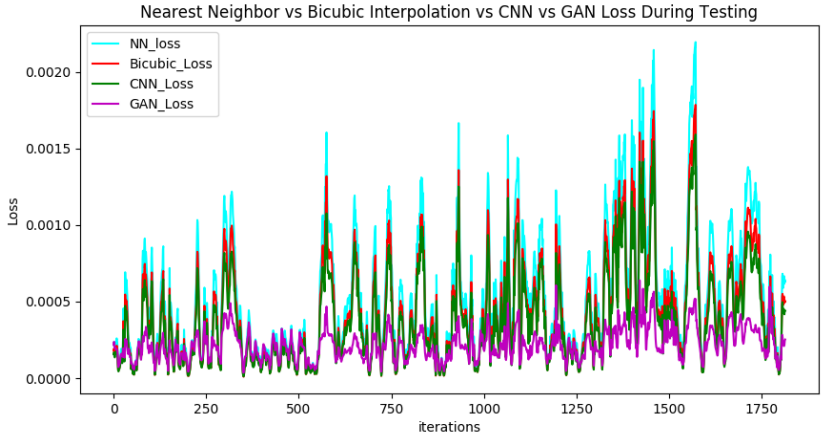


Figure 4.11: L2-norm error comparison of nearest neighbor (NN), bicubic interpolation, SRCNN and ESRGAN over part of the test set. The samples were taken from the September-October 2019 period. Each iteration corresponds to one hour.

Fig. 4.12 shows more qualitative results where the wind field in the first column corresponds to the high-resolution wind field obtained by applying bicubic interpolation on the subsampled wind field. Second column corresponds to the high-resolution wind field obtained by applying SRCNN on the subsampled wind field. The third column corresponds to the high-resolution field obtained from the application of the trained ESRGAN on the coarse field. Even though SRCNN significantly sharpens edges and is able to remove aliasing compared to bicubic interpolation, ESRGAN on the other hand produces additional textures yielding a much sharper, realistic-looking result. The ESRGAN output images contain high frequency information that is similar to the ground truth HR images, while SRCNN tend to smooth out the images in order to achieve a higher PSNR.

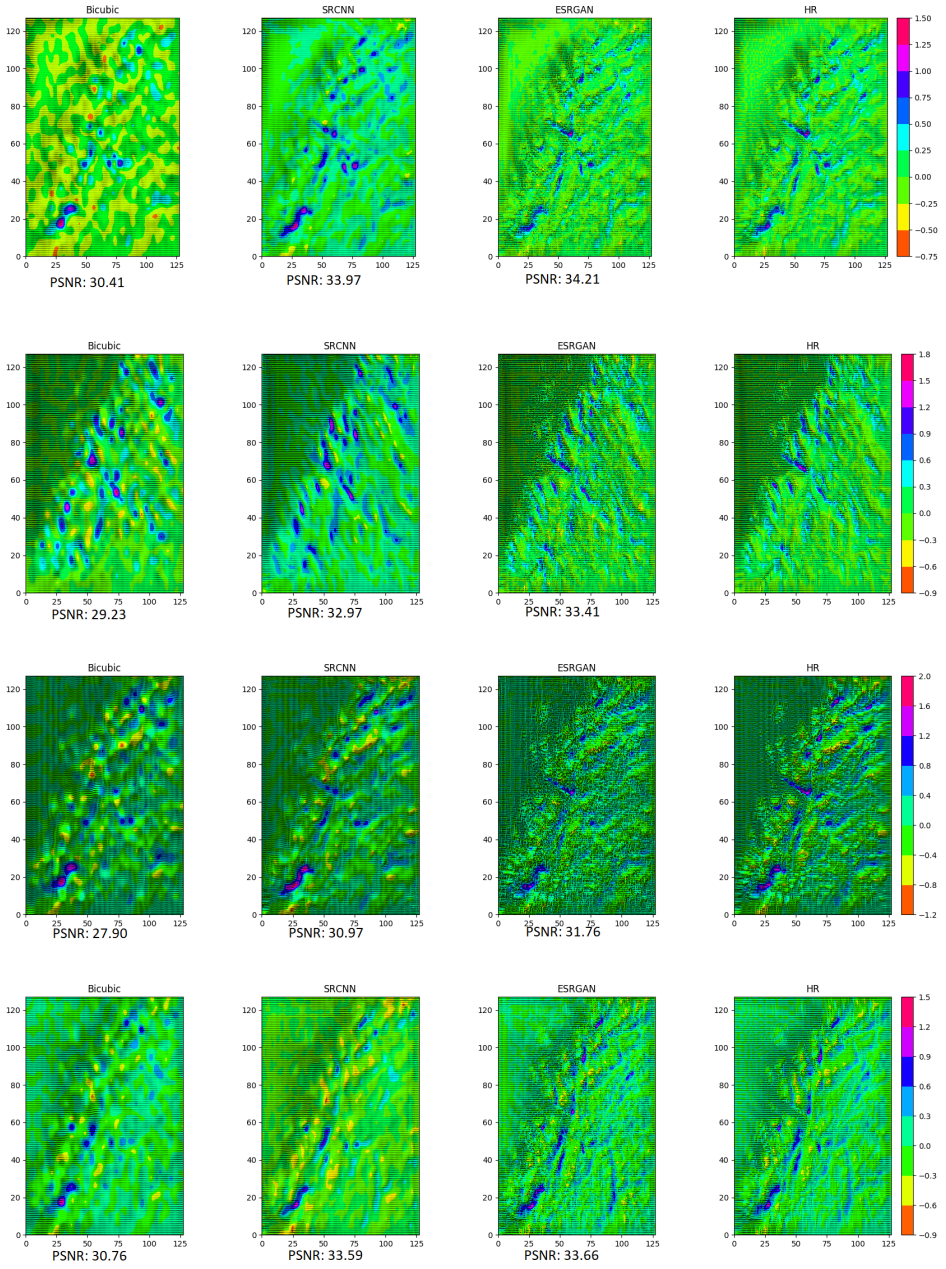


Figure 4.12: More $\times 4$ enhanced qualitative results (from left to right) of the bicubic interpolation, super-resolution CNN (SRCNN) Enhanced super-resolution GAN (ESRGAN) and high-resolution fields. Note the consistently higher value of PSNR of the ESRGAN generated field in comparison to SRCNN and bicubic interpolation.

Faithful high-frequency information is reproduced by ESRGAN, resulting in realistic wind field, at first glance almost indistinguishable from the ground truth fine scale wind field. Furthermore, ESRGAN generates high-frequency patterns missing completely in the coarser scale wind field, showing that the model is fully capable of detecting and generating patterns that lead to a realistic wind field.

A suitable solution to successfully reconstruct high-resolution flow in complex terrain is a necessity in many engineering settings, such as generating physically realistic high frequency to compute structural loads on wind turbines or upsampling experimental data that are constrained by sensor resolution. Due to the highly nonlinear nature of turbulence, we believe the difficulty conditions will increase as there are fewer feasible correlations between each velocity component. The findings here propose that the training data for our model can be very coarse.

Computational Time

Training GANs is a tedious and relatively expensive process. There are numerous factors that affect computational time, e.g. the validation set, which uses a runtime of 24 minutes to iterate through. For a total of $150k$ iterations, this results in additional $\sim 3h$ worth of time. Recall from Sec. 3.2 that a validation set is required in order to adjust the hyperparameters and avoid overfitting.

The limited memory of the GPU used in this work introduces a constraint on the batch size that can be processed at every epoch, since the feature maps need a significant amount of memory in the transitional layers. If the batch size is increased, the computational time will naturally decrease. However, deciding the correct number of batches can be a difficult task, since a larger batch size will lead to worse generalization.

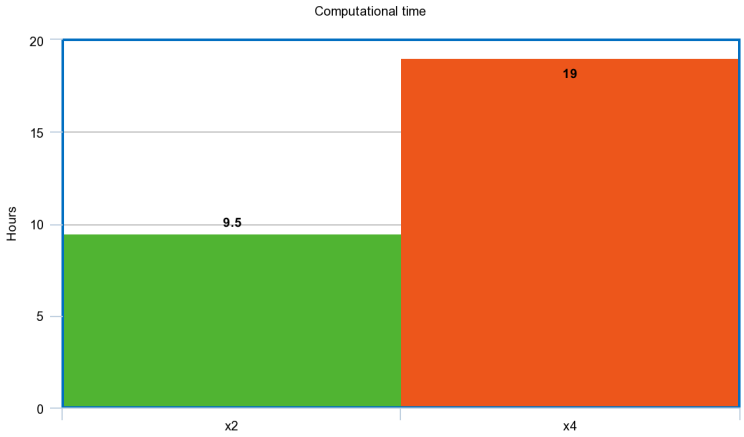


Figure 4.13: Computational time between $\times 2$ and $\times 4$ upscaling factors.

The computational time for the $\times 2$ and $\times 4$ upscaling factors is shown in Fig. 4.13. Notice that the upscaling factor $\times 2$ needs twice as less time to converge. This is expected since the $\times 4$ network requires far more iterations to learn the main characteristics of the flow. In addition, the network needs far more hyperparameter tuning when transitioning from upscaling factor $\times 2$ into $\times 4$, since the input wind fields into the generator are far coarser. In other words, the generator needs more time to recreate, i.e. generate realistic-looking wind field.

Even though the $\times 4$ upscaling model uses nearly twice as much time to train, we need to stress that once the model with the most satisfactory results is found, the training does not have to be performed again. Further, the model is used as a plug and play model with the coarser scale simulation results of SIMRA. Moreover, the SRCNN model needs around $19.5h$ to converge, which is nearly 30 minutes more than ESRGAN. Even with more training time, the results were inferior compared to ESRGAN. This fact further strengthens the superior performance of ESRGAN.

While the tuning process is intuitive in control theory, GANs on the other hand lack intuitive control knobs that influence the generated results. This can lead to a tedious process of adjusting hyperparameters and changing the network architectures. Since the initial few values of losses and generated samples will often never indicate any signs of progress, each training phase needs several hours before one can see some meaningful results. Certainly, due to the utmost separation of spatio-temporal scales, resolving all scales in turbulent flows have large costs.

4.2 Insight into the Inner Working of SRCNN and ESRGAN

In order to interpret and understand the inner workings of the network, feature maps from the hidden layers were extracted. Since the results from the last section proved the outperformance of ESRGAN compared to SRCNN, we will first focus on the former. From the description of the generator architecture in Sec. 2.7 it is noted that the first layer of the network has 32 filters of size 3×3. In actual fact these filters are $3 \times 3 \times 3$ because the wind fields have three velocity components, which is analogue to three color channels in digital images. Moreover, 32 is the lowest number of filters in any layer.

There exist 256 filters in some layers in the network, thus making the task of interpreting and visualizing them individually infeasible. It is worth highlighting that the result generated by the convolution operation on the wind field, interpreted as colorful images, do not in fact generate images that can be visualized in an understandable way. The reason for this is that the resulting matrix values are not constrained to the $0 - 255$ interval. For that reason, to actually create visualizations, the values were normalized and the default colormap "viridis" from Matplotlib was utilized. The colormap maps lower values closer to the color of dark blue and higher values closer to the color of yellow. Moreover, the input image that was fed to the pre-trained network from which intermediate feature maps were extracted is given by Fig. 3.2.

In Fig. 4.14 we see some plots of the intermediate images generated in the very first convolutional layer of the generator network. It can be observed from these images that the filters apparently generate every imaginable variant of the input image. Some filters sharpen the image while others blur it. A ton of the original information from the input image is preserved, because the initial convolution layers in CNNs (remember that the generator is actually a fully connected CNN) typically act as an edge detector. In Fig. 4.14e we observe strong gradients emphasizing edges, while in Fig. 4.14f the image is almost entirely smooth. In Fig. 4.14c and 4.14d it can be seen that edges are generated on opposing sides.

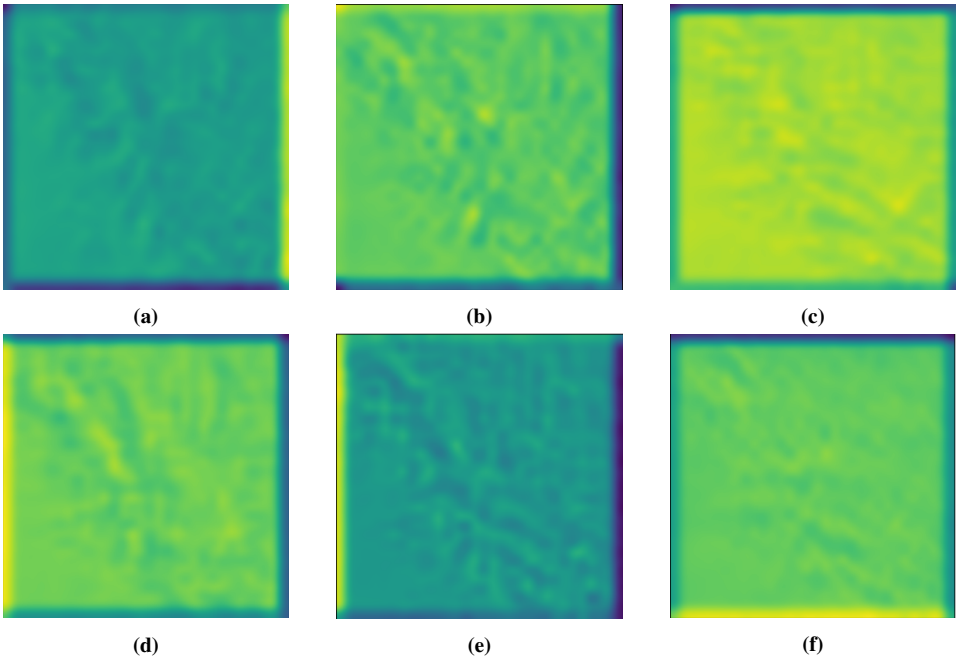
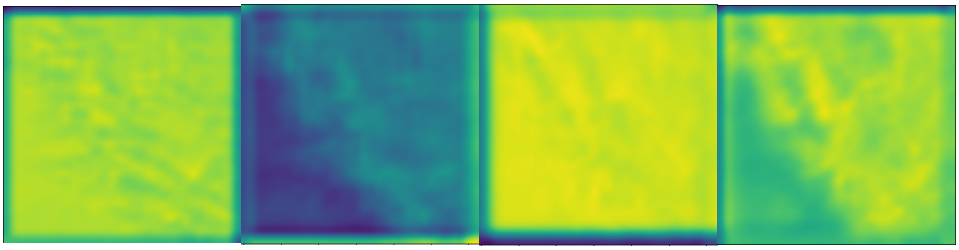


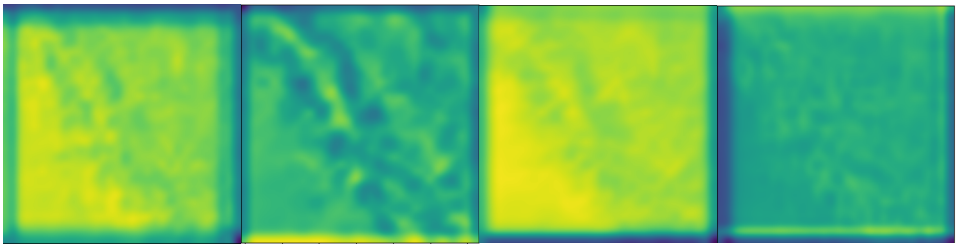
Figure 4.14: Intermediate results after the very first layer in the generator.

In Fig. 4.15 images reconstructed in deeper layers are presented. Not only are these interesting, they also give an intuition of what filters are searching for as we move deeper into the network. In the layers closer to the input layer, e.g. Fig. 4.15a, one can notice that the visualizations closely resemble the input image. We recall that the colormap "viridis" maps low values to dark blue and high values to yellow, and this distinction is followed throughout the layers. The generator is able to separate high and low values, as can be seen in e.g. Fig. 4.15d. It is after the first RRDB, i.e. Fig. 4.15f, that the generator is able to generate finer details in the middle section of the image. Gradually as one traverse through the layers the images contain more and more details.

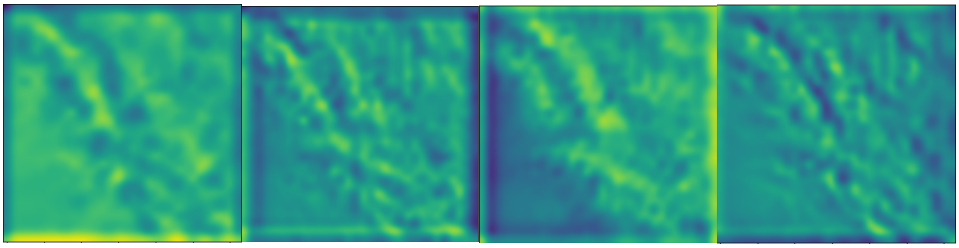
Recall that the generator utilizes residual learning, which scales down the residuals by multiplying a constant of 0.2 before adding them to the main path to prevent instability. For each residual block, the residual features after the last convolution layer are multiplied by 0.2. Intuitively, one can interpret the residual scaling to fix the improper initialization, thus preventing the magnitudes of input signals in residual networks from being amplified. The results of residual scaling can therefore be seen in e.g. Figs. 4.15g and 4.15h. After the fifth RRDB, i.e. Fig. 4.15j, the images start to make a bit more sense. Recognize the concatenated turbulent velocity fields that correspond to the same pattern as the input image. It is evident that the generator is able to "remember" what the original input image resembled. From this layer onwards, and to the end, these concatenated turbulent velocity fields become more and more distinct as observed in Figs. 4.15n, 4.15o and 4.15p.



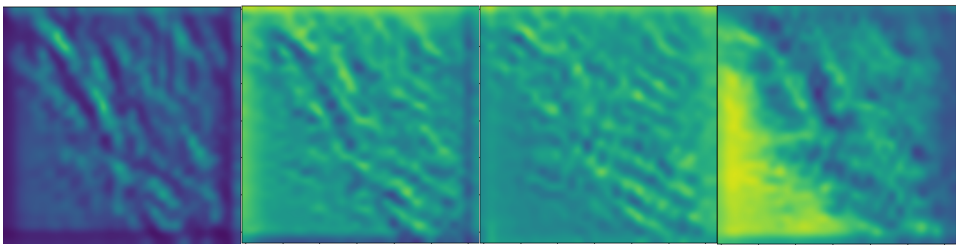
(a) L1: conv **(b)** L2 (RDB:1): first conv **(c)** L6 (RDB:1): last conv **(d)** L11 (RDB:2): last conv



(e) L16 (RRDB: 0, RDB:3): last conv **(f)** L17 (RRDB:1, RDB:1): first conv **(g)** L22 (RRDB:1, RDB:1): last conv **(h)** L34 (RRDB:1, RDB:3): last conv



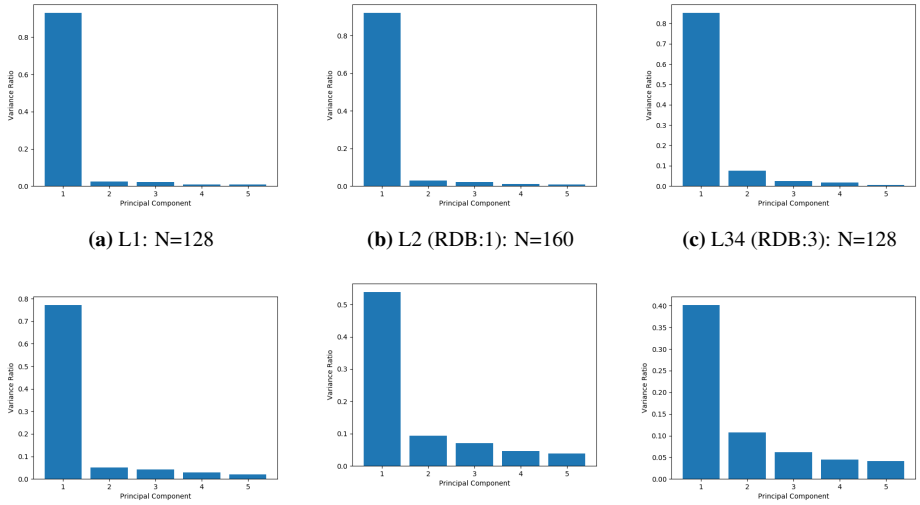
(i) L35 (RRDB:2, RDB:1): first conv **(j)** L91 (RRDB:5, RDB:3): last conv **(k)** L92 (RRDB:6, RDB:1): first conv **(l)** L166 (RRDB:10, RDB:3): last conv



(m) L167 (RRDB:11, RDB:1): first conv **(n)** L246 (RRDB:16, RDB:1): last conv **(o)** L251 (RRDB:16, RDB:2): last conv **(p)** L256 (RRDB:16, RDB:3): last conv

Figure 4.15: Feature maps from intermediate layers in generator.

The discussion has so far been based upon a bunch of images extracted from the respective layers. As clarified earlier, there can be up to 256 feature maps in some of the hidden layers in the generator, which is a tedious process to interpret. By conducting PCA on the images extracted from each hidden layer individually [61], we developed to some extent statistical understanding of the different layers. Fig. 4.16 presents the plots of the ratio of variance for the five most prominent principal components. It seems that for the early layers, as can be seen in in Figs. 4.16a and 4.16b, most of the information can be described using a single principal component. Progressively, as we traverse through the layers, it appears that more and more information is distributed over the images within a layer. In other terms the feature maps become increasingly distinct within each layer. This fact is evident from the first layer where all the images are quite similar, while the range of the difference of the images is much larger in the 251^{st} layer.



(a) L1: N=128 **(b)** L2 (RDB:1): N=160 **(c)** L34 (RDB:3): N=128

(d) L91 (RRDB:5, RDB:3): N=128 **(e)** L167 (RRDB:11, RDB:1): N=160 **(f)** L251 (RRDB:16, RDB:2): N=128

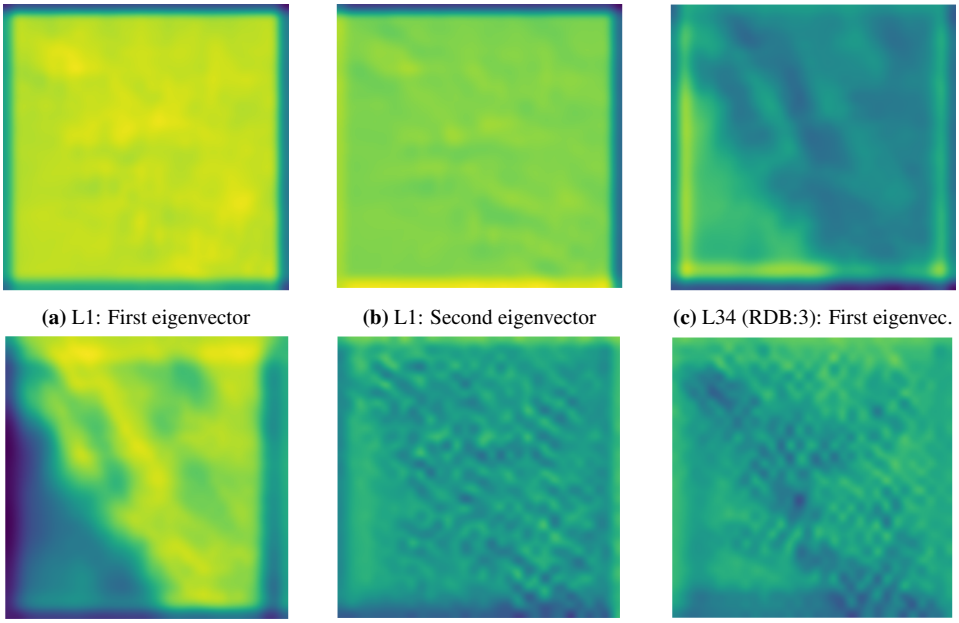
Figure 4.16: Bar plots: PCA analysis of intermediate layers in the generator. N is the number of feature maps.

It may be possible to use this kind of PCA as an optimization technique for the generator. The number of filters in the respective layer could possibly be reduced, since most of the variance is either explained by one, or only a few, principal components. The run time of the generator would be reduced if this could be implemented. One idea is to first train the network, and then run PCA to reduce the number of filters in the layers that are primarily described by a few principal components. Then we retrain the network and re-calculate PCA. This proposal can be done until the accuracy begins to fall beneath a specific threshold with regards to the accuracy of the original network. Note that the network may be unable to train because the previous layers may have contained some important information which PCA did not pick up. Ideally, the reduced network should have the same result as

the original one, but this is not a guarantee. Furthermore, the computational training time would be significantly larger, but the computational test time could be decreased while also retaining nearly the same accuracy as the initial network. Finally, this could make it simpler to understand and describe the network as a streamlined network is nonetheless simpler to analyze.

Fig. 4.17 displays visualizations made by implementing images from only the first and second most important component generated by the PCA. Notice in Fig. 4.16a that we preserve nearly all the variance just by using the first component. Accordingly, this is seen from Figs. 4.17a and 4.17b. Apparently, there is very few information in the second image that cannot be identified in the first. Furthermore, we can see from Figs. 4.17c and 4.17d that the principal components begin to refine in specific regions.

In the 251^{st} layer, as seen in Figs. 4.17e and 4.17f, we see that the level of distinctness has increased for the images. In Fig. 4.16f we observe that about 40% of the variance in this image is described by the first component. It is important to stress that every layer contains a huge amount of filters and that only a few selected convolved results have been visualized here. Hence, it is virtually impossible to draw any conclusions in general based on such limited visualizations.



(a) L1: First eigenvector

(b) L1: Second eigenvector

(c) L34 (RDB:3): First eigenvec.

(d) L34 (RDB:3): Second eigenvector

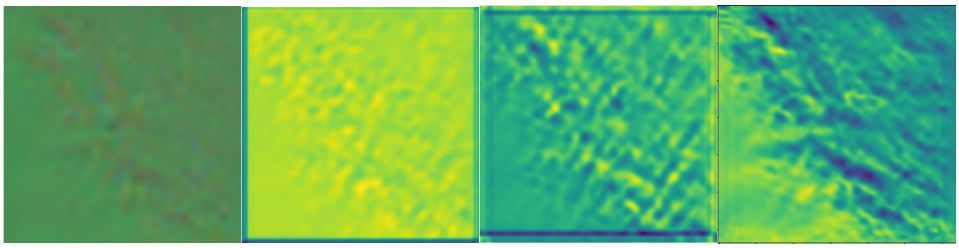
(e) L251 (RRDB:16, RDB:2): First eigenvector

(f) L251 (RRDB:16, RDB:2): Second eigenvector

Figure 4.17: Images: PCA analysis of intermediate layers in ESRGAN.

SRCNN

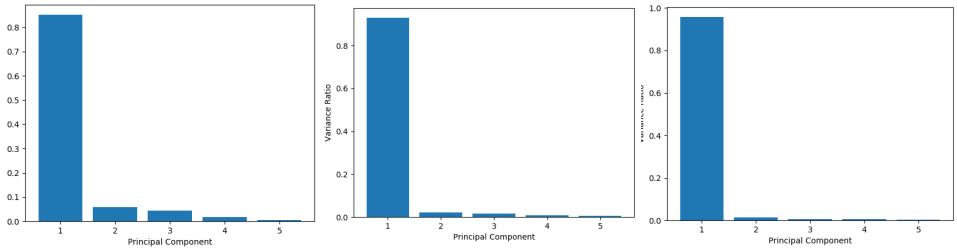
Fig. 4.18 shows the input and all three layers in SRCNN. We can immediately see that the first layer, i.e. Fig. 4.18b, is similar to the first layer of ESRGAN. Recall that the input of the SRCNN, i.e. Fig. 4.18a, is bicubic interpolated of the LR. This may be the reason why only three convolution layers are sufficient to yield relatively good results visually. Nonetheless, the ability to reconstruct and express wind field features is limited with only three convolution layers.



(a) Input **(b)** L1: conv **(c)** L2: conv **(d)** L3: conv

Figure 4.18: Input and feature maps from intermediate layers in SRCNN.

Once again, we apply PCA on the images extracted from all three layers individually in order to develop some statistical understanding. Fig. 4.19 shows the plots of the variance of the ratio for the five most prominent principal components. It appears that the variance of the second and third component decreases as we traverse through the network. As with the early layers in the generator, most of the information can be described by a single principal component. Overall, we have proven far better results with ESRGAN compared to the simplest structure of SRCNN, both quantitatively and visually, and we therefore end the analysis of SRCNN here.



(a) L1: N = 64 **(b)** L2: N = 32 **(c)** L3: N = 3

Figure 4.19: Bar plots: PCA analysis of intermediate layers in SRCNN. N is the number of feature maps.

Chapter 5

Conclusion

This thesis was initiated with the overall goal:

To explore the possibility of replacing computationally expensive high-resolution simulations with a combination of coarse scale simulation and advanced machine learning algorithms like CNNs and GANs.

In this work we presented two deep learning models, SRCNN and ESRGAN for super-resolution reconstruction of high-resolution wind fields from coarser scale, and provided insight into the internal workings of the models. A comparison of SRCNN and ESRGAN suggests that the GAN-based model has a better ability to reconstruct finer flow fields from coarser input fields than CNN ones due to competitive neural networks employed by it. Additionally, we provided a comparison between the L2-norm of the proposed models against state-of-the-art NN and bicubic interpolation methods. We highlight the major findings from the thesis below:

- The GAN-based artificial intelligence framework utilized in this work was able to learn the features and main characteristics of the wind flow in complex terrain without the aid of any equation or explicit programming of the physics.
- The GAN-based approach made use of the learned knowledge of the wind characteristics to successfully reconstruct finer scale wind field from previously unseen coarse wind field data.
- In a convincing manner, the GAN-based strategy convincingly outperformed both SRCNN and the state-of-the-art bicubic interpolation technique with the L2-norm and PSNR used as comparisons.
- Although the training of our model requires hyperparameter tuning and is a tedious process, the training is only done once and can accomplish a speed up to $10000\times$ in comparison to generating the same high-resolution field using a numerical simulator, whereas computational efficiency is much required in the context of Digital

Twin. This can enable further enhancement of wind farm optimization and resource assessment.

- The work led to a better insight into the inner working of both proposed models. Internal layers of the generator network were illustrated, and the multivariate data analysis tool PCA was applied to extract information from the enormous amount of filters utilized in the network. Examining the principal component vs. variance ratio plots for the different layers, we noticed that most of the variance in the reconstruction of the images utilizing the trained filters in the first few layers can be interpreted by a single component. The amount of components needed to describe the variance in the deeper layers gradually increases.
- It is important to stress the unresolved issues with the GAN-based approach. Even though we were able to produce outstanding results, the GANs do not generalize the problem. If we were to choose a different location than Bessaker, the model would potentially result in stability issues and not work as intended. Potential solutions to improve the generalization capability are therefore proposed in the next section.

In the current work we demonstrated the strength of combining a physics-based simulator with machine learning on a 2D dataset due to computational constraints. An extension to 3D is straightforward. The work also shows that machine learning models such as GANs and CNNs can be employed to enable prediction of accurate high-resolution flow fields from coarse flow fields for complex real life flow problems. The methodology of combining ML with physics-based models will be relevant in applications where one needs to save computational time associated with stand-alone high fidelity physics-based models, as now even the coarser results from physics-based can be used to reconstruct accurate finer flow field using the power of ML. We believe our contribution will be a key enabler for further work into applying machine learning in the fields of physics and toward Digital Twin.

5.1 Future Work

The presented results are promising, but our model still requires further work to ensure robustness and potentially better results. In this section, we will first discuss some improvements that can be made in our comparison analysis between ESRGAN and SRCNN. Next, we briefly discuss higher upscaling factors for computational efficiency and some techniques to reduce the computational complexity. Finally, we review different architectures in order to further improve our proposed model and ensure generalization.

5.1.1 Further Improvements on the Comparison Analysis

The current work uses the L2-norm, however, more flow-based metric parameters for comparison of results can be developed, like comparing energy spectra of reconstructed flow field by GANs/CNNs.

We focused mainly on ESRGAN due to its outperformance against SRCNN, but it would be interesting to see if we can improve the SRCNN architecture by modifying the cost function of the network to include normalized gradients. This enables a more thorough comparison analysis against ESRGAN.

PCA was only applied onto the generator to gain a statistical understanding of which information was generated. Hence, some aspects of the whole network are not completely understood yet and we should emphasize the discriminator network more in future work. What information does the discriminator actually use to classify? Also, it will be interesting to investigate the possibility of applying PCA to optimize the ESRGAN architecture even more. Further details of this optimization strategy can be found in Sec. 4.2.

One could also develop another data compression technique such as the Deep Autoencoder and compare against the PCA algorithm. Next, show how much it is possible to compress the data. Then, look into the possibility of on-the-fly method for dimensionality reduction, and investigate if it is possible to use the Deep Autoencoder as a turbulence scale separator.

5.1.2 Higher Upscaling Factors

The findings here suggest that the training data for our model can be very coarse. However, it is worth emphasizing that higher upsampling factors beyond $\times 4$ do not necessarily guarantee a desirable solution. At a higher upscaling factor the performance of ESRGAN is expected to be slightly degraded, and particular fine-scale structures will be unrecoverable. This can be due to the loss of much high-frequency information, which is in agreement with the results from Deng *et al.* [14] on wake flow behind two side-by-side cylinders. Furthermore, they experienced a minor blurring problem at the upscaling factor $\times 8$, and the blurring problem became more distinct with the upscaling factor $\times 16$.

5.1.3 Computational Time Reduction

The training phase of our model was a long and tedious process. Even though the training is only needed to be performed once after finding the optimal hyperparameters, there is still room for improvements. One could for instance try resizing the input images to smaller images or change the operations in the code from standard 32-bit floating point (FP32) to half-precision floating points (FP16) to accelerate training with the slight risk of a decrease in accuracy. Finally, it is worth trying to adopt more useful learning tricks to obtain faster and better network training:

- Adopt H-Swish [62] activation in the discriminator to accelerate the learning process.
- Remove the fully connected layers from the discriminator. Instead, use global averaging pooling to better catch the spatial information of feature maps [63].
- In the first 10k iterations have a larger learning rate, followed by a relative small learning rate to train the model.

5.1.4 Architecture Improvements of ESRGAN

Since the origin of the ESRGAN model back in late 2018, there have been some improvements on the network architecture. The following subsections will depict those, and should be further explored in future work.

ESRGAN+

Rakotonirina *et al.* [64] extended the original ESRGAN model to further enhance the perceptual quality of the images. In this fashion, they replaced the block used in the original model by designing a novel block. Moreover, the generator network was introduced to noise inputs in order to take advantage of stochastic variation. The reconstructed images contained more detailed structures and were less distinguishable from the HR images when compared to ESRGAN. As future work, we therefore propose a new block called Residual in Residual Dense Residual Block (RRDRB) [64] and introduce noise inputs in the network as in [65]. Note that ESRGAN+ outperforms the original model as long as perceptual quality is concerned.

Super-Resolution Using Segmentation-Prior Self-Attention Generative Adversarial Network (SPSAGAN)

Since recent studies have shown that unnecessary connections may affect the performance [66], Zhang *et al.* [67] investigated pruning methods to automatically remove unnecessary connections and obtain compact residual blocks. A lightweight skip connection design called Residual-in-Residual Sparse Block (RRSB) was introduced to prune unnecessary connections of RRDB, yielding better performance and reducing computational time. SPSAGAN was shown through extensive experiments that it can generate more realistic and visually pleasing textures in comparison to the state-of-the-art ESRGAN on many public benchmarks, such as generating more natural water waves on the OST dataset. Hence, it is worth investigating this suggested method. This can be done by extending our baseline network to add a segmentation-prior self-attention (SPSA) module and make it emphasize on the textures in the same segmentation class of the image.

Dual Reconstruction with Densely Connected Residual Network for Single Image Super-Resolution

Recently, Hsu *et al.* [68] proposed an improvement of the original ESRGAN, where they added one more shortcut between two dense blocks, along with a shortcut between two convolutional layers inside the two dense block. This simple strategy enabled a faster learning processing since the gradient information could be backpropagated more easily. The final super-resolution model was obtained by fusing two different models by weighted-summing their parameters. Through experimental results, the proposed method has demonstrated excellent performance in a real-world image super-resolution challenge. This strategy has also been verified to further enhance the quality of the reconstructed image contrary to the original ESRGAN in terms of both SSIM and PSNR. Additionally, they generated *more realistic* details in the reconstructed images.

Bibliography

[1] Adil Rasheed, Omer San, and Trond Kvamsdal. Digital twin: Values, challenges and enablers from a modeling perspective. *IEEE Access*, 2020.

[2] Manuela Lehner and Mathias W Rotach. Current challenges in understanding and predicting transport and exchange in the atmosphere over mountainous terrain. *Atmosphere*, 9(7):276, 2018.

[3] M Salman Siddiqui, Sidra Tul Muntaha Latif, Muhammad Saeed, Muhammad Rahman, Abdul Waheed Badar, and Syed Maaz Hasan. Reduced order model of offshore wind turbine wake by proper orthogonal decomposition. *International Journal of Heat and Fluid Flow*, 82:108554, 2020.

[4] Jian Yu, Chao Yan, and Mengwu Guo. Non-intrusive reduced-order modeling for fluid problems: A brief review. *Journal of Aerospace Engineering*, 233(16):5896–5912, 2019.

[5] Shady E Ahmed, Sk Mashfiqur Rahman, Omer San, Adil Rasheed, and Ionel M Navon. Memory embedded non-intrusive reduced order modeling of non-ergodic flows. *Physics of Fluids*, 31(12):126602, 2019.

[6] J Nathan Kutz. Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814:1–4, 2017.

[7] Markus Reichstein, Gustau Camps-Valls, Bjorn Stevens, Martin Jung, Joachim Denzler, Nuno Carvalhais, et al. Deep learning and process understanding for data-driven Earth system science. *Nature*, 566(7743):195–204, 2019.

[8] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[9] Samuel E Otto and Clarence W Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019.

[10] PA Srinivasan, L Guastoni, Hossein Azizpour, Philipp Schlatter, and Ricardo Vinuesa. Predictions of turbulent shear flows using deep neural networks. *Physical Review Fluids*, 4(5):054603, 2019.

[11] Romit Maulik, Omer San, Adil Rasheed, and Prakash Vedula. Subgrid modelling for two-dimensional turbulence using neural networks. *Journal of Fluid Mechanics*, 858:122–144, 2019.

[12] Michele Alessandro Bucci, Onofrio Semeraro, Alexandre Allauzen, Guillaume Wisniewski, Laurent Cordier, and Lionel Mathelin. Control of chaotic systems by deep reinforcement learning. *Proceedings of the Royal Society A*, 475(2231):20190351, 2019.

[13] Jean Rabault and Alexander Kuhnle. Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. *Physics of Fluids*, 31(9):094105, 2019.

[14] Zhiwen Deng, Chuangxin He, Yingzheng Liu, and Kyung Chun Kim. Super-resolution reconstruction of turbulent velocity fields using a generative adversarial network-based artificial intelligence framework. *Physics of Fluids*, 31(12):125111, 2019.

[15] Jared L Callaham, Kazuki Maeda, and Steven L Brunton. Robust flow reconstruction from limited measurements via sparse representation. *Physical Review Fluids*, 4(10):103907, 2019.

[16] Ryan King, Oliver Hennigh, Arvind Mohan, and Michael Chertkov. From deep to physics-informed learning of turbulence: Diagnostics. *arXiv preprint arXiv:1810.07785*, 2018.

[17] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. TEMPOGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Transactions on Graphics (TOG)*, 37(4):1–15, 2018.

[18] Liu Yang, Dongkun Zhang, and George Em Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *arXiv preprint arXiv:1811.02033*, 2018.

[19] Panos Stinis, Tobias Hagge, Alexandre M Tartakovsky, and Enoch Yeung. Enforcing constraints for interpolation and extrapolation in generative adversarial networks. *Journal of Computational Physics*, 397:108844, 2019.

[20] Jin-Long Wu, Karthik Kashinath, Adrian Albert, Dragos Chirila, Heng Xiao, et al. Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems. *Journal of Computational Physics*, 406:109209, 2020.

[21] Junhyuk Kim and Changhoon Lee. Deep unsupervised learning of turbulence for inflow generation at various Reynolds numbers. *Journal of Computational Physics*, page 109216, 2020.

[22] Mathis Bode, Michael Gauding, Zeyu Lian, Dominik Denker, Marco Davidovic, Konstantin Kleinheinz, Jenia Jitsev, and Heinz Pitsch. Using physics-informed super-resolution generative adversarial networks for subgrid modeling in turbulent reactive flows. *arXiv preprint arXiv:1911.11380*, 2019.

[23] Sangseung Lee and Donghyun You. Prediction of laminar vortex shedding over a cylinder using deep learning. *arXiv preprint arXiv:1712.07854*, 2017.

[24] Jinu Lee, Sangseung Lee, and Donghyun You. Deep learning approach in multi-scale prediction of turbulent mixing-layer. *arXiv preprint arXiv:1809.07021*, 2018.

[25] Maximilian Werhahn, You Xie, Mengyu Chu, and Nils Thuerey. A multi-pass GAN for fluid flow super-resolution. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2(2):1–21, 2019.

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

[27] Thomas B. Gatski, M. Yousuff Hussaini, and John L. Lumley. Simulation and modeling of turbulent flows. page 314, 1996.

[28] Rodrigo M. S. de Oliveira, Ramon C. F. AraÃ, FabrÃcio J. B. Barros, Adriano Paranhos Segundo, Ronaldo F. Zampolo, Wellington Fonseca, Victor Dmitriev, and Fernando S. Brasil. A System Based on Artificial Neural Networks for Automatic Classification of Hydro-generator Stator Windings Partial Discharges. *Journal of Microwaves, Optoelectronics and Electromagnetic Applications*, 16:628 – 645, 09 2017.

[29] Wikipedia contributors. Artificial neural network. 2020. `https://en.wikipedia.org/wiki/Artificial_neural_network`.

[30] Michael A. Nielsen. Neural networks and deep learning, 2018.

[31] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678, 2016.

[32] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[34] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2016.

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[36] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, apr 1980.

[37] Mikkel Cornelius Nielsen. Lecture notes in underwater robotics for safe and autonomous subsea operations. 2018.

[38] Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep sparse rectifier neural networks. volume 15, 01 2010.

[39] Tianyi Liu, Minshuo Chen, Mo Zhou, Simon S. Du, Enlu Zhou, and Tuo Zhao. Towards understanding the importance of shortcut connections in residual networks, 2019.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[41] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. 2014.

[42] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks, 2014.

[43] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2016.

[44] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[47] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *The European Conference on Computer Vision Workshops (ECCVW)*, September 2018.

[48] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2016.

[49] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, 2016.

[50] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan, 2018.

[51] Lindsay I Smith. A tutorial on principal components analysis. Technical report, Cornell University, USA, February 26 2002.

[52] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[53] Adil Rasheed, Jakob Kristoffer Süld, and Trond Kvamsdal. A multiscale wind and power forecast system for wind farms. *Energy Procedia*, 53:290 – 299, 2014. EERA DeepWind' 2014, 11th Deep Sea Offshore Wind RD Conference.

[54] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NeurIPS Autodiff Workshop*, 2017.

[55] Unidata. Netcdf http://doi.org/10.5065/d6rn35xm. 2019.

[56] OPeNDAP. https://www.opendap.org/. 2019.

[57] Magnus Själander, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure, 2019.

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

[59] Sutskever I. Hinton G.E. Krizhevsky, A. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[60] Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. How to train a gan? tips and tricks to make gans work, 2016.

[61] Sidharth Mishra, Uttam Sarkar, Subhash Taraphder, Sanjoy Datta, Devi Swain, Reshma Saikhom, Sasmita Panda, and Menalsh Laishram. Principal component analysis. *International Journal of Livestock Research*, page 1, 01 2017.

[62] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[63] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2016.

[64] Nathanaël Carraz Rakotonirina and Andry Rasoanaivo. Esrgan+ : Further improving enhanced super-resolution generative adversarial network, 2020.

[65] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2018.

[66] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions, 2017.

[67] Yuxin Zhang, Zuquan Zheng, and Roland Hu. Super resolution using segmentation-prior self-attention generative adversarial network, 2020.

[68] Chih-Chung Hsu and Chia-Hsiang Lin. Dual reconstruction with densely connected residual network for single image super-resolution, 2019.