Master's thesis

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

Helle Mortensen Gråberg

Towards Physics-Informed Neural Networks for Urban Wind Flow Prediction

Master's thesis in Computer Science
Supervisor: Massimiliano Ruocco
February 2022

NTNU
Norwegian University of
Science and Technology

NABLAFLOW

Helle Mortensen Gråberg

Towards Physics-Informed Neural Networks for Urban Wind Flow Prediction

Master's thesis in Computer Science
Supervisor: Massimiliano Ruocco
February 2022

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

NTNU
Norwegian University of
Science and Technology

Abstract

Computational Fluid Dynamics (CFD) is a scientific field encompassing methods for simulating fluid dynamics. These methods are used to solve a wide range of complex engineering problems, including urban wind flow prediction. With increased urbanization, assessing how new buildings affect the wind flow is vital to ensure pedestrian comfort and safety. However, while CFD simulations provide detailed information about the fluid flow they are extremely complex and computationally expensive.

The rapid improvements of deep learning have allowed researchers to model fluid dynamics using neural networks. The fast inference time of neural networks makes it possible to predict the state of a fluid system in a matter of seconds, making processes such as interactivity possible. Using neural networks to predict fluid flow trades detailed information about the fluid flow for faster approximation time. This trade-off is extremely desirable when assessing how wind flow in urban areas affects pedestrians. Predicting how changing building geometries affect high-level wind trajectories will allow architects to interactively design buildings that are suited for urban development. Unfortunately, using CFD to generate enough training data for the neural network to learn how to describe a fluid system is still computationally expensive.

To combat the need for large amounts of data, a new paradigm of physics-AI coupling has emerged. Neural networks are programmed to learn from data from CFD simulations as well as the physical equations governing the fluid system. These PINNs (Physics-Informed Neural Networks) are showing great potential for describing fluid systems. However, researchers are still working towards the coupling of the most complex fluid systems and state-of-the-art deep learning.

Turning our attention towards the complicated domain of turbulent fluid flow, we introduce a PINN on several fluid flow systems of increasing complexity. The PINN is developed by combining a state-of-the-art deep learning architecture, the UNet, and a conservation law of fluid dynamics. We show that the PINN is able to efficiently learn the fluid flow of our systems, and shows increased interpretability in terms of the physics governing the fluid systems. Notably, the PINN shows the highest improvement in interpretability in the low-resource region, where simulated data is scarce.

Sammendrag

Numerisk fluidmekanikk (CFD) er et fagområde som inkluderer metoder for simulering av fluidmekanikk. Disse metodene brukes for å løse en rekke komplekse problemer, blant annet predikering av vind i urbane områder. Med økende grad av urbanisering er det viktig å undersøke hvordan nye bygninger påvirker vindstrømmen, slik at man sørger for komfort og sikkerhet for fotgjengere. CFD-simuleringer tilbyr detaljert informasjon om fluid strømning, men er ekstremt komplekse og krevende å beregne.

Den raske utviklingen innen dyp læring har tillatt forskere å modellere fluidmekanikk ved bruk av nevrale nettverk. Den raske prediksjonstiden til nevrale nett betyr at fluidsystemer kan beskrives i løpet av sekunder, noe som muliggjør interaktivitet. Ved å bruke nevrale nettverk for å predikere fluidstrømninger kan man oppnå raskere approksimeringer, men man gir slipp på noen av detaljene i fluidstrømmen. Denne avveiningen er meget verdifull når det gjelder å undersøke hvordan vindstrømmene i urbane områder påvirker fotgjengere. Predikering av hvordan de overordnene vindstrømmene endrer seg når man endrer geometrien til bygninger vil tillate arkitekter å interaktivt designe bygninger som er tilpasset urban utvikling. Problemet med slike metoder er at det fremdeles er krevende å bruke CFD til å fremstille nok treningsdata til at det nevrale nettet kan lære å beskrive fluidsystemet.

For å redusere behovet for store menger data har en kobling mellom fysikk og kunstig intelligens begynt å utvikle seg. Nevrale nett programmeres til å lære både fra data fra CFD-simuleringer og fra fysikkligningene som beskriver fluidsystemet. Slike PINN-er (fysikk-inspirerte nevrale nettverk) har vist seg å ha stort potensiale for å beskrive fluidsystemer, men forskere arbeider fremdeles med koble sammen de mest komplekse fluidsystemene og toppmoderne nevrale nett.

Vi retter fokuset mot det kompliserte domenet turbulente strømninger, og introduserer en PINN for en rekke strømningssystemer med økende kompleksitet. Vår PINN er utviklet ved å kombinere en toppmoderne nevral nettverksarkitektur, UNet, og en konserveringslov innen fluidmekanikk. Vi viser at PINN-en effektivt lærer seg strømningene i systemene vi studerer, og at den følger fysiske lover som beskriver systemet. Vi ønsker også å vektlegge at PINN-en viser størst forbedring når den trenes på få eksempler fra CFD-simuleringer.

Preface

This thesis concludes my Master of Science (MSc) degree at the Department of Computer Science at the Norwegian University of Science and Technology (NTNU). It was written as part of a collaboration between the Norwegian Open AI Lab (NAIL) and Nabla Flow. I am especially fortunate to have worked on a topic that combines my two main areas of academic interest – artificial intelligence and physics. I believe that these two areas have a lot to offer each other, and I hope this thesis can prove useful to students and researchers looking to further integrate these domains.

I would like to thank my supervisor, Massimiliano Ruocco at NTNU, for excellent guidance throughout my work on this thesis. As this thesis does not serve as a continuation of my specialization project, I would also extend my gratitude to Massimiliano for deciding to supervise me for my final semester. Additionally, I would like to thank both him and Luca Oggiano, from Nabla Flow, for proposing this thesis topic. To Luca, I also want to recognise the support you have given me throughout my thesis work.

I would also like to thank Eliezer de Souza da Silva at NTNU, and Fredrik Fang Liland and Knut Erik T. Giljarhus at Nabla Flow. Thank you Eliezer for your invaluable input on the behaviour of advanced neural networks. Thank you Fredrik and Knut Erik for the datasets used in this project, and for your guidance on the topic of CFD.

Lastly, I want to thank my friends for providing feedback on my writing, and for our discussions around the content of this thesis. Thank you Sindre Stenen Blakseth, Simen Burud, and Alexandra Metallinou Log.

Trondheim, February 2022
Helle Mortensen Gråberg

Contents

List of Figures

List of Tables

Acronyms

CFD Computational Fluid Dynamics.

CNN Convolutional Neural Network.

GAN Generative Adversarial Network.

LES Large Eddy Simulations.

PCA Principal Component Analysis.

PDE Partial Differential Equation.

PGNN Physics-Guided Neural Network.

PINN Physics-Informed Neural Network.

RANS Reynolds-Averaged Navier–Stokes.

ReLU Rectified Linear Unit.

VD Velocity Divergence.

Chapter 1

Introduction

This chapter contains an introduction to this thesis. First, section 1.1 contains the background and motivation for investigating the use of neural networks to solve fluid flow problems. Then, section 1.2 presents the overall research goal and three research questions that are to be investigated. The contributions of this thesis are summarized in section 1.3. Lastly, the structure of the thesis is presented in section 1.4.

1.1 Background and Motivation

Computational Fluid Dynamics (CFD) is the task of numerically simulating the fluid flow of a system by solving a set of Partial Differential Equations (PDEs). To simulate the flow of wind, CFD is used as a cheaper and more accessible alternative to experiments in wind tunnels. However, numerical experiments are still extremely time- and resource-intensive, requiring expensive hardware to run simulations on.

CFD methods have stayed the same for several decades, but the development of computer hardware has allowed more complex physical simulations. Improved computer hardware has also allowed the advancement of neural networks. Neural networks are able to predict solutions to PDEs in a matter of seconds. There are several practical applications of predicting fluid flow in real-time. Faster flow predictions make it possible to simulate systems with higher resolution, predict the future state of a system further ahead in time, and to include more realistic models in the optimization of environmental systems. Another application of faster simulations is interactivity with the fluid system, which is our main motivation for moving towards real-time prediction of urban wind flow. Such interactivity will allow architects to iteratively design buildings that are suited for urban development. New buildings are evaluated by their impact on the surrounding microclimate, and must adhere to rules and regulations regarding this impact. The evaluations can be conducted with CFD simulations. Providing architects with a tool to ensure that their building designs adhere to regulations reduces the number of expensive CFD simulations that must be conducted. In Figure 1.1, we illustrate how a CFD simulation can be used to depict wind movement through an urban area.

Due to the practical applications of predicting fluid flow in real-time, such as interactivity with the fluid system, a branch of research using neural networks to approximate physical systems has emerged. Early attempts to model PDEs using neural networks include the framework developed by Dissanayake and Phan-Thien 1994, but as generating training data become more feasible, data-driven approaches have gained popularity. These approaches require very little knowledge of fluid dynamics - data-driven neural networks[1] learn to map an input to an output by looking at input-target pairs of data.

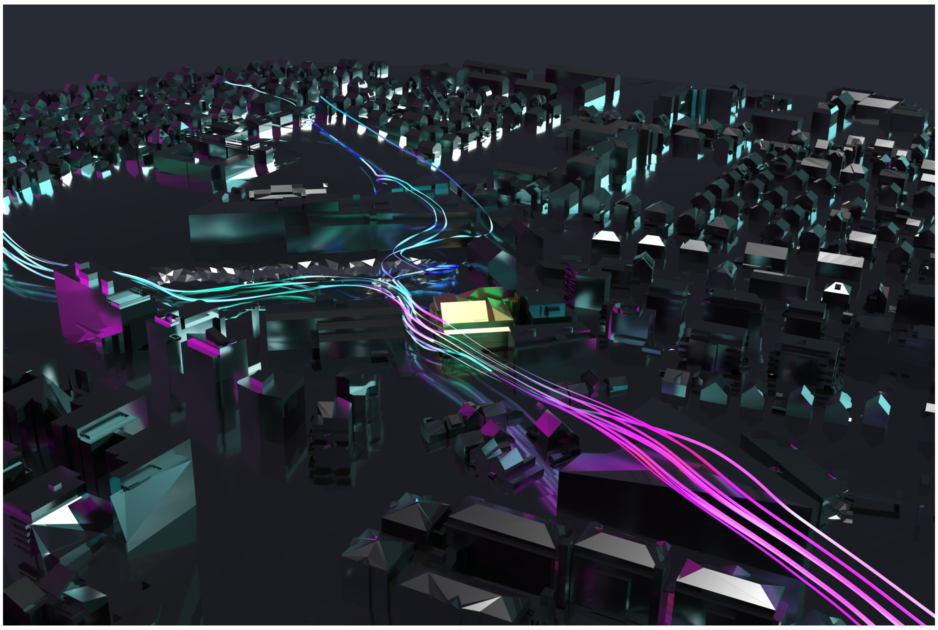
Figure 1.1: An illustration of how the wind moves through an urban area, generated with a CFD simulation. Courtesy of Nabla Flow.

[1]The term “data-driven“ refers to “normal“ neural networks in an explicit way, as neural networks usually are trained on datasets with input-target pairs.

1.1.1 CFD as a Low-Resource Domain

The size of the dataset used to train a neural network has a high impact on the resulting predictions from the neural network. The size needed for the dataset depends on the task at hand: The examples in the dataset must accurately capture the underlying distribution which we want the neural network to learn. Complex tasks thus need larger amounts of data than simpler tasks. An example of a complex task is the prediction of wind flow in urban areas, due to the turbulent nature of wind and the complex geometries of real-life buildings.

While in theory, one could produce infinite amounts of training data using CFD simulations, in practice this would take a very long time. Generating only one training example in 3D takes several hours, making it infeasible to generate a large dataset. Brunton et al. 2020 reported that in the field of fluid dynamics, datasets for complex problems do not model the complete underlying distribution that one would want the neural network to learn. Improving the generalization capabilities of the neural network is thus vital for accurate fluid flow predictions.

For low-resource domains where one wishes to use deep learning, data augmentation can often be used to expand the size of the dataset. Data augmentation is the process of creating new data from the existing dataset. For image data, this could for instance be by skewing or rotating the images. For modelling fluid flow, data augmentation is not always feasible as augmenting the dataset could lead to the physical laws and initial conditions breaking.

On the other hand, these physical laws and initial conditions can be used to constrain the optimization space, simplifying the optimization task of the neural network. Raissi et al. 2017 introduced the Physics-Informed Neural Network (PINN) – a neural network that is programmed to learn from both data samples and the governing equations of the physics system. In addition to reducing the need for training data, the PINN ensures that the physical laws of the system are obeyed. However, adapting the PINN to a new domain is more complex than just using data-driven neural networks. In addition to generating the required amount of training data, the equations describing the system must be modelled in the neural network.

1.2 Goals and Research Questions

This thesis is a part of an ongoing project with Nabla Flow and SINTEF Digital, aiming to offer interactive approximation of urban wind conditions. Urban wind conditions are highly affected by the shape and size of buildings, so such interactivity would allow architects to test how their designs impact the wind flow. Estimating urban wind flow is important for accommodating pedestrians, and has become more important in recent years due to ongoing urbanization.

While we work towards increasing the interactivity of urban wind prediction using neural networks, the focus of this thesis is to investigate if a PINN is suitable for wind flow prediction. To determine whether a PINN is suitable for wind flow predictions, we propose three research questions.

Research question 1 *How does the addition of physics knowledge affect a neural network's training process?*

Our hypothesis is that including physics knowledge should help the neural network navigate the optimization landscape. During training, a smaller solution space should lead to more stable predictions as well as faster convergence.

Research question 2 *How does the PINN's performance differ as the amount of training data changes?*

We hypothesize that constraining the solution space should reduce the amount of training data needed for a neural network. Thus, the performance of the PINN with small amounts of training data should be similar to, or better than, the performance of the data-driven neural network with larger amounts of training data. Another way of looking at this hypothesis is that the inclusion of physical equations provides more information to the neural network by making each data sample more salient.

This research question also turns our focus to the low-resource region, in terms of the amount of training data required. By evaluating the performance across training set sizes we are hoping to learn where the number of samples is so small that the neural network is struggling. Evaluating the PINNs performance in such a region tells us how much we can benefit from using PINNs to model fluid flow with CFD as a low-resource domain.

Research question 3 *How well can the PINN generalize to a more complex data distribution?*

PINNs are gaining popularity due to their generalization capabilities on unseen data samples. However, their performance on unseen data *distributions* is rarely discussed - probably due to neural networks' lack of generalization capabilities outside of their training distribution. As the task of urban wind prediction will require a model that is robust to a large range of building geometries, we propose experiments to learn if the PINN can outperform a traditional neural network on unseen distributions.

1.3 Contributions

We bring state-of-the-art deep learning techniques to turbulent fluid flow prediction. Using a UNet model with a physics-informed loss function on increasingly complex datasets, we obtain consistent improvements over the already strong baseline.

Making use of a range of training set sizes is key to our approach. We find that overall, using the physics-informed loss term improves the training process of the neural network. In particular, we find that both the training stability and the convergence of the PINN are better than that of the baseline. Our results show that for turbulent flow, utilizing a PINN allows for a reduction in the length of the training process regardless of the size of the training set.

Further, we demonstrate that the PINN consistently outperforms the baseline in terms of interpretability. In the region with minimal amounts of training data, this improvement is even higher - our PINN is more sample efficient in the low-resource region.

Finally, we analyse how embedding the physics-informed loss term in the neural network affects its generalization capabilities. Our PINN is able to better generalize to more complex data than what it has seen during training. However, our experiments show that more research is needed to reach the full potential of PINNs as general fluid flow approximators.

A key component to our investigation, apart from the use of PINNs, is the use of increasingly complex datasets. Our PINN performs well regardless of dataset complexity, suggesting that PINNs are viable models for real-life wind prediction in urban areas.

Lastly, some of the novelty of our idea lies in the simplicity of the input provided to the neural networks. While other works, such as Ma et al. 2021 and Wandel et al. 2020, input more specific information to the neural network, we only provide the geometry of the buildings. This approach is inspired by the basic principles of deep learning, regarding the abilities of neural networks to extract relevant features from the input.

1.4 Thesis Structure

The next chapter contains a brief theoretical overview of CFD, neural networks, and the integration of deep learning and physics. In chapter 3, we report state-of-the-art methods for using neural networks to predict fluid flow. Chapter 4 outlines our neural network architecture and defines our physics-informed loss terms. Chapter 5 details the datasets and evaluation metrics used and how the experiments were conducted, while chapter 6 analyses the outcome. Finally, in chapter 7 we summarise the results in terms of the research questions and provide pointers for further research.

Chapter 2

Background Theory

This chapter contains the theoretical background for this thesis. First, computational fluid dynamics is described in section 2.1. Then, relevant aspects of artificial neural networks are presented in section 2.2. Finally, modern integration of the two fields are presented in section 2.3.

2.1 Computational Fluid Dynamics

Computational Fluid Dynamics (CFD) is the numerical modelling of fluid flows, generally described by the Navier–Stokes equations. This section presents theory regarding these equations, as well as how to solve them. The theory, unless otherwise stated, is based off of Malalasekera and Versteeg 2007.

Solving the Navier–Stokes equations analytically is not possible in general, and solving them numerically often requires immense computational power. The increase in available computing power over the years has allowed for larger and more complex simulations. In comparison, commercial CFD methods have largely stayed the same for decades, as shown by Slotnick et al. 2014. Still, reaching the steady-state solution for a simulation may take several days. Reaching steady-state is especially problematic for turbulent flow, where the changes in pressure and velocity are chaotic and the system therefore exhibits highly unpredictable behaviour.

2.1.1 The Navier–Stokes Equations

The Navier–Stokes equations are second-order partial differential equations describing the velocity $\vec{v}$ and pressure p of a fluid flow. For an incompressible fluid where external forces, such as gravity, are neglected, the Navier–Stokes equations are

$$\nabla \cdot \vec{v} = 0 \tag{2.1a}$$

$$\rho \left(\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\vec{v} \right) = -\nabla p + \mu \Delta \vec{v}, \tag{2.1b}$$

where $\vec{v}$ is the velocity, ρ describes the fluid density, p is the pressure, and μ is the viscosity. Equation 2.1a is the continuity equation, stating that the velocity divergence must be zero in a fluid system without sinks or sources. Equation 2.1b describes the conservation of momentum within the fluid system. Solving these equations results in the velocity field of the fluid, $\vec{v}$, and the pressure of the fluid, p. By setting the time-derivative equal to zero, one can determine the steady-state flow of the fluid.

The Navier–Stokes equations do not constitute a fully determined system of equations on their own. To make the system fully determined, a set of initial conditions and boundary conditions have to be prescribed. There exist several types of boundary conditions, including the commonly used Dirichlet conditions $\vec{v} = \vec{v}_d$ which sets the velocity at the boundary of the simulation space.

2.1.2 Simulating Turbulent Fluid Flow

Currently, the state-of-the-art within computational aerodynamics is to model the flow using either Direct Numerical Simulation (DNS), Large Eddy Simulations (LES), or Reynolds-Averaged Navier–Stokes (RANS). In DNS one attempts to solve the Navier–Stokes equations by including all time-dependent fluctuations and in LES one includes most of these fluctuations, whereas in RANS one instead tries to solve the simpler time-averaged equations.

Slotnick et al. 2014 presented NASA's technology development roadmap, which shows that the technology readiness level of RANS for computational aerodynamics is high while for LES this level is low. RANS simulations can therefore be used to generate training data for machine learning methods such as neural networks.

A common method for using CFD to solve turbulent flow is the k-epsilon model. In addition to solving the Navier–Stokes equations, one also solves an equation for turbulent energy, k, and an equation for the dissipation of turbulence, ϵ. In 3D this gives a set of 6 equations: one for each of k, ϵ, and continuity, in addition to the three equations for momentum. A different version of the k-epsilon model is the realizable k-epsilon model, which performs better on more complex structures.

To solve the set of equations, the volume or area of interest is split into smaller pieces. This is called meshing, and for high-resolution urban wind simulations one often require millions of these pieces. Then, the equations and boundary conditions are set up for each piece, and the resulting millions of equations are solved for the entire system.

A different method that is worth mentioning is the Lattice-Boltzmann method, presented by Chen and Doolen 1998. This method is still being developed, but it reduces the simulation time by orders of magnitude. The Lattice-Boltzmann method is inspired by methods used to model molecules, and models the fluid system as a grid where only the nearest neighbours depend on each other. One can thus use parallel computations to quickly calculate the fluid flow.

For simple systems, Lattice-Boltzmann can compute the fluid flow of the system in real-time. For more complex system, like for instance the flow field in urban areas, the method is not fast enough to give real-time results. In addition, the grid-structure means that the method can only model systems with a regular grid with equally sized squares. But even with these limitations, the fast simulation time of the Lattice-Bolztmann method already offers opportunities to create larger amounts of training data which can be used when researching the use of deep learning methods on fluid flow. Some of the work presented in chapter 3 uses the Lattice-Boltzmann method to generate training data.

2.1.3 CFD for Urban Wind Flow

Wind flow in urban areas is turbulent and incompressible, meaning that it is a complex type of fluid flow. CFD is used to simulate wind flow to study how new buildings impact wind conditions in an area, which in turn impacts pedestrian comfort and safety. As explained in Hågbo et al. 2020, an increasing number of city authorities request comprehensive wind studies before granting a building permit.

Running CFD simulations for predicting the wind conditions in an urban area is extremely computationally expensive. The simulation must be run for several wind directions since in the real world, wind will come from several different directions.

If one could interact with wind flow simulations in real-time, these simulations could be used for interactive urban development. Instead of investigating if a building has a low enough impact on the pedestrian comfort to be constructed, one could get immediate feedback while designing buildings and thus move forward accordingly. The finished building designs would then be much more likely to adhere to regulations regarding their impact on the surrounding microclimate. As we will see in chapter 3, deep learning is emerging as a tool for interactive fluid flow predictions.

2.2 Artificial Neural Networks

With observational samples (x_i, y_i), $i = 1, ..., n$ from a CFD simulation, one can train a general function approximator to learn the underlying distribution of the observations. For two-dimensional fluid flow approximation, the observational samples are often images. The input, x, includes information about the geometry one wishes to simulate fluid flow around. The target y depicts the variables describing the fluid flow, such as flow velocity, attained from a CFD simulation.

With an underlying distribution $y = f(x) + \epsilon$, where ϵ is random noise, the goal is to approximate f. The quality of this approximation is measured by a loss function $L(y_i, \hat{f}(x_i))$. The loss function provides a measure between the estimated values $\hat{f}(x_i)$ and the true observed values y_i. Averaging the loss over all observations gives a single metric for the quality of the function approximator. Training such a general function approximator is referred to as supervised learning.

Artificial Neural Networks (NNs) are a type of general function approximators. The universal approximation theory states that any function may be approximated by a sufficiently large and deep network. As fluid flow in a physical system is defined by the Navier–Stokes equations, we can use an appropriate neural network architecture to learn the desired mapping that allows us to directly infer solutions to this nonlinear problem.

A neural network can solve the CFD task as an image-to-image translation task using the UNet architecture. This section therefore describes the architectural elements behind UNet, while a thorough description of UNet is provided in chapter 4. We also present a strategy for improving the generalization abilities of a neural network. The latter is an important principle for our hybrid data-physics model.

2.2.1 A Primer on Neural Networks

A comprehensive treatment of neural networks is far out of scope for this thesis. Instead, we provide a short description of basic neural networks and turn our focus to more complex architectural elements that are relevant for the work done in this thesis. The interested reader may find an in-depth explanation of neural networks by Goodfellow, Bengio, and Courville 2016.

A neural network consists of layers of neurons. The neurons in the first layer are set equal to the input example x_i. Neurons in other layers get their values from linear combinations and simple non-linear functions of the previous layer's values. This way, information is propagated through the network and the values of the last layer is the network's prediction. The neural network has now completed a forward pass. The prediction is then compared to the true value y_i, and the loss function is used to calculate the error of the prediction. The neural network then calculates the error gradient with respect to each parameters, and updates each parameter according to how much it contributed to the error. Minimizing the error this way is called gradient descent, and the whole procedure of calculating the error and performing gradient descent is called backpropagation.

Training a neural network is done by performing forward passes and backpropagation for either a certain amount of times, or *epochs*, or until a stopping criteria is met. This way, the neural network approximates f by finding combinations of parameters that minimize the error given by the loss function.

After training, inference can be done in a matter of seconds with a single forward pass. Training neural networks to learn complex physics problems, which are computationally expensive to solve numerically, therefore provides much faster solutions. The challenges of using neural networks to model physics is the general lack of training data, as neural networks need sufficient amounts of training data to properly tune their weights. In addition, neural networks are black-box models, i.e. we do not have insights in their internal workings.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural networks that are well-suited for processing grid-like data, such as images. Convolutional neural networks use the convolution operation[1] to propagate information from one layer to the next. The convolution operation for a two-dimensional image I and a two-dimensional kernel K is

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n), \tag{2.2}$$

where i, j are indices in the image and in the resulting feature map S. In a CNN, the resulting feature map forms the basis of the next layer. There are some major benefits to using kernels, which we will explain by comparing CNNs to traditional neural networks.

Whereas traditional neural networks have one weight for each pair of neurons in neighbouring layers, a kernel in a CNN has the same value when applied to all positions of the layer. Thus, a kernel is a trainable filter, and it is trained based on all neurons in the layer it is applied to. This sharing of parameters reduces memory requirements, and opens for the possibility of using several kernels for each layer. Doing so allows each kernel for a layer to learn one set of features, e.g. recognizing horizontal edges or vertical edges in an image. CNNs can thus learn spatial relations.

[1]Most libraries implement CNNs with the cross-correlation operation. For deep learning applications, the end-results are the same.

Compared to traditional neural networks, where each neuron in one layer is connected to each neuron in the next layer, CNNs are sparsely connected. In convolutional neural networks, each neuron is only connected to k neurons in the next layer. Thus, CNNs have fewer parameters. In addition, since the neurons in the neighbouring layers are semi-independent of each other, CNNs are suitable for hardware acceleration through parallel computations of the parameters using graphics cards. Training a CNN is thus less time-consuming than training a dense neural network with the same number of parameters.

There are some other benefits of the sparsely connected weight-sharing kernels. CNNs are less prone to overfitting, as each value in the feature map comes from several configurations from the previous layer. In addition, as each layer in a deep neural network can build more complex abstractions by using the output from the previous layer, CNNs can build increasingly complex representations of the input. If the first layers model the edges and curves of an input image, the following layers might be able to model more complex features such as ears or eyes.

In the same way that CNNs are well-suited for two-dimensional input, they can output predictions in two dimensions. When using a CNN to predict fluid flow, one can thus directly output the entire fluid flow field by training the CNN to output an image representation of the flow components v_x, v_y, and p.

2.2.3 Autoencoders

Autoencoders are neural networks that are trained to reconstruct their input. This reconstruction is done so that the autoencoder can learn a salient representation of the dataset's underlying distribution. Autoencoders are therefore used as feature extractors or dimensionality reducers, where they in the latter case were shown to outperform PCA by Hinton and Salakhutdinov 2006.

Autoencoders consists of an encoder and a decoder, where the architecture of the decoder is the mirror image of the encoder. The output layer of the encoder is the input layer to the decoder, and is called the latent space. After training, the latent space provides the feature representation of any unseen data example.

To ensure that the autoencoder does not reconstruct the input by performing the identity mapping, the neural network is restricted so that it must prioritize the most important aspects of the input. One way to restrict the autoencoder is by making it undercomplete, i.e. constraining the layers in the encoder to decrease in size. Such an undercomplete architecture is shown in figure 2.1.

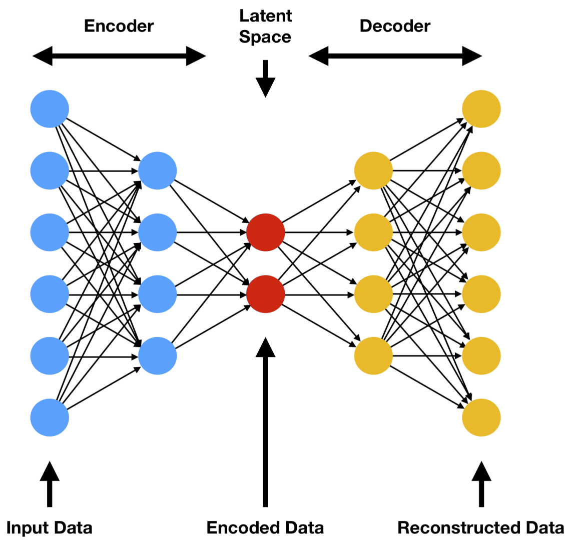

Figure 2.1: The general architecture of an undercomplete autoencoder (adapted from Flores 2019).

The undercomplete autoencoder architecture is often used in neural networks to ensure that the neural network learns the most useful properties of the observations. When used in a general deep learning setting, not only to reconstruct the input data, the neural network is said to have an autoencoder-like architecture or an encoder-decoder structure.

2.2.4 Skip-Connections

Historically, neural networks with many layers and fewer neurons in each layer have performed better than neural networks with fewer layers and many neurons in each layer. The first type is called “deep“ networks, while the latter is called “wide“. For a deep and a wide neural network with the same number of parameters, the deep neural network benefits from being able to build increasingly complex abstractions in each layer. This suggests that a deep and narrow neural network should perform better than a more shallow and narrow neural network. A deep neural network should never have decreased performance compared to a shallow one, as the deep neural network should be able to copy all parameters of a shallow neural network and perform the identity mapping for the rest of it’s layers. However, very deep networks have shown degrading performance on both training and test data, indicating that neural networks struggle to learn the identity mapping.

He et al. 2015 proposed a solution to this degradation problem. Instead of learning the mapping that best describes the data, they learn the residual of the desired mapping and the identity mapping. Learning this residual is done by adding skip-connections between layers. The skip-connections are just the identity mapping and does not add computational complexity or more parameters.

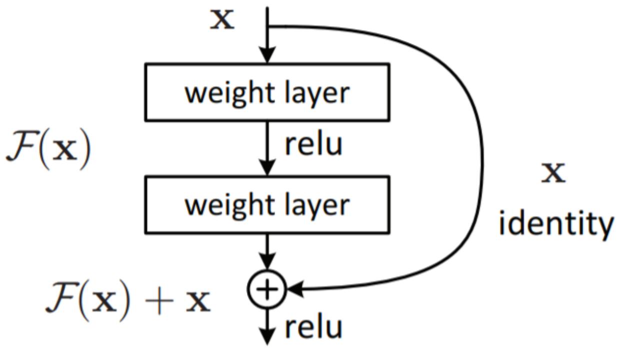

Figure 2.2: A skip-connection used for residual learning in neural networks. With $\mathcal{H}(x)$ as the desired mapping between input and target, learning $\mathcal{F}(x) = \mathcal{H}(x) - x$ is easier for neural networks.

Such a skip-connection can be seen in figure 2.2. The dataset's distribution is described by the unknown function $\mathcal{H}(x)$, which we want the neural network to learn. By instead learning the residual $\mathcal{F}(x) = \mathcal{H}(x) - x$, and adding x to the output of the layers, we obtain the desired $\mathcal{F}(x) + x = \mathcal{H}(x)$. The experiments conducted by He et al. 2015 showed that it is often easier for neural networks to learn the residual. Using skip-connections improved the training performance of deep neural networks, and allowed the use of more layers to increase the accuracy even further.

2.2.5 Regularization

Regularization are strategies that are used to improve a neural network's generalization abilities, i.e. performing better on unseen data. Regularization shift a neural network's priorities, trading a higher training error for a lower test error.

Common regularization methods include dropout, early stopping, parameter norm penalization, and simply using a smaller neural network to reduce the representational power of the neural network. Dropout, proposed by Srivastava et al. 2014, randomly deactivates neurons during training to ensure that the neural network's parameters does not co-adapt too much. Early stopping is when the training is stopped before all epochs are concluded, e.g. if the validation loss is non-decreasing. Increasing validation loss is a sign that the neural network has overfitted on the training data. Parameters norm penalization adds a term to the loss function that penalizes parameters with large absolute-values, as large absolute-values often is a sign of overfitting. Adding such a regularizing term to the loss function has inspired the use of scientific equations in the loss function of neural networks.

2.3 Integrating Deep Learning and Physics

Traditionally, domain knowledge has been used to feature engineer input to, and post-process results from, machine learning models. In recent years, integrating scientific knowledge more closely with neural networks has emerged as a promising scientific endeavour. Neural networks do not need feature engineering in the same way that other machine learning methods do, and they can also directly output the desired targets. Therefore, the new integration of neural networks and scientific knowledge differs from the traditional one. This section shed light on this new coupling of physics and deep learning, with a focus on solving partial differential equations.

2.3.1 Data-Driven Neural Networks for CFD

As explained in section 2.2, neural networks can be trained to predict fluid flow by using a dataset containing samples from a CFD simulation. We refer to this approach, using a dataset to train a neural network, as *data-driven* training.

Willard et al. 2021 describe nine ways in which machine learning and physics can be coupled, one of which is using data-driven neural networks to solve PDEs. In systems where the governing equations are known, but traditional numerical solutions are computationally expensive, neural networks can be used as surrogate models for the PDE solver. For fluid systems, this means that neural networks can be used to solve the Navier–Stokes equations. Even though both generating a dataset using CFD and training the neural network are computationally expensive processes, inference using the fully trained neural network is fast. Neural networks can thus output approximate solutions to PDEs in a manner of seconds. In addition to computational speed up, the results obtained from neural networks are differentiable and can be used in subsequent calculations.

The drawback of such data-driven neural networks is the lack of interpretability and trustworthiness from unawareness of physical laws. Training the neural network may also be an issue, both in terms of training time and generating training data. Lastly, another challenge with data-driven neural networks is overfitting. With scarce amounts of training data, overfitting is more likely to occur.

2.3.2 Physics-Informed Neural Networks

To reduce the drawbacks of the data-driven method described above, one can take one step further to incorporate physics in a neural network. Instead of just using neural networks to solve PDEs by learning from observational samples, Raissi et al. 2017 showed that one can embed knowledge of the physical laws that describe the system into the neural network. Such knowledge pertains to both the conservation laws of the system, as well as boundary conditions.

Formal Definition of a PINN

Neural networks can be used to solve a general nonlinear PDE

$$u_t + N[u] = 0 \tag{2.3}$$

with the solution $u(t, x)$. Here, $N[:]$ is a nonlinear operator, t is the timestep, and $x \in \Omega$. For three dimensions, Ω is a subset of $\mathbb{R}^3$. By using the neural network to approximate $u(t, x)$ and defining

$$f := u_t + N[u] = 0, \tag{2.4}$$

$f(t, x)$ is a PINN. In section 4.2.1, we derive our physics-informed loss term by explicitly connecting this definition with equation 2.1a.

To train the PINN, a loss function with both a data-driven and a physics-driven term is minimized:

$$\mathcal{L}_{\text{PINN}} = \mathcal{L}_{\text{DD}}(u(t, x), \hat{u}(t, x)) + \mathcal{L}_{\text{Physics}}(\hat{u}(t, x)). \tag{2.5}$$

Here, $\mathcal{L}_{\text{DD}}$ is a data-driven loss function[2], e.g. L1, which takes as input the target values $u(t, x)$ and the output from the model $\hat{u}(t, x)$. $\mathcal{L}_{\text{Physics}}$ is the residual function modeling $f(t, x)$, requiring the PDEs of the physics system to be satisfied by the output of the neural network.

Incorporating a physics-based loss term changes the topography of the solution space, reducing the part of this space that contains viable solutions to the task at hand. This reduced number of viable solutions may simplify the search for the model, and therefore might lead to higher predictive performance due

[2]In deep learning libraries, many loss functions are implemented. For a list of PyTorch's loss functions, see `https://pytorch.org/docs/stable/nn.html#loss-functions`.

to convergence to a global optima instead of a local optima. As we will see in section 3.3, it is also possible for the model to only use $\mathcal{L}_{\text{Physics}}$ to learn, removing the need for computationally expensive target values. Lastly, when a neural network follows the governing physical equations of a system, it is more likely to have strong generalization capabilities.

2.3.3 Evaluating Physics-Informed Neural Networks

There are three general computational objectives one should consider when evaluating a physics-informed neural network:

- First of all, the predictive performance of the neural network should improve.
- Secondly, one must consider sample efficiency. One could either reduce the number of training examples required for adequate performance, or improve the quality of the samples so the overall search space for the optimization algorithm is smaller.
- Lastly, interpretability of the model must be considered. Designing a model that is physically consistent ensures that the governing equations of the physical system are obeyed.

Chapter 3

State of the Art

This chapter explores state-of-the-art research on the use of neural networks to perform fluid flow simulations. First, research regarding data-driven models for fluid flow is presented in section 3.1. Then, in section 3.2, we present research on hybrid models such as PINNs. Lastly, unsupervised physics-driven models using only physical knowledge are presented in section 3.3. Within all three sections, the work is presented in chronological order to convey an understanding of how the integration of fluid dynamics and deep learning has emerged.

3.1 Data-Driven Neural Networks for Fluid Flow

From section 2.2, we have seen that neural networks are trained using a set of observations, more commonly referred to as a dataset of input-target pairs. This data-driven approach has been used for fluid flow prediction, due to the fast inference time and impressive predictive capabilities of neural networks.

Thuerey et al. 2018 used a UNet architecture to approximate the RANS solutions of flow around an airfoil in 2D. By using a convolutional neural network architecture, they are able to embed spatial information into latent space. As input to the model, they used freestream velocities in the x- and y-direction and a mask encoding the geometry. They argued that the latter was redudant, as the freestream velocities were already set to zero at the pixel location of the geometry. The network learned to predict velocity in the x- and y-directions, v_x and v_y, as well as the pressure, p, around airfoils by using images with the simulated values of v_x, v_y, and p as targets. The ground truth images were created by CFD simulations. The model was purely data-driven, using only the difference between model output and the target images to train.

Thuerey et al. 2018 showed how the accuracy of the predictions were influenced by the amount of training examples, ranging from 100 to 12800 examples, and the size of the model. The size of the model was determined by the number of trainable parameters in the model, while the number of layers were the same across models. Five different models were used, where the number of weights ranged from 122k to 30.9M.

For smaller amounts of training data, the smaller models performed better while the larger ones were prone to overfitting. Thuerey et al. 2018 demonstrated that the regularization stemming from the smaller number of weights led to better generalization capabilities. When a larger amount of data was made available, the larger models were able to generalize better than the smaller models. All models exhibited signs of stagnating validation loss, motivating the need to investigate other approaches to the prediction of the fluid flow.

In addition to their investigation on the influence of training data size and the number of trainable parameters, Thuerey et al. 2018 also illustrated the importance of data normalization and using dimensionless quantities. Normalizing the pressure and the velocity substantially reduced the average absolute error in their experiments.

Musil et al. 2019 used the UNet architecture with 3D convolutions to predict the steady-state turbulent wind flow of an urban area. They generated a dataset consisting of 3500 samples representing different sets of buildings with heights, widths, and depths that are common in real cities. Their main focus was on developing an interactive tool for wind flow prediction that can be used in urban development.

Their model showed good generalization abilities, and with the added complexity of three dimensions the neural network was three orders of magnitude faster than a CFD solver – operating almost in real-time. In addition to the move to 3D, Musil et al. 2019 also trained the model on the reverse workflow – predicting building volumes for a desired wind flow.

A different architectural approach was done by Pfaff et al. 2020, who developed a mesh-based model to predict a wide range of physical systems, including aerodynamics, structural mechanics, and cloth. Their model is able to predict a long sequence of a physical system's behaviour with only a short sequence as initialization. In addition, the model handles generalization extremely well, predicting fluid dynamics of more complex shapes than what it had been trained on. The impressive generalization was due to the use of relative encoding on graphs, meaning that positional information was invariant to absolute spatial location in the system.

The architecture of the model was based on Sanchez-Gonzalez et al. 2020, who showed that an encode-process-decode architecture performs fluid simulations well. Each of the steps – encoding, processing, and decoding – consisted of a graph, allowing the representation of simulation meshes. In addition to training the model to predict the simulated system's future dynamics, Pfaff et al. 2020 also had the model learn the mesh discretization so that the mesh could be changed at inference. For simulations of Lagrangian systems, i.e. systems where the simulation nodes may move with the velocity field, the world-space is encoded in addition to the mesh-space. In the same way that the mesh-space encodes the simulation mesh, the world-space encodes the dynamic space of the mesh in 3D space. The world space thus allows for explicit information about interactions, such as contact and collision.

Using meshes to represent the physical system is, as CNNs, suitable for hardware acceleration which leads to a large improvement in runtime compared to physical simulations. Pfaff et al. 2020 outperformed the UNet architecture implemented in Thuerey et al. 2018[1]. For all experiments, a training dataset of 1000 samples, and validation and test datasets of 100 samples each was used. Each data-sample contained 600 training steps.

Lastly, Høiness et al. 2021 evaluated two state-of-the-art Generative Adversarial Network (GAN) architectures and a UNet architecture on the data-driven prediction of the velocity field around building geometries. While their main focus was on the assessment of GANs for predicting urban wind flow, their results showed that the UNet was able to outperform the two GANs on almost all tasks. This strongly motivates our use of the UNet architecture in our experiments.

Where Musil et al. 2019 used a neural network supporting three-dimensional data, Høiness et al. 2021 took a different approach. Their datasets was generated by simulating the entire 3D velocity field, and then extracting the velocity magnitude in a slice two meters above the ground. Thus, they could transform the three-dimensional velocity magnitude at the pedestrian level to an image-to-image translation task. For our experiments, this simplified solution is not feasible. Injecting the governing equations of a fluid system into a neural network requires information regarding the changes in pressure and velocity for all dimensions.

3.2 Hybrid Neural Networks for Fluid Flow

As mentioned in section 2.3.2, using both physics and data in neural networks to solve nonlinear PDEs gained popularity with the framework formalizing the

[1]It is worth remembering that this UNet architecture was not made for predicting long sequences.

PINN developed by Raissi et al. 2017. In addition to further research on such PINNs, we also present the work of another research group with a completely different approach to injecting physics into a data-driven neural network.

Raissi et al. 2019 expanded on the research presented in Raissi et al. 2017 with additional examples. For the data-driven discovery of PDEs, one of the additional examples Raissi et al. 2019 use to demonstrate their framework's capacity for solving PDEs is the Navier–Stokes equations. A dense neural network was trained with the Navier–Stokes equations and boundary conditions in the loss function. The network learned to output the system's velocity in x- and y-direction, as well as the pressure p, from spatio-temporal input x, y, and t. Using 1% of the total simulated data in a two-dimensional time-dependent system, the network's ability to learn from sparsely available data was demonstrated.

Using the framework from Raissi et al. 2019, Cheng and Zhang 2021 developed Res-PINN – a physics-informed dense neural network that incorporates skip-connections. When demonstrating the framework's capacities, solving the Navier–Stokes equations was once again used as one of the examples. The relevant equations and boundary conditions were used in the loss function. Compared to a physics-informed neural network without Resnet blocks, Res-PINN has stronger predictive abilities. This strongly suggest that using skip-connections improves the performance on neural networks on the fluid flow task, which motivates our use of UNet.

Wang et al. 2020 developed a complex neural network, TF-Net, where a specialized UNet architecture is inspired by hybrid LES-RANS CFD methods. TF-Net focus on multi-step prediction of the future turbulent flow field of a system, given an initial flow field. The network consist of two main parts: decomposing of the velocity field, inspired by hybrid RANS-LES coupling, and a UNet based encoder-decoder scheme.

The velocity field is decomposed into three parts: the time-averaged mean flow $\bar{w}$, resolved fluctuations w', and a spatially filtered variable $\tilde{w}$. The first part stems from RANS modelling, and the third part from LES modelling. The second part is present in both RANS and LES. The hybrid CFD approach, using the velocity decomposition from both RANS and LES, provides better resolving power than RANS while being less computationally expensive than LES. These decompositions are created with a spatial filter and a temporal filter. In traditional CFD, these two filters are pre-defined but Wang et al. 2020 implemented both filters with neural networks.

The UNet based architecture used in TF-Net has a separate-encoder-shared-decoder scheme, i.e. there are three encoders that contribute to the latent representation which is the input to one decoder. After decomposing the velocity field, as described in the previous paragraph, each decomposed part is passed to their own encoder. Then, the latent vectors from each encoder is combined into the shared decoder which generates the final prediction of the future flow field.

Like the methods mentioned in section 3.1, TF-Net is also data-driven. The reason why we consider this work relevant for research into PINNs is that a model with an explicit physical constraint, Con TF-Net, was evaluated by Wang et al. 2020. Con TF-Net includes a regularizing loss term, modeling the continuity equation. This PINN showed improved predictive performance. However, they found that there was a trade-off between their data-driven metric and their physics-driven metric. Better performance on the data-driven metric led to a lower performance on the physics-driven metric, and vice versa. In contrast to Con TF-Net, we use a UNet that is not engineered specifically towards solving fluid flow as our neural network architecture. Our goal is to investigate if the classic principles of deep learning in combination with the restrictions provided by the physical laws can provide an elegant solution to turbulent flow prediction.

Pawar et al. 2020 takes a different approach in combining deep learning and physics. Instead of incorporating scientific equations in the loss function, they embed physical flow-parameters and predictions from a simpler simulation method into a hidden layer in the neural network.

Their method is validated by predicting the lift coefficient around an airfoil, and comparing the predictions to those from a neural network without additional physics injected. Both network's predictions are compared to the true value of the lift coefficient, obtained from a simulation. The Physics-Guided Neural Network (PGNN) is both more accurate and has a smaller uncertainty than the black-box neural network. The uncertainty of the neural networks is calculated by training many models with different random seeds.

A benefit with our approach – a physics-informed loss term – is that PINNs are rewarded for generating predictions that adhere to physical equations. A PGNN, on the other hand, is purely trained using a data-driven loss function. Therefore, PGNNs decide for themselves how much weight to give the physical knowledge. Our goal is to approximate values from simulated wind flow. These values are perfectly described by physical equations, which means that using a PINN ensures that the physical knowledge is obeyed.

3.3 Unsupervised PINNs

With the emergence of PINNs, some researchers have trained neural networks using only the physics-informed loss term. We classify these neural networks as *unsupervised* PINNs, as they do not need targets from CFD simulations. The downside to this approach is that the networks are slow to converge in training due the large optimization space and the few constraints offered by the physical equations.

Wandel et al. 2020 use such a purely physics-constrained loss, incorporating the UNet architecture. But instead of predicting the pressure p and velocity $\vec{v}$, the network is trained to learn pressure p and the vector potential $\vec{a}$ of a system. Predicting the vector potential for which $\vec{v} = \nabla \times \vec{a}$, automatically fulfills the continuity equation as the divergence of a curl is always zero, i.e.

$$\nabla \cdot \vec{v} = \nabla \cdot (\nabla \times \vec{a}) = 0. \tag{3.1}$$

To force the network to obey physical laws, a loss function that enforces both the conservation of momentum and the Dirichlet boundary condition is used. The loss term enforcing the Dirichlet boundary condition had to be given a relatively large weight to reduce leakage through boundaries.

Using a set of domain geometries, boundary velocities, and initial conditions for the vector potential and the pressure fields, a set of features are calculated and fed as their own separate channel to a UNet architecture. The neural network outputs two channels, predicting the vector potential and the pressure field for the next timestep.

Training data was subsequently updated by replacing the old vector potential and pressure fields with the newly predicted ones, continuously providing better training data for the model. Wandel et al. 2020 argued that re-using geometries and boundary velocities with new vector potentials and pressure fields is one of the reasons why the model is able to generalize so well to unseen shapes during inference. The second reason is because the model is spatially local, allowing it to learn basic geometric shapes and use this knowledge to predict more complex shapes.

For comparison, Wandel et al. 2020 also trained a model that directly predicted the velocity field $\vec{v}$. Here, they had to include a loss term that enforced the continuity equation. When training this second model, the loss term modelling the continuity equation had to be given a high weight to avoid sources and sinks appearing. The Dirichlet boundary condition was simple to learn for this model, so it could be given a lower weight. Qualitatively, the model that predicted the vector potential $\vec{a}$ showed better results than the model predicting the flow field $\vec{v}$ directly, even though a quantitative analysis showed similar errors.

Moving back to the a less complex UNet architecture, Ma et al. 2021 adapts the UNet architecture used by Thuerey et al. 2018. They use this UNet for physics-driven training for the prediction of steady-state laminar flow. Laminar flow is in the opposite of the spectrum compared to turbulent flow, and is smooth and predictable. The input used by Ma et al. 2021 has the dimensions of an image with four channels, where initial velocities u_0 and v_0, information about the object's geometry G, and the Reynolds number Re each have their own channel.

Their physics-driven CNN, PD-CNN, used the approach of a physics-informed loss term with the discretized Navier–Stokes equations and boundary conditions in the loss function. PD-CNN predicts velocities and pressure as images, outputting the entire flow field.

Ma et al. 2021 also showed how the training of the neural network could be accelerated with the inclusion of a data-driven loss term, turning it into a PINN. While their assessment of this PINN was brief, they showed that using both the data-driven and the physics-driven loss term accelerated the network training. However, laminar flow is, as mentioned in section 2.1, a lot simpler than turbulent flow. Our work attempts to investigate if using a PINN on turbulent flow leads to similar results, which in turn could motivate the adaption of the strategies from Ma et al. 2021 to the complex turbulent domain.

Chapter 4

Methodology

In chapter 3, we saw how deep learning and CFD have been coupled. We presented research on using neural networks to predict laminar and turbulent flow, steady-state prediction and the prediction of sequences. While this provides a wider understanding of the integration of CFD and deep learning, our focus is on the task of approximating steady-state turbulent flow using the methodology presented in this chapter.

We will use the UNet architecture to predict the steady-state wind velocity around building geometries. From chapter 3, we know that translating the CFD task to an image-to-image translation problem and solving this problem with UNet has been highly successful. By using the UNet architecture and incorporating physics knowledge, we investigate the effects of combining deep learning with traditional physics. In section 4.1 and section 4.2 we describe the UNet architecture used and how the physical knowledge has been incorporated, respectively.

As our research questions requires the training of several neural networks, we reduce the three dimensional wind flow prediction problem to a two dimensional one as done in several of the works presented in chapter 3. This allows us to generate substantially more training data with numerical CFD methods while still evaluating the impact of added physics knowledge to a neural network. Using a two dimensional approximation is the equivalent of modelling the airflow around infinitely high buildings.

4.1 Neural Network Architecture

We are looking to develop a physics-informed model that is able to approximate a CFD task more efficiently than a comparable data-driven model. To ensure that these two models are comparable, the physics-informed neural network uses the same architecture as the data-driven neural network, only changing elements which are strictly necessary to incorporate physics.

The UNet architecture developed by Thuerey et al. 2018 is used, as this has shown promising results both as a data-driven model and when adapted to a purely physics-driven model. We present as input to the UNet an image representation of the building geometries, and train the model to output the velocity-components in the x- and y-direction. The desired mapping from geometries to velocity-components is presented in Figure 4.1.

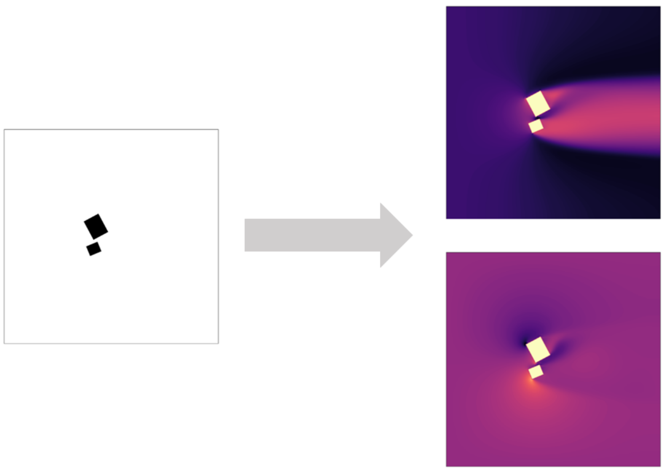

Figure 4.1: Mapping from input geometries to target velocity components v_x (top) and v_y (bottom).

Originally, Ronneberger et al. 2015 developed the UNet for biomedical image segmentation. The model has later been used in a number of image-to-image-translation tasks, due to its high performance on image data from combining skip-connections and a convolutional autoencoder architecture.

The autoencoder architecture of UNet has a contracting part and an expanding part. There are skip-connections between each corresponding level of the contracting part and the expanding part, allowing the network to learn residuals. The UNet architecture used is shown in Figure 4.2. The input has one channel of size $H \times W$, representing building geometry. The output from the model is $H \times W \times 2$, one channel each for the x- and y-component of the velocity.

Each level, or convolutional block, consists of an activation function, a convolution operation, and batch normalization. Batch normalization, without dropout, was shown by Ioffe and Szegedy 2015 to work as a regularization layer in convolutional neural networks. The encoder-blocks have Leaky ReLU as activation functions, and either 4×4 2D convolutions or 1×1 2D convolutions. The last layer of the encoder does not have batch normalization. The decoder blocks have ReLU as the activation function, while the convolutional operation consists of an upsampling and either 3×3 convolutions or 1×1 convolutions.

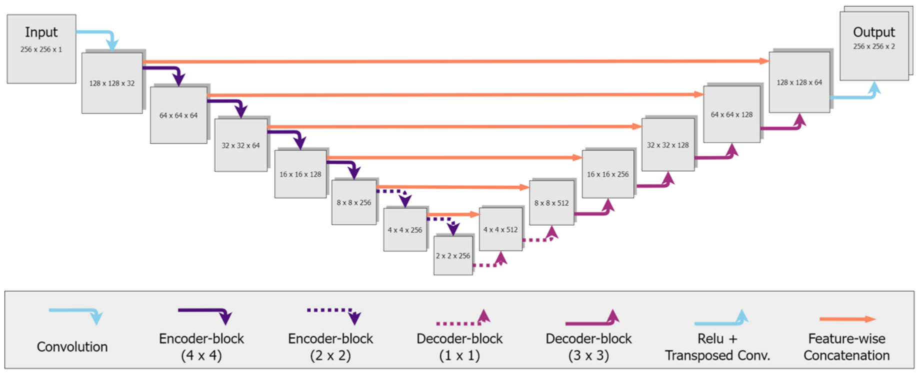

Figure 4.2: The UNet architecture transforms the input image, and then consists of a series of encoder-blocks and decoder-blocks connected by skip-connections. The output from the last decoder-block is transformed to have 2 output channels, one for the x-component of the velocity and one for the y-component.

For each layer in the encoder, the height and width of the image is halved. The transformation of the input image results in 32 channels, and the number of channels are increased by a factor of two by the first, third, and fourth layer of the encoder.

In the decoder, the height and width of the image is doubled. Here, the number of channels are decreased by a factor of two such that the decoder and encoder are symmetric. However, the skip-connections between the encoder and the decoder results in twice as many channels in the decoder as in the encoder.

Using 32 channels for the transformation of the input image results in a UNet architecture with $2,329,986$ parameters. This relatively small network size was chosen due to the relatively small size of our datasets, to ensure that the network is less prone to overfitting.

4.2 Physics-Informed Loss Functions

Inspired by work done with PINNs, as seen in section 3.2, we integrate the continuity equation into our loss function and name the resulting model VD-UNet (velocity-divergence-UNet). Additionally, we test how a simpler loss term, requiring very little knowledge of the physical laws describing the system, impacts the predictions from a UNet model. We name this model GM-UNet (geometry-masked-UNet). Our baseline is the UNet without any physics-informed loss terms. Our data-driven loss term is the L1, as Thuerey et al. 2018 showed that this metric improved accuracy compared with other metrics. The two PINNs and the baseline are summarized in Table 4.1. All loss terms are given equal weight.

Table 4.1: Neural networks considered in our experiments.

Neural network name	Loss terms
UNet (baseline)	L1
VD-UNet	L1 and Velocity Divergence
GM-UNet	L1 and Geometry Masking

4.2.1 The Continuity Equation as a Loss Term

We model the first of the Navier–Stokes equations, the continuity equation, in our VD-UNet through an additional loss term. Using the formal definition of a PINN from section 2.3.2, we approximate the solution $u = v(x, y)$ with a neural network by requiring

$$f = \nabla \cdot \vec{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} = 0.$$

With $\left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}\right)_i$ as the velocity divergence at pixel i and N as the total number of pixels, the loss term to minimize for an image-representation of the fluid flow field is

$$\mathcal{L}_{\text{Physics}}(\hat{v}) = \frac{1}{N}\sum_{i}^{N}\left(\frac{\partial \hat{v}_x}{\partial x} + \frac{\partial \hat{v}_y}{\partial y}\right)_i . \tag{4.1}$$

As calculating the continuity divergence requires predictions of the velocity's x- and y-components, $\hat{v}_x$ and $\hat{v}_y$, we require the neural network to output each component as a separate channel.

A benefit of modeling the continuity equation, as opposed to the momentum conservation equation, is that we can inject a physical law into the neural network without knowing the pressure of the system. For urban wind flow prediction, the pressure itself is not necessarily of interest.

In equation 4.1, our loss-term is independent of the target images. We have seen the use of such loss terms in the works of Ma et al. 2021, as well as by Wandel et al. 2020. However, in our research we observed that the velocity divergence calculated for the 256×256 target images did not obey the continuity equation for the entire image. This effect is most noticeable close to the building geometries. This non-zero velocity divergence is shown in Figure 4.3, and is due to the rapid changes in velocity near the building geometries which the image resolution is unable to display. Therefore, our framework calculates the velocity divergence from the target images, and compares this to the divergence of the predicted velocity from VD-UNet:

$$\mathcal{L}_{\text{Physics}}(v, \hat{v}) = \frac{1}{N} \sum_{i}^{N} \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} - \left(\frac{\partial \hat{v}_x}{\partial x} + \frac{\partial \hat{v}_y}{\partial y} \right) \right)_i . \tag{4.2}$$

Here, v_x and v_y are the target velocity components while $\hat{v}_x$ and $\hat{v}_y$ are the predicted velocity components. We approximate the derivatives numerically, using PyTorch's gradient-method. The architecture of VD-UNet with the physics-guided loss term is shown in Figure 4.4.

4.2.2 Adding a Constraining Mask to the Geometries

A simple constraint for the prediction of wind flow is the following: Inside a building, the velocity should be zero. We impose a hard constraint on the loss of GM-UNet to ensure that the neural network does not try to predict the velocity inside a building, and instead we manually set this velocity to zero. Removing the need for the neural network to predict the velocity at these pixels should hopefully allow it to instead focus on other, more interesting areas of the image. Our loss term for this geometry masking is

$$\mathcal{L}_{\text{GM}} = \frac{1}{M} \sum_{j=1}^{M} \left| \hat{y}_j - y_j \right| , \tag{4.3}$$

where j is an image pixel index corresponding to a pixel which is not part of a building geometry, and M is the total number of pixels that are not part of a building geometry.

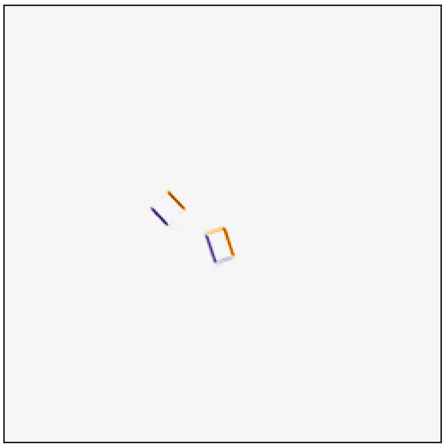

Figure 4.3: The velocity divergence is non-zero close to the building geometries. The image resolution is unable to capture the rapid changes in velocity in this area.

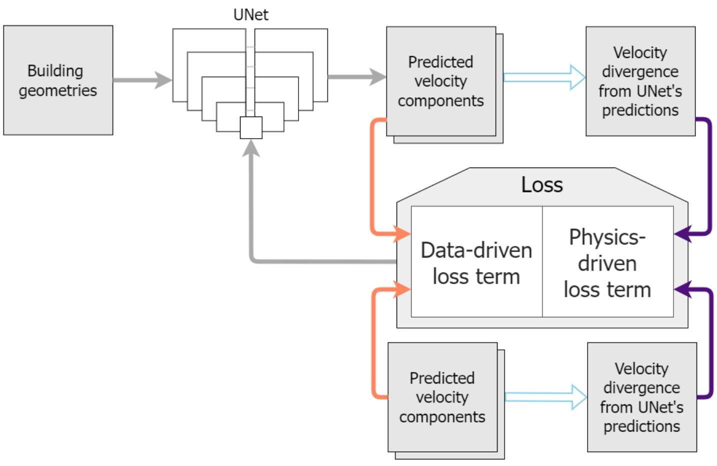
UNet
Building geometries
Predicted velocity components
Velocity divergence from UNet's predictions
Loss
Data-driven loss term
Physics-driven loss term
Predicted velocity components
Velocity divergence from UNet's predictions

Figure 4.4: The PINN with the velocity divergence as a loss term. The physics-guided loss term compares how well the output from the neural network obeys the continuity equation.

Chapter 5

Experiments

In this chapter, the practical aspects of our experiments are presented. We begin with a description of the datasets used, along with the preprocessing applied to them. The two subsequent sections present the evaluation metrics used to compare our model with the baseline, as well as our experimental plan. Finally, technical details regarding the implementation of the framework is provided.

5.1 Datasets

The experiments were conducted using three data sources, each of which contains images depicting building geometries and the corresponding velocity components. For each of the three datasets $\mathcal{D}_1$ (2 buildings), $\mathcal{D}_2$ (4 buildings), and $\mathcal{D}_3$ (6 buildings), RANS CFD simulations were used to create $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ with $n = 1000$ pairs of input and target images. The image generation process is described in Appendix A. The input images, x_i, display building geometries while the target images, y_i contain the steady-state velocity components around the geometries.

Due to non-convergent simulations[1], some of the dataset examples had to be removed. For $\mathcal{D}_1$, 3 images had non-convergent velocity fields and thus the resulting dataset contains $n = 997$ training examples. For the same reason, $\mathcal{D}_2$ and $\mathcal{D}_3$ contains 841 and 845 examples, respectively. We hold out 100 samples for our validation set and 100 samples for our test set, resulting in maximum training dataset sizes of 797, 641, and 645 for $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$, respectively.

Each input image x_i is an element of $\mathbb{R}^{H \times W \times 1}$, where $H = W = 1000$. Each dataset $\mathcal{D}_j$ contains $B = 2 \times j$ square geometries representing buildings. The

[1]Some of the CFD simulations diverged, resulting in target images consisting purely of noise.

geometries have different offsets $(\Delta x, \Delta y)$ from the center, and were allowed to overlap. The overlapping results in some training examples with less than B geometries. Some of these images, containing fewer distinct buildings, have more complex geometries due to semi-overlapping buildings. Overall, $\mathcal{D}_1$ is the least complex dataset, followed by $\mathcal{D}_2$, and then $\mathcal{D}_3$ is the most complex.

Each target image y_i is an element of $\mathbb{R}^{H \times W \times 2}$, with the same H and W as the input images. These target images depict the simulated velocity components resulting from an inlet wind of $1\,\mathrm{ms}^{-1}$ at the left edge of the image. The first channel contains the x-component of the velocity, while the second channel contains the y-component of the velocity.

In Figure 5.1, examples of input images for all three datasets are shown in the top row. The middle row shows the corresponding velocity components in the x-direction, and in the bottom row we see the corresponding velocity components in the y-direction.

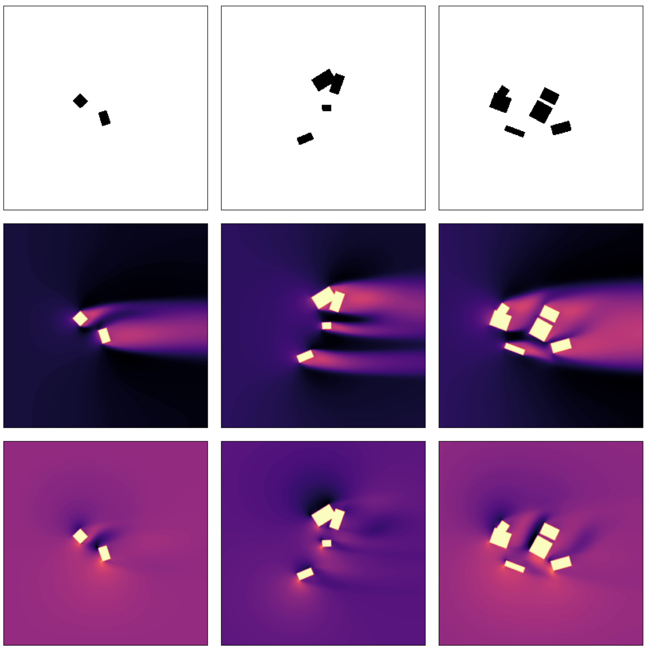
Figure 5.1: Examples of x_i (top) and the first and second channel of y_i (middle & bottom) for $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$. In the examples from $\mathcal{D}_2$ and $\mathcal{D}_3$, we see that the geometries are semi-overlapping which leads to more complex building geometries.

5.1.1 Preprocessing

All images were resized to 256×256 pixels using Torchvision transform's Resize class[2]. Note that previous research has mainly considered images with 128×128 pixels, such as the work of Thuerey et al. 2018. The decision to not reduce the image sizes all the way to 128×128 pixels was to include smaller changes in the velocity components, ensuring that the velocity divergence fulfilled the continuity equation to a greater extent.

Resizing the images lead to x_i no longer having completely binary pixel-values. Therefore, all input images were binarized using 0.5 as the threshold value for a pixel being set to 0 or 1. The target images were normalized to have values from 0 to 1.

5.2 Evaluation

The overall goal of our experiments is to evaluate whether or not PINNs are suitable models for wind flow prediction in urban areas. As mentioned in section 2.3.3, there are three computational objectives to consider when evaluating the performance of a PINN: predictive performance, sample efficiency, and interpretability. We therefore evaluate our PINN with these three criteria.

The three research questions allow us to investigate the computational objectives. The research questions are repeated here for convenience:

1. How does the addition of physics knowledge affect a neural network's training process?
2. How does the PINN's performance differ as the amount of training data changes?
3. How well can the PINN generalize to a more complex data distribution?

We use one metric for the predictive performance and another for interpretability, both of which will be described in detail shortly. Sample efficiency is evaluated differently for the three research questions, in a more qualitative way.

Research question 1 is formulated to evalute the predictive performance and interpretability of the PINN by investigating its stability and convergence during training. Evaluating the training performance allows us to assess how well the PINN is able to learn the mapping function describing the dataset. For this research question, we evaluate sample efficiency by analysing how well the PINN utilize the training data compared to the UNet.

[2]`https://pytorch.org/vision/stable/transforms.html#torchvision.transforms.Resize`

For research question 2, we evaluate the models across a range of training dataset sizes. This tells us both how several PINNs and UNets perform, in terms of predictive performance and interpretability, and how well they are able to utilize samples when the amount of available samples differs.

Research question 3 is related to the evaluation of the predictive performance and interpretability for the more challenging case of generalizing to an unseen data distribution. If the PINN exhibits improved predictive performance or interpretability, compared to the UNet, this also suggests that the PINN is able to utilize the samples better than UNet.

5.2.1 Quantitative Evaluation Metrics

To compare the performance of the neural networks to one another, two quantitative metrics are used. The first measures accuracy of the wind flow prediction, while the second measures the extent to which the neural networks' predictions satisfy the continuity equation.

Each of our three datasets are split into a training set, a validation set, and a test set. The training set is, as the name suggests, used to train each neural network. The validation set and the test set contain samples that the neural networks have not seen during training, and thus test how well the neural networks generalize to unseen data samples. The validation set is used to evalute the neural network's performance during training, while the test set is used to evaluate the neural network's performance after training.

Mean Absolute Error

The Mean Absolute Error (MAE), also called the L1 loss, measures the average absolute distance between a prediction from the neural network and the corresponding target values:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i| \,. \tag{5.1}$$

Here, i is an image pixel index and N is the total number of pixels. The MAE thus results in one number for each image, and is a metric of accuracy. In the rest of the thesis, we refer to the MAE as the L1, to avoid additional terminology. Since L1 is used as our data-driven loss function, the neural networks are trained to minimize it and it is thus the natural choice for our evaluation.

We thus use the L1 as an evaluation metric in two ways:

- During training we measure the *average* L1 on the validation sets to assess training stability and convergence.

- After training the neural networks, we look at the *distributions* of the L1 on the test sets to evaluate predictive capabilities.

Velocity Divergence

While L1 is the indicator for the accuracy of the neural networks, the velocity divergence (VD) is included as a way to evaluate the neural networks' interpretability. This evaluation tells us if there is more potential for developing deep learning models that we can trust for predicting physics.

The continuity equation (equation 2.1a) requires the velocity divergence to be zero. As explained in section 4.2.1, the continuity equation does not always hold for image-representations of the flow field, depending on the resolution of the image. Our metric is given by numerical approximation[3] of equation 4.2, which to recap is:

$$\mathcal{L}_{\text{Physics}}(y_{\text{target}}, y_{\text{pred}}) = \frac{1}{N}\sum_{i}^{N}\left|\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} - \left(\frac{\partial \hat{v}_x}{\partial x} + \frac{\partial \hat{v}_y}{\partial y}\right)\right|_i .$$

We use the velocity divergence in the same two ways as the L1, looking at either the average over the validation set or the distributions of the test set.

5.2.2 Qualitative Evaluation Metrics

As additional evaluation of the predictive performance of the neural networks, the predictions are compared visually. For the x-component of the velocity, we compare the output from the neural networks to the target images. In addition, we display the absolute pixel-wise errors for the predictions of this component, to provide further understanding of the neural networks performance for different spatial locations in the image. We chose to focus on the x-component, as the inlet wind in the CFD simulations is parallel to the x-axis, thereby making v_x the most interesting flow component. To illustrate the velocity field, we display $\vec{v}$, were $|\vec{v}| = \sqrt{v_x^2 + v_y^2}$. The velocity field provides a complete comparison of the predictive performance of the neural networks, as well as implicitly conveying the y-component of the velocity, v_y.

5.3 Experiment Plan

To investigate our hypotheses, as presented in section 1.2, the experiments are structured into two main parts. The first experiment allows us to determine

[3]With PyTorch's gradient-method.

which physics-informed loss terms to use in the second experiment. The second experiment is structured to investigate our research questions.

5.3.1 Experiment 1: Evaluating Physics-Informed Losses

We conduct an experiment to determine if the geometry masking loss term, incorporated in GM-UNet, should be included in Experiment 2. To learn how each physics-informed loss term affects the performance of the UNet architecture, we compare the test loss distributions for the three types of models – baseline UNet, VD-UNet, and GM-UNet. All three types of models are trained on a range of dataset sizes using $\mathcal{D}_1$, i.e. the dataset where only two square geometries are included.

The training dataset sizes are increased exponentially, from 10, to 100, to the maximum of 797 samples. The exponential increase in dataset size lets us analyze the model's behaviour for low-resource regions while keeping the number of models trained to a minimum.

5.3.2 Experiment 2: Evaluating the Final PINN

The final PINN used in this experiment includes the loss term modelling the continuity equation. Depending on the results from Experiment 1 we may also include the geometry masking loss term. Thus, this PINN is either identical to VD-UNet, or it is a combination of both VD-UNet and GM-UNet. Regardless, we name our final PINN Ph-UNet (Physics-UNet). To investigate the performance of the models for several fluid systems complexities, we conduct our experiment on all three datasets, i.e. $\mathcal{D}_1$ with only 2 square geometries in the flow field, $\mathcal{D}_2$ with 4 square geometries and $\mathcal{D}_3$ with 6 square geometries. As mentioned in section 5.1, these geometries may overlap.

Both types of models, the UNet and the final Ph-UNet, are trained on a larger number of training dataset sizes than in Experiment 1. The increase in the number of training dataset sizes is to allow for an in-depth investigation of the performance of the Ph-UNet. For $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$, we trained 8, 7 and 7 Ph-UNets, respectively, in addition to the corresponding baseline UNets. The number of samples used for $\mathcal{D}_1$ ranged from 97 to 797, increasing in steps of 100 samples. The same step sizes of 100 samples was used for $\mathcal{D}_2$ and $\mathcal{D}_3$, but here the minimum was 41 and 45. The minimum training set size of 41 and 45 is because $\mathcal{D}_2$ and $\mathcal{D}_3$ only contain 641 and 645 training examples, respectively. As mentioned in section 5.1, the number of non-convergent CFD simulations were higher for $\mathcal{D}_2$ and $\mathcal{D}_3$.

Training Performance

To compare the training performance of the Ph-UNet to that of the UNet, we compare the validation losses of both types of models. While training, we record the average L1 and VD on the validation set of 100 images. We assess the training performance using two criteria: stability and convergence.

Performance Across Various Amounts of Training Data

All models are evaluated on the test set from the dataset with which they have been trained. For instance, the models trained on data from $\mathcal{D}_2$ will be evaluated against unseen data from $\mathcal{D}_2$. The L1 and the VD of the Ph-UNets and the UNets will then be compared to determine if there are differences in their predictive performance and interpretability that vary with the amount of training data used. By evaluating the models across training dataset sizes, we can see how each of them utilizes the training data available. Further, we evaluate how the complexity of the three datasets affect the results.

Generalization Capabilities

We evaluate the Ph-UNet and the UNet on a more complex and unseen data *distribution* to see if Ph-UNet exhibits improved generalization capabilities. Using the Ph-UNet and the UNet trained on the full available training data from $\mathcal{D}_1$, as well as the ones trained on the full training set from $\mathcal{D}_2$, we evaluate the models on the test set from $\mathcal{D}_3$. This assessment again uses both the L1 and the VD as separate metrics. When assessing the performance measured with the L1, we use the UNet trained on all training data from $\mathcal{D}_3$ as a baseline. When assessing the performance with the VD, our baseline is the Ph-UNet trained on all training data from $\mathcal{D}_3$.

5.4 Implementation Details

Due to the large number of artificial intelligence libraries available in Python, Python was the natural choice for the implementation language. The models were trained on a computer provided by SINTEF Digital with an Nvidia GeForce RTX 3090 graphics card. All models were trained with a batch size of 16, and the number of workers for the dataloader was set to 8 to speed up I/O operations.

Hyperparameters and other implementation details are shown in table 5.1. Since we have less than 800 training examples, learning rate decay is used as Thuerey et al. 2018 showed that, for datasets of this size, reducing the learning rate during training improved the results.

Table 5.1: Parameters and settings used in training all the neural networks.

Parameter/setting	Value
Epochs	600
Learning rate	0.0006
Optimizer	Adam ($\beta_1 = 0.5$, $\beta_2 = 0.999$)
Validation dataset size	100
Test dataset size	100
Random seed	0

While the code for the experiments and the datasets used is not publicly available, due to the thesis being in collaboration with a private company, access to the GitHub repository may be granted by request.

Chapter 6

Results and Discussion

In this chapter, the results from the experiments are presented. First, we briefly summarise our findings. Then, in section 6.2, we present our findings regarding the use of the geometry masking loss term. In section 6.3, we present the evaluation on the training stability of the Ph-UNet and the UNet. To visually convey the predictive performance of the Ph-UNet and the UNet, we provide a qualitative assessment in section 6.4. In section 6.5 we present our analysis of the Ph-UNets and the UNets performance across training dataset sizes, and in section 6.6 we discuss their generalization capabilities when evaluated on unseen distributions. Lastly, we comment on training and prediction time for the two types of models.

6.1 Summary

We found that, compared to the UNets, the PINNs converge faster and are more stable during training in 68.5% of the cases. However, with the amount of training we have used, both types of models are able to converge to low L1 on both validation data and test data.

We found no significant differences in the predictive performance of the PINN compared to the baseline UNet. However, in terms of obeying the continuity equation, our PINNs outperformed the UNets on statistically significant levels. In addition, the PINNs showed great potential for predicting flow fields on unseen data distributions that obey the continuity equation, showing that inclusion of a physics-informed loss term improves the interpretability of UNet.

The speed-up when using neural networks compared to using CFD simulations is substantial. Simulating one example using the CFD RANS approach presented in Appendix A takes approximately 30 seconds. Approximating the velocity with a neural network takes approximately 0.09 seconds – over 300 times faster.

6.2 Masking Geometry

Here, the results of Experiment 1 are presented. The experiment was conducted to determine if including the loss term masking the geometry consistently improves the performance of Ph-UNet. Figure 6.1 shows the distributions of the test set L1 for UNet, VD-UNet, and GM-UNet, and Figure 6.2 shows the distributions of the test set VD for the three types of models.

As we can see from Figure 6.1 and Figure 6.2, the constraint from adding the geometry masking loss term does not consistently improve the predictions of the neural network. For the three sets of training data from $\mathcal{D}_1$, with either 10, 100, or 797 samples, there is no clear trend in the variability or the median when comparing the GM-UNet with the UNet and VD-UNet. In line with Occam's razor[1], we therefore dismiss the geometry masking loss term. The following sections present the results from Experiment 2, where we focus on investigating the VD loss term to understand how modelling some physical behaviour impacts a model's predictions.

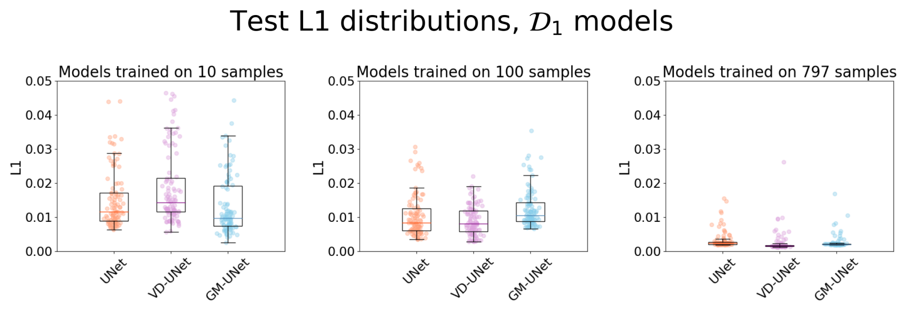

Figure 6.1: Distributions of test set L1 for UNet, VD-UNet, and GM-UNet, where $\mathcal{D}_1$ is used for training and testing.

[1]A principle that, for machine learning, states that we should prefer simpler models over complex ones, as long as their performance as similar.

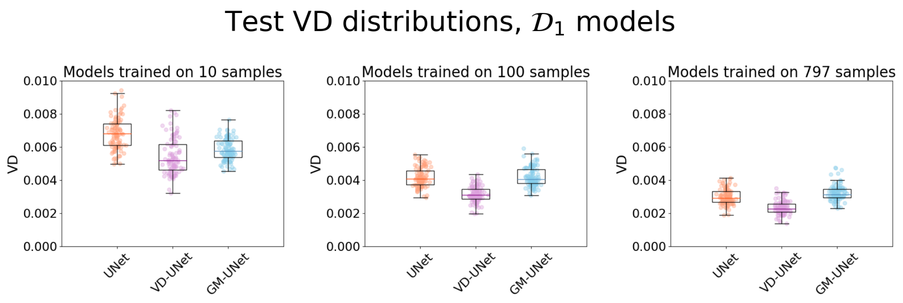

Figure 6.2: Distributions of test set VD for UNet, VD-UNet, and GM-UNet, where $\mathcal{D}_1$ is used for training and testing.

6.3 Training Stability

The first analysis for Experiment 2 is structured to answer research question 1, as described in section 5.3.2. Training stability is evaluated from the evolution of the L1 and the VD during the training of the models. We first present an analysis of the predictive performance of the models during training by looking at the L1 loss on the validation set. Then, we discuss how the two models compare in terms of learning the continuity equation by analysing the validation VD.

6.3.1 Learning to Predict Wind

To assess the training behaviour of the Ph-UNet[2] and the UNet, we present a qualitative analysis followed by a quantitative analysis. For the qualitative analysis, we study the evolution of the validation L1 for a selection of models. For the quantitative analysis, we summarize the validation L1 for all models.

Figure 6.3 shows the validation L1 for three out of the 22 pairs of Ph-UNets and UNets (left), along with the corresponding sliding window average and standard deviation (right) using a window size of 50. For the models depicted, Ph-UNet have smaller sliding window means and standard deviations, which can be caused by either higher stability or faster convergence. There is no sign of overfitting, as seen by the non-increasing trend in the validation L1.

We can clearly see from the leftmost plots of Figure 6.3 that the Ph-UNets have smaller spikes in the validation L1 than the UNets. The Ph-UNets are thus more stable during training. The spikes influence both the sliding window

[2]As a result of the findings in Experiment 1, Ph-UNet is equivalent to VD-UNet.

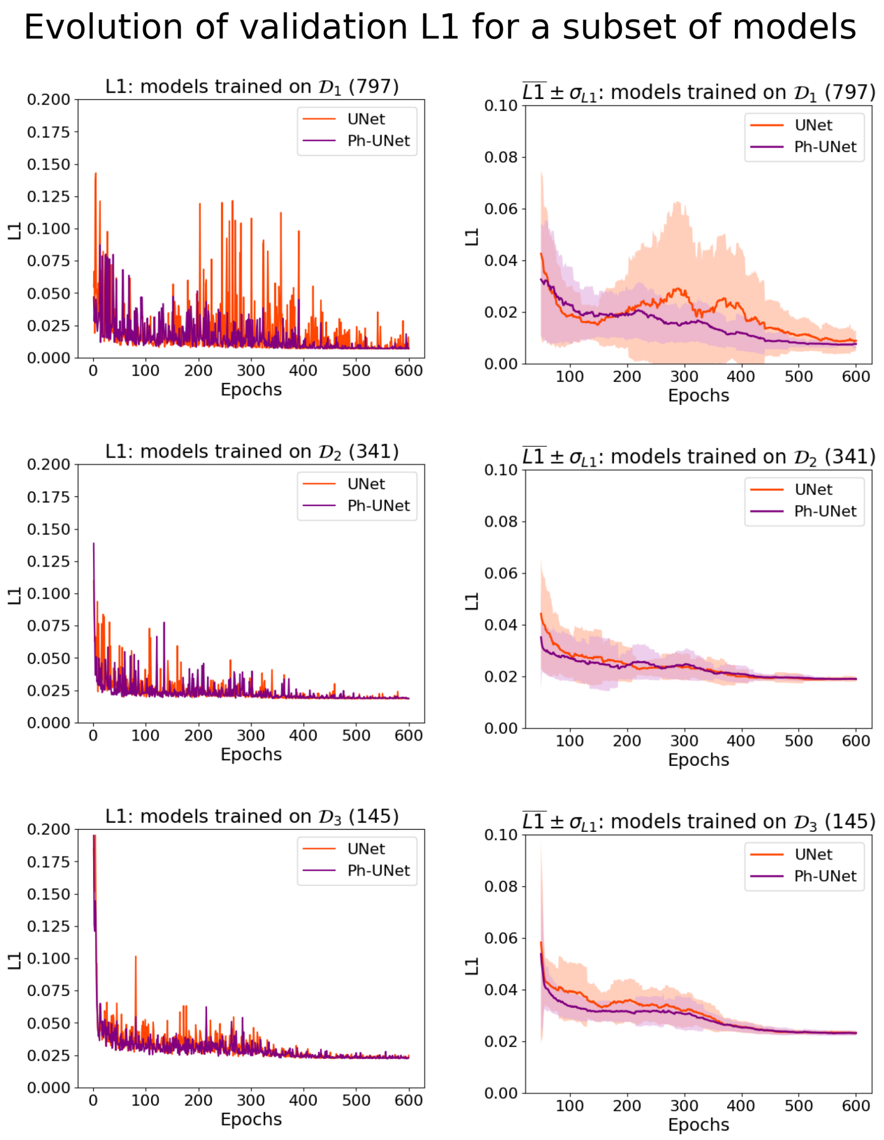

Figure 6.3: Validation L1 (left) and the corresponding sliding window mean and standard deviation (right) during training. The size of each training set is shown in parentheses. Overall, the Ph-UNets appear to be more stable and may exhibit faster convergence.

mean and the sliding window standard deviation. The sliding window mean is also affected by convergence. It is therefore difficult to determine convergence by analyzing the sliding window mean. The rightmost plots do indicate that the Ph-UNets converge faster than the UNets, which we confirm in the upcoming quantitative analysis.

To quantify the training convergence and stability of all 22 pairs of Ph-UNets and UNets, we use the median and the median absolute deviation (MAD). Comparing the median for the models tells us how well they converge, while the MAD quantifies how stable they are.

As we can see in Figure 6.4, the Ph-UNets performance and stability is overall better than the UNets. Training sets where Ph-UNet has a lower median and MAD than UNet are made darker in the plots, to convey when Ph-UNet performs better. For a larger version of this figure, refer to Figure C.1 Appendix C.

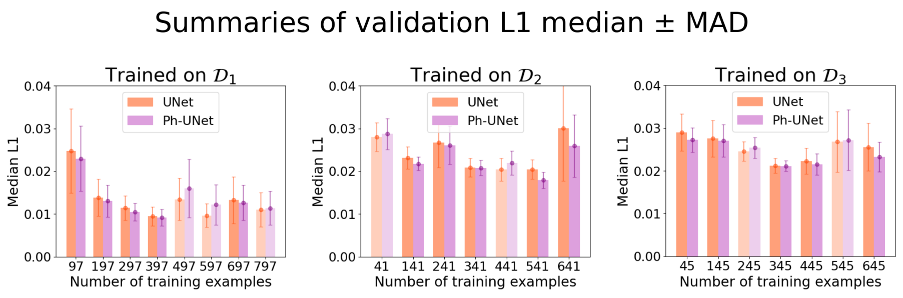

Figure 6.4: Median and MAD validation L1 from training for all models, by dataset. The 8 Ph-UNets and UNets trained on $\mathcal{D}_1$ are shown in the leftmost plot, the 7 model pairs trained on $\mathcal{D}_2$ are in the middle plot, and the 7 model pairs trained on $\mathcal{D}_3$ are to the right.

The Ph-UNets have both a lower median and a lower MAD than the UNets for 5/8 of the training sets from $\mathcal{D}_1$, corresponding to 62.5% better training performance and stability. For $\mathcal{D}_2$, the Ph-UNets outperform the UNets for 5/7 training sets, or in 71.4% of the cases. The same happens for $\mathcal{D}_3$. Averaging for the three datasets, we see that training with the physics-informed loss function results in a model that converges faster and is more stable during training in 68.5% of the cases. In practice, this means that one could reduce the training time when using a Ph-UNet model instead of a UNet. Lastly, we note that for the largest training set in $\mathcal{D}_1$, using 797 examples, Ph-UNet has a slightly higher median than UNet but a lower MAD. In this case, the Ph-UNet is thus slower to converge than the UNet, but more stable.

As mentioned in section 3.3, Ma et al. 2021 showed that a PINN was able to converge faster for the task of predicting laminar flow, compared to an unsupervised physics-driven model. Our data depicts simulations of turbulent flow. We have shown that a PINN is able to converge faster than a purely data-driven model for the more complex case of turbulent flow. This suggests that PINNs for fluid flow prediction is also efficient for turbulent flow.

Causes for Inconsistent Results

We see that in some cases, increasing the training dataset by 100 samples increases the validation L1 for both Ph-UNet and UNet. One example of this is when moving from 397 to 497 training examples from $\mathcal{D}_1$. An explanation for this increase is that the training dataset contains relatively few samples for some types of building geometries. Under-represented geometries will cause the neural networks to have trouble learning the mapping of these geometries, resulting in a larger validation L1. This pertains to Research Question 3, regarding generalization to unseen distributions, which we discuss further in section 6.6.

6.3.2 Learning to Obey Physics

In Figure 6.5 we show the validation VD and the sliding window mean and standard deviation, again using a window size of 50, for the same models as in Figure 6.3. This indicates that the Ph-UNets are all able to learn the continuity equation better than the UNets. It is worth noting that also the UNets appear to reduce the VD when training more. While UNet does not guarantee interpretability, the close approximation of the targets allows it to fulfill the continuity equation quite well, as the targets themselves fulfill the continuity equation.

To investigate if the Ph-UNets are consistently better at obeying the continuity equation, we again use the median and the MAD. In Figure 6.6[3], we show the validation VD from training all models. The Ph-UNets outperform the UNets in terms of obeying the continuity equation and exhibit higher stability, shown by the lower medians and MADs, respectively. Including the physics-informed loss term thus improves the neural network's ability to obey the physical law being modelled.

[3]In Figure C.2 in Appendix C, an enlarged version of this figure is presented.

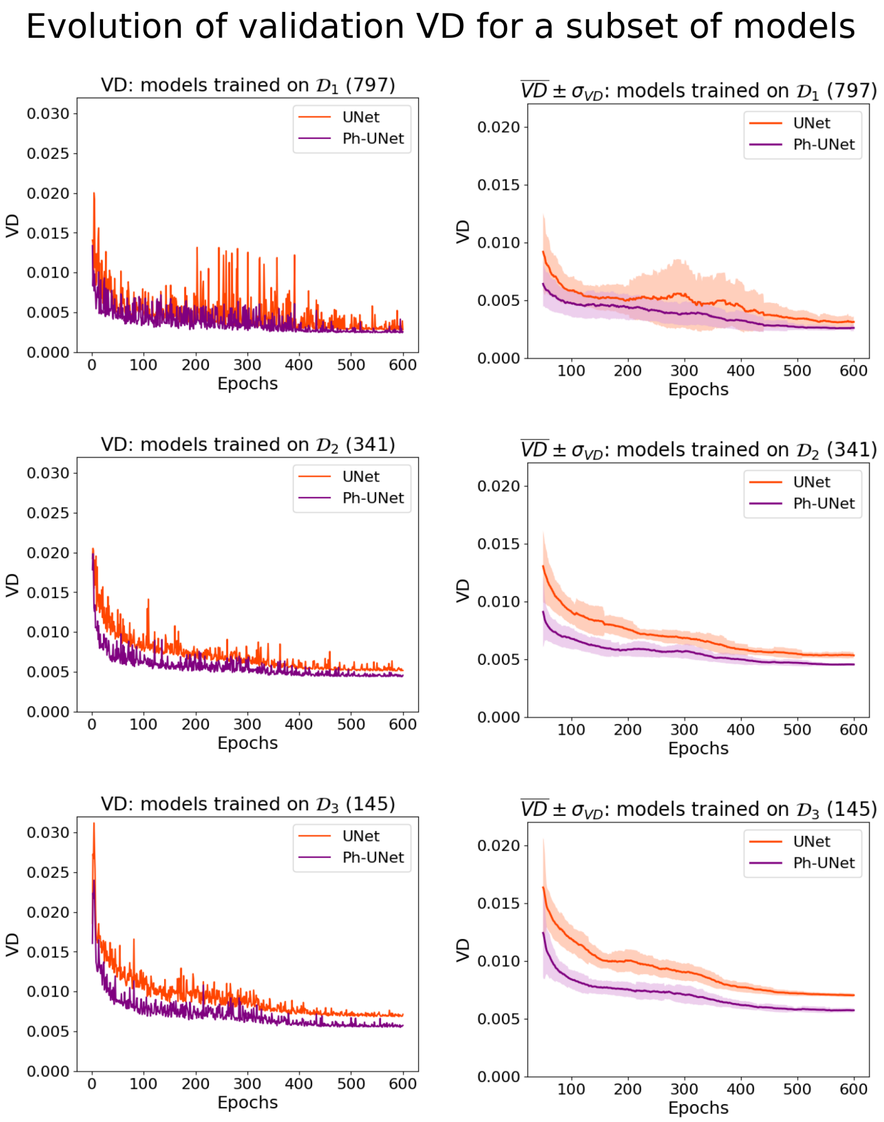

Figure 6.5: Validation VD (left) and the corresponding sliding window mean and standard deviation (right) during training. The size of each training set is shown in parentheses. While both types of models converge, the Ph-UNets have lower values all through training and seem to be more stable.

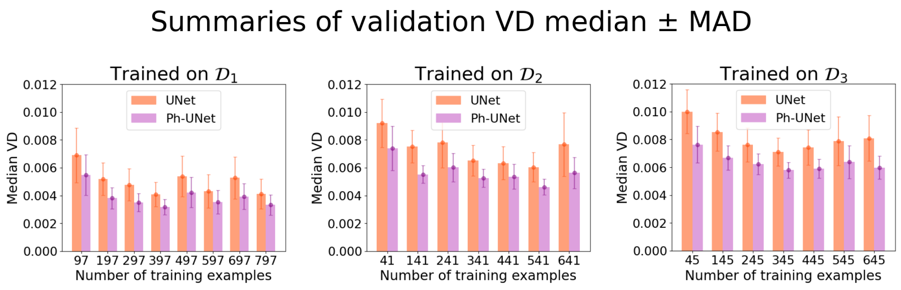

Figure 6.6: Median and MAD validation VD from training for all models, by dataset. The 8 Ph-UNets and UNets trained on $\mathcal{D}_1$ are shown in the leftmost plot, the 7 model pairs trained on $\mathcal{D}_2$ are in the middle plot, and the 7 model pairs trained on $\mathcal{D}_3$ are to the right.

6.4 Qualitative Assessment

In this section, we present a qualitative analysis to visually convey the predictive performance of the Ph-UNets and the UNets. We have selected five data examples from the test set, and display results produced by the models that have been trained on the maximum amount of training data. Each example is displayed with three types of visualizations. The first row in each figure shows the x-component of the velocity, v_x, and the second row shows the corresponding absolute error for UNet and Ph-UNet. The third row depicts the contours and the streamlines for the wind field, illustrating $\vec{v}$.

Generally, the predictions from both Ph-UNet and UNet are of high quality, as those shown for $\mathcal{D}_1$ in Figure 6.7. Here, the errors are mainly around the edges and the corners of the buildings. For urban wind flow prediction, such small errors are negligible as they are too small to affect pedestrian comfort and safety. With increasing dataset complexity, i.e. for $\mathcal{D}_2$ and $\mathcal{D}_3$, less accurate predictions seem to occur more often. However, as shown in Figure 6.8 for $\mathcal{D}_3$, these predictions are still impressive. The high performance of both models is in line with them both having low L1 on the test set, which we shortly will demonstrate is the case.

In some cases, either UNet, Ph-UNet, or both, have more noticeable deviations from the target. Examples of each of these cases are shown in Figure 6.9–Figure 6.11. The first two of these figures illustrate examples from $\mathcal{D}_2$, and the last illustrates an example from $\mathcal{D}_3$. The last example also highlights differences in the predictions from Ph-UNet and UNet.

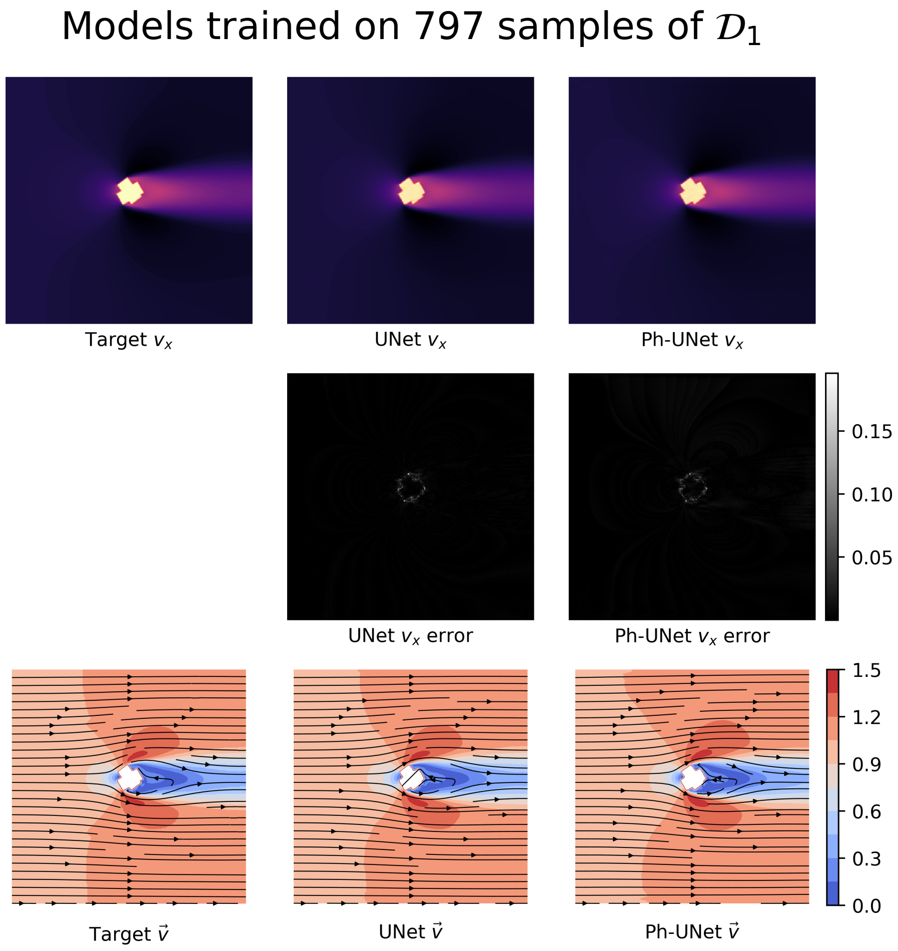

Figure 6.7: An example from $\mathcal{D}_1$ where both the UNet and the Ph-UNet provide excellent estimates. There are slight errors at the building's edges and corners, but these are negligible for urban wind flow prediction.

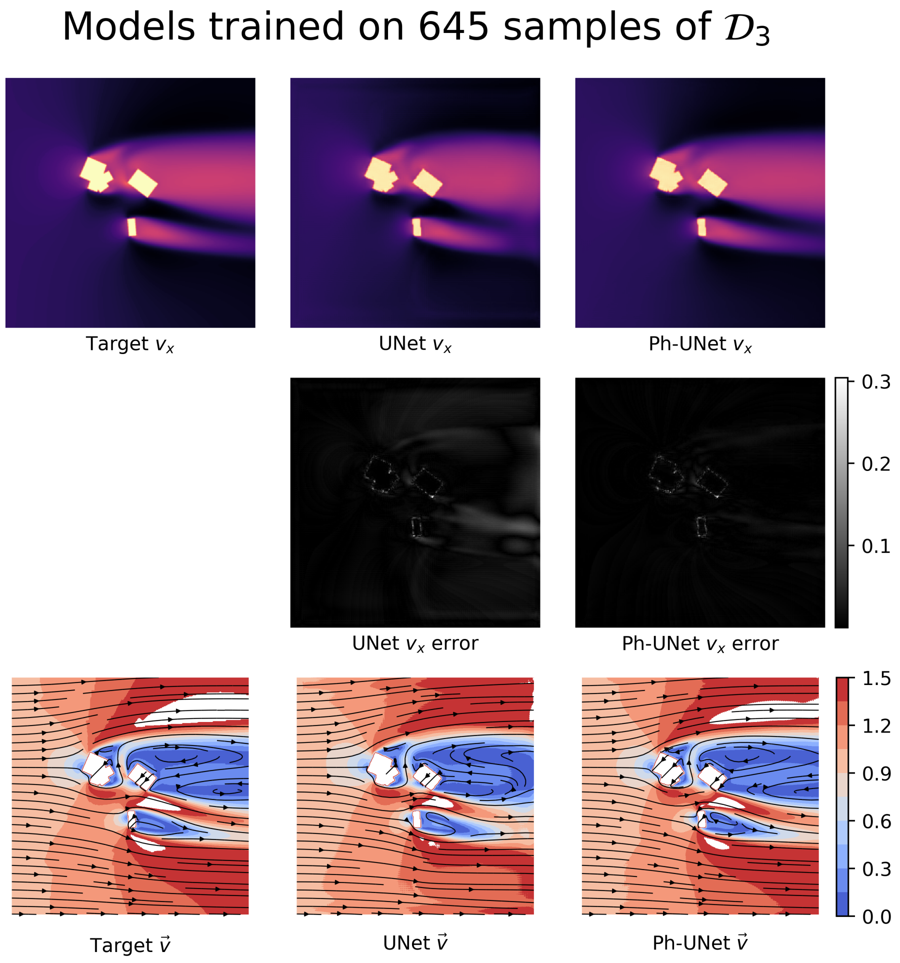

Figure 6.8: An example from the most complex dataset $\mathcal{D}_3$, where occurrences of non-optimal performance from Ph-UNet and UNet are more frequent than for $\mathcal{D}_1$. For these cases, both types of models still perform quite well. In this example, particularly the UNet's prediction lacks accuracy, but the contours and streamlines also shows that Ph-UNet's prediction is not perfectly accurate.

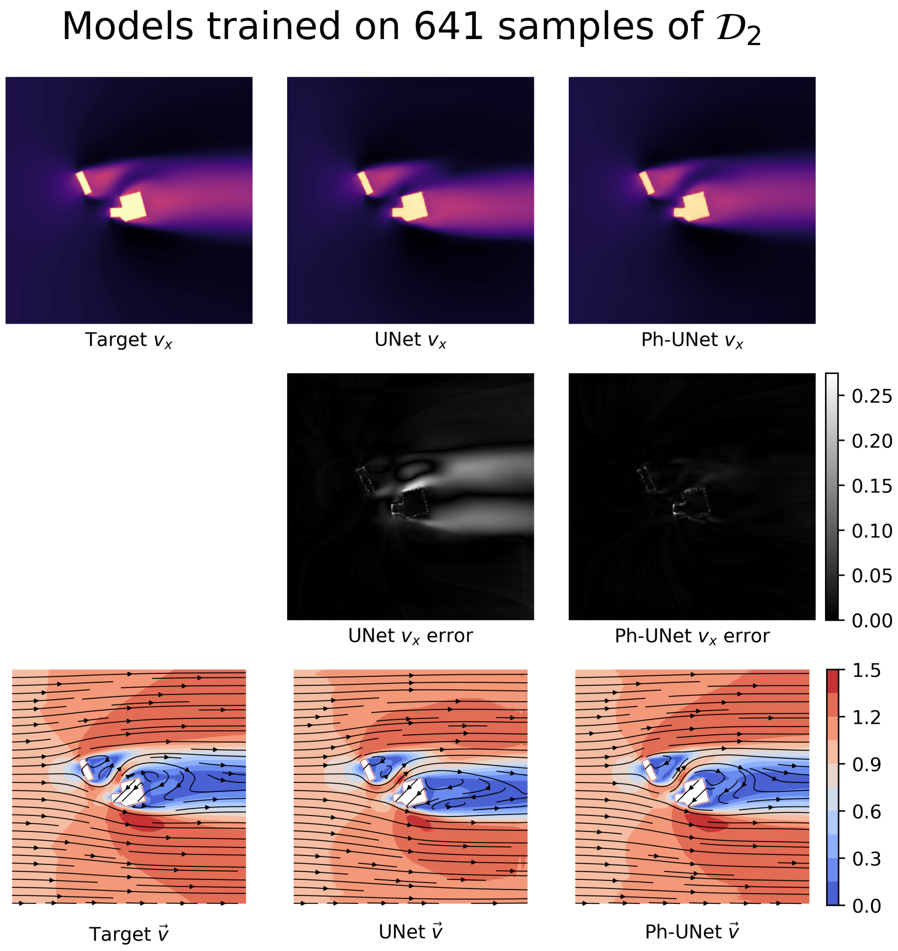

Figure 6.9: For this example, taken from $\mathcal{D}_2$, the UNet's prediction does not achieve satisfactory accuracy. UNet's prediction is particularly inaccurate in the wake.

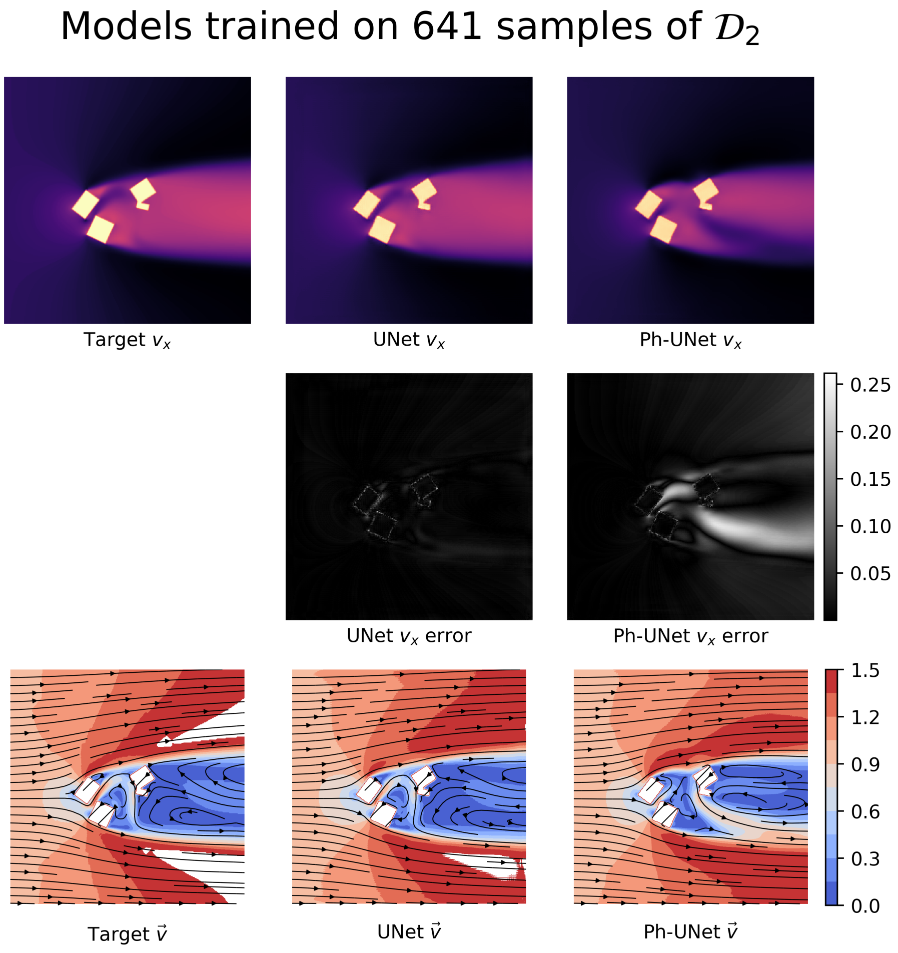

Figure 6.10: An example from $\mathcal{D}_2$ where the Ph-UNet's prediction does not achieve satisfactory accuracy. The largest issues in Ph-UNets prediction is the wake.

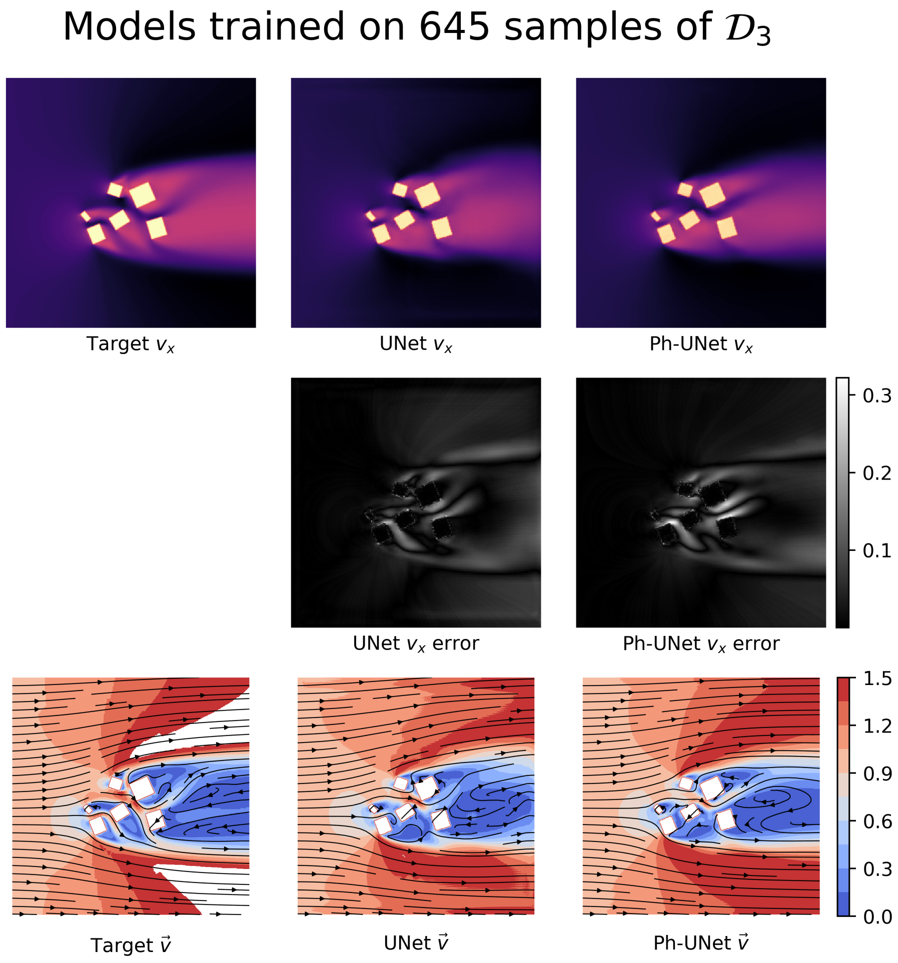

Figure 6.11: An example from $\mathcal{D}_3$ where both the UNet and the Ph-UNet struggle to predict the wind flow. The streamlines also show that their output differs from one another.

6.5 Performance across Training Dataset Sizes

For the second analysis of Experiment 2, we move our focus away from evaluating the training process of the models. Instead, to provide answers to research question 2, we investigate how the fully trained neural networks perform on test data.

For all 22 model pairs, we test their predictive performance and interpretability on a test set of 100 unseen samples from the same dataset $\mathcal{D}_i$, $i = 1, 2, 3$, that the models were trained on. A common way to evaluate models is using the average of the evaluation metric over the test samples. We report the distributions of the test set loss, instead of the average, to retain more information about the models' performance on unseen data.

To assess whether or not the differences in the predictive performance of the Ph-UNets and the UNets are statistically significant, we use the Wilcoxon signed-rank test introduced in Wilcoxon 1945. The null hypothesis of this test is that two related samples come from the same distribution. We thus use the Wilcoxon signed-rank test on the test set predictions from the Ph-UNets and the UNets to determine the likeliness of $H1$, i.e. the likeliness that the Ph-UNets and UNets have learned two different approximations of the desired mapping. We determine the results to be statistically significant for p-values lower than 0.01.

6.5.1 Predictive Performance

The distributions of the L1 test loss for the Ph-UNets are compared to that of the corresponding baseline UNets in Figures 6.12–6.14, illustrating the results for $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$, respectively. A larger version, showing these figures side-by-side, is shown in Figure C.3 in Appendix C. Statistically significant results are highlighted with darker outlines of the boxes. We see that for all three datasets, most training dataset sizes lead to non-significant differences in the test loss distribution for the Ph-UNet and the UNet. For the most complex dataset, $\mathcal{D}_3$, there might be a trend of Ph-UNet performing better for the three largest datasets. We return to this topic shortly.

The most interesting result is that, across all three datasets, the distributions of the L1 are more symmetric for the smallest training set sizes. When more training data is made available, the distributions quickly become more skewed. This indicates a lack of convergence for the models trained on the smallest amounts of images, suggesting that a minimum amount of training data is necessary for both Ph-UNet and UNet to converge properly.

The amounts of training data needed for the models to converge varies depending on the complexity of the dataset. For the least complex dataset, $\mathcal{D}_1$, we see from Figure 6.12 that the models are beginning to reach convergence with only 97 samples in the training set. Figure 6.13 shows that, for $\mathcal{D}_2$, the models

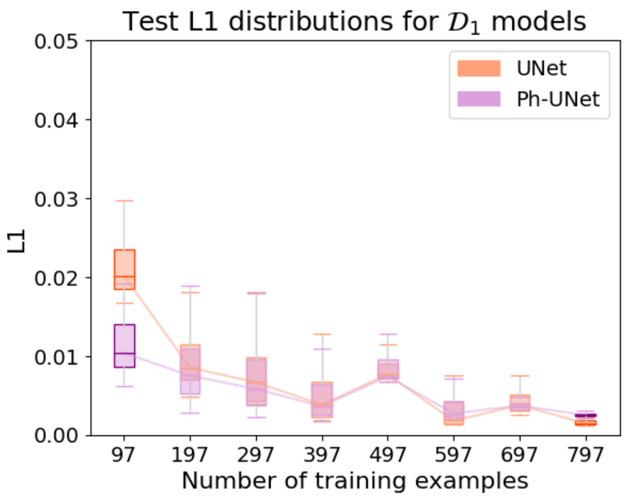

Figure 6.12: The distributions of the test set L1 for $\mathcal{D}_1$, comparing Ph-UNet and UNet for 8 training set sizes.

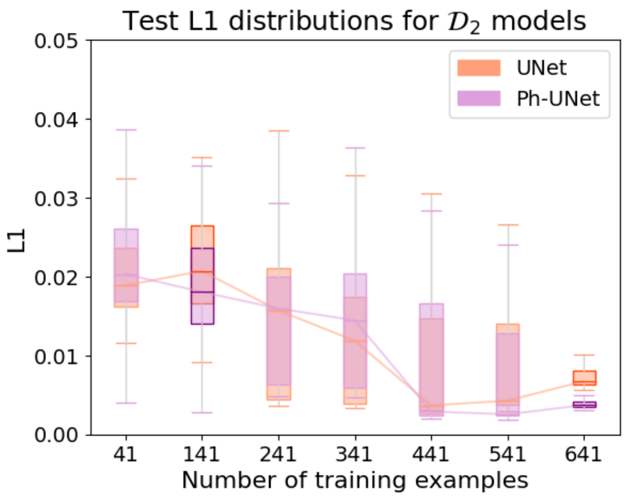

Figure 6.13: The distributions of the test set L1 for $\mathcal{D}_2$, comparing Ph-UNet and UNet for 7 training set sizes.

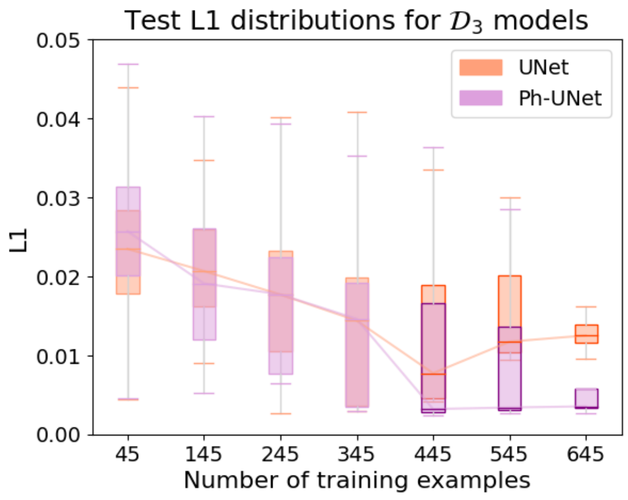

Figure 6.14: The distributions of the test set L1 for $\mathcal{D}_3$, comparing Ph-UNet and UNet for 7 training set sizes.

begin to converge with 241 samples in the training set. The increasing complexity of 4 buildings instead of 2 thus has a noticeable impact on the amount of training data required. For the most complex dataset, $\mathcal{D}_3$, Figure 6.14 shows that the models seem to begin to converge with around $245 - 345$ samples in the training set.

The relatively large increase in complexity from $\mathcal{D}_1$ to $\mathcal{D}_2$, compared to that from $\mathcal{D}_2$ to $\mathcal{D}_3$, is in line with the general values of the distributions. The L1 distributions for $\mathcal{D}_2$ and $\mathcal{D}_3$ have larger variance and and higher values than the distributions for $\mathcal{D}_1$[4]. It does make sense that the datasets with 4 and 6 buildings have such an increase in complexity, as the increasing number of buildings allow for substantially more complicated geometries to form. While the 6-building dataset, $\mathcal{D}_3$, is more complex than the 4-building dataset, $\mathcal{D}_2$, the relatively small increase in complexity from $\mathcal{D}_2$ to $\mathcal{D}_3$ is likely caused by an increased number of overlapping geometries for the increased number of maximum geometries. When increasing the number of geometries from 4 to 6, the probability of overlapping geometries increases. This, in turn, results in more similarities for $\mathcal{D}_3$ and $\mathcal{D}_2$ than for $\mathcal{D}_2$ and $\mathcal{D}_1$.

While the analysis of the convergence and dataset complexity provides insight into which amounts of training data can be deemed low-resource regions of train-

[4]This is easier to see from Figure figure C.3.

ing data, the regions where the models do converge are also interesting. Given sufficient amounts of training data, both types of neural networks eventually converge on all three datasets. This shows that PINNS, as well as data-driven neural networks, are viable for turbulent flow prediction in both simple and complex systems. As the prediction of urban wind is complex, both due to turbulent flow and due to complex building geometries, this finding encourages the use of our interpretable PINN for urban wind prediction.

Statistically Significant Differences in Predictive Performance

For some of the training set sizes, Figures 6.12–6.14 convey statistically significant differences in the test L1 distributions from Ph-UNet compared to UNet. However, these differences are likely due to fluctuations in the quality of the models' learned mapping.

Our qualitative analysis of the validation loss in section 6.3.1 revealed that both the Ph-UNets and the UNets seemed to converge towards the same L1 value. In Figure 6.15, we zoom in on the tail of the validation loss for the models trained on 341 images from $\mathcal{D}_1$. It is clear that in this case, for the last 50 epochs of training, both models converge to a similar value.

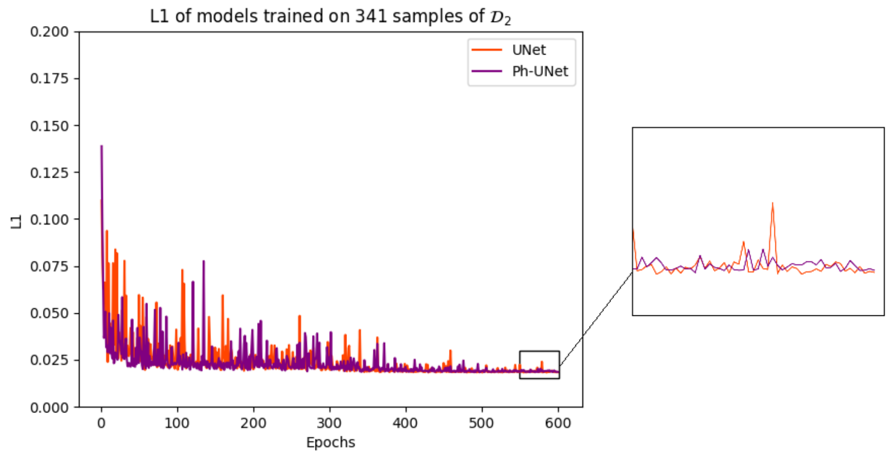

Figure 6.15: Validation L1 for Ph-UNet and UNet trained on 341 samples of $\mathcal{D}_1$, focusing on the validation L1 for the last 50 epochs. Both types of models converge towards the same value.

As we train all models for 600 epochs before evaluating, the state of the Ph-UNets and UNets at the 600th epoch determines how well they perform on test data. The fluctuations in the L1 from one epoch to the next are larger than the average difference in L1 between the two types of models. Therefore, the Ph-UNet will not consistently outperform the UNet. However, as we have shown Ph-UNet to be more stable during training, there is an increased probability that Ph-UNet has a lower L1 at the 600th epoch. The inclusion of our physics-informed loss term did not improve the predictive capabilities of the neural network, and the main benefit of this loss term so far is faster convergence and more stable training.

Lack of Improved Predictive Performance

The addition of the physics-informed loss term gives the Ph-UNets an inherent ability to extract more information from each target image, as these undergo an explicit transformation in the physics-informed loss term. With more information available, the Ph-UNets' predictions should result in lower L1 than the UNets' predictions. We postulate two explanations for why the Ph-UNet is unable to outperform UNet in terms of L1.

Firstly, the Ph-UNets and the UNets may focus on predicting different parts of the image. L1 only measures the *average* pixel-wise difference. If the Ph-UNet and the UNet prioritise different regions of the images, the L1 will not reflect these priorities. In Appendix B, we provide an analysis to determine if there are differences in the model's predictions accross three regions of the images. The three regions are the area around and between the buildings, the wake, and the rest of the image. The results from the analysis showed that there are no notable differences between the Ph-UNet's and the UNet's performance for these three regions, compared to that of the full image. The results did however confirm that the most difficult regions for both models are around and between the buildings, and the wake area.

The second possible reason is that the neural networks might be large enough, measured in the number of parameters, and trained long enough, for the UNets to learn to calculate the VD by itself. By learning this calculation, the UNets will have the same information as the Ph-UNets have. We saw in Figure 6.5 that, to some degree, the UNets learn to fulfill the continuity equation during training. The similar results for the UNets and the Ph-UNets can thus partly be due to the expressive capabilities of the baseline architecture. Compared to the UNet architectures used by Thuerey et al. 2018, we have used an architecture where the number of parameters is in the mid-range. Larger performance differences for the L1 might occur with smaller neural networks.

6.5.2 Interpretability

In Figure 6.16 we show a side-by-side comparison of the distributions of the VD for the three datasets. The details of the VD distributions are less interesting than those of the L1. The details of the VD distributions are apparent from a larger version of Figure 6.16, shown in Figure C.4 in Appendix C. The Wilcoxon signed-rank test shows that all differences between the test losses from Ph-UNets predictions and the test losses from UNets predictions are statistically significant.

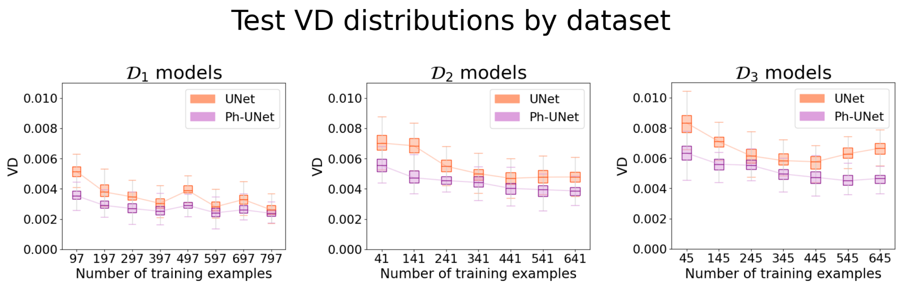

Figure 6.16: The distributions of the test VD loss for models trained and tested on $\mathcal{D}_1$ (left), $\mathcal{D}_2$ (middle), and $\mathcal{D}_3$ (right). For all three datasets, the Ph-UNets are better at obeying the continuity equation than the UNets are.

Figure 6.16 confirm what we saw during training, namely that the Ph-UNets are better at predicting flow fields that obey the continuity equation. For all three datasets, the differences in the VD between Ph-UNet and UNet are larger for smaller amounts of data. In this region of training dataset sizes, UNet is unable to learn to predict the fluid field well enough for it to also obey the continuity equation properly. In general, both Ph-UNet and UNet benefit from more training data when it comes to reducing the VD. For $\mathcal{D}_3$, there are also larger differences for the larger training datasets. This is likely to be connected to the performance on the L1. We see that the VD distributions have similar trends to the L1 distributions, e.g. for 497 training examples from $\mathcal{D}_1$ where both distributions increase. As shown in the previous section, the Ph-UNets trained on the larger amounts of $\mathcal{D}_3$ had a low validation L1 at the 600th epoch, compared to the UNets. Thus, for the larger amounts of training data from $\mathcal{D}_3$, the Ph-UNets ended their training with a beneficial approximation to the mapping. A better approximation to the mapping also results in a better approximation of the VD, as the targets are generated using the continuity equation, resulting in the improved interpretability for these training sets. Another possible explanation for the increased difference in VD for the largest training dataset sizes from $\mathcal{D}_3$ is the dataset complexity. The increased complexity of $\mathcal{D}_3$, compared to the two other datasets, increases the influence of the physics-informed loss term. Thus, when the amounts of training data required for the models to converge are available, Ph-UNet is able to utilize the samples more efficiently for modelling the continuity equation.

The side-by-side comparison shows that there is an overall increase in the general level of the VD as the complexity of the datasets increase. This is not surprising, as neither Ph-UNet or UNet has information regarding momentum conservation[5]. Increasing the number of buildings increases the area where the momentum equation is most pertinent, namely next to the buildings, and thereby makes it more difficult to produce accurate flow predictions without explicit information about this equation.

The differences in VD between Ph-UNet and UNet are consistent and statistically significant. However, the values of the VD are an order of magnitude smaller than those of the L1. Therefore, the differences in predictions that lead to these differences in VD does not significantly affect the the L1. An overall goal is to increase the predictive performance of the neural network, i.e. to reduce the test L1. However, the improved interpretability exhibited by the PINN convey the potential of integrating more physical knowledge into a neural network. Integrating additional physics-informed loss terms could affect the neural network's predictions enough to improve predictive accuracy.

Recall that one of the models developed in Wang et al. 2020 also had two loss terms, one data-driven and one that modeled the continuity equation. Both their models had a complex LES-RANS inspired neural network architecture. Their PINN was unable to perform well on both L1 and VD, compared to their purely data-driven model, and instead had to prioritize one of the metrics. It is therefore quite interesting that our classic deep learning architecture results in a PINN (Ph-UNet) that is able to match the performance of the corresponding data-driven model (UNet) on L1 while outperforming the data-driven model on the VD. This suggests that using a more traditional deep learning architecture as a base and then guiding it with scientific knowledge is an efficient way of combining deep learning and fluid dynamics.

6.6 Generalizing to an Unseen Distribution

The third, and final, analysis for Experiment 2 is structured to answer research question 3. To assess the generalization capabilities on a more complex, unseen data distribution, we evaluate the models that have been trained on all training

[5]The second Navier–Stokes equation.

data from $\mathcal{D}_1$ or $\mathcal{D}_2$ on the test set from $\mathcal{D}_3$. $\mathcal{D}_1$ only includes 2 possibly overlapping buildings, while $\mathcal{D}_2$ includes 4 and $\mathcal{D}_3$ includes 6. We thus test how the neural networks generalize to a higher complexity than what they are trained on.

6.6.1 Predictive Performance

Figure 6.17 shows the L1 distribution on the test set of $\mathcal{D}_3$ for UNet and Ph-UNet trained on $\mathcal{D}_1$, and $\mathcal{D}_2$, and the baseline UNet. The UNet and Ph-UNet trained on the full training set from $\mathcal{D}_1$ are represented with the two leftmost boxplots, while the performance of the UNet and Ph-UNet trained on the full training set from $\mathcal{D}_2$ are shown in the middle. The boxplot to the right is the baseline used in this generalization task, which is the UNet trained on the full training set from $\mathcal{D}_3$.

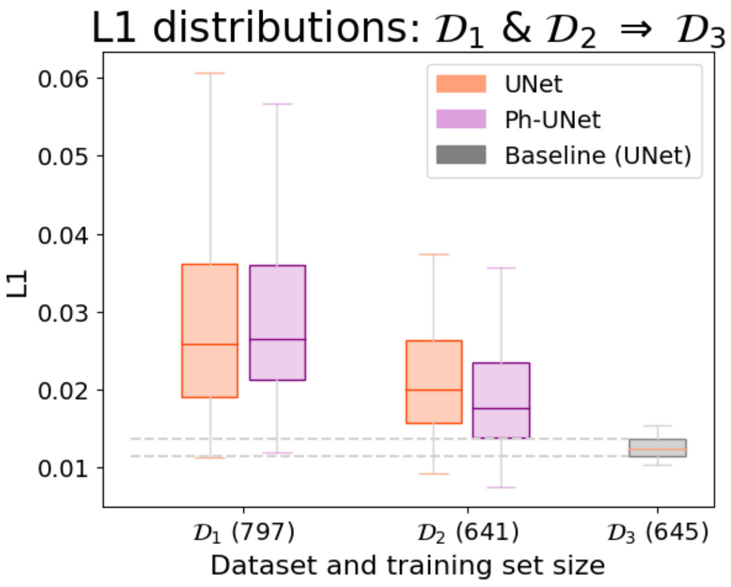

Figure 6.17: Distributions of L1 for UNet and Ph-UNet trained on all training data from $\mathcal{D}_1$, and for UNet and Ph-UNet trained on all training data from $\mathcal{D}_2$. All four models are then evaluated on the test set from $\mathcal{D}_3$. As a baseline, the test set L1 for the UNet trained on all training data from $\mathcal{D}_3$ is used.

The figure clearly shows that neither the Ph-UNet trained on $\mathcal{D}_1$ nor the Ph-UNet trained on $\mathcal{D}_2$ are able to reach the same predictive performance as the baseline in this case. The models trained on $\mathcal{D}_2$ do perform better than those trained on $\mathcal{D}_1$. This difference in performance is unsurprising, as $\mathcal{D}_2$ is more complex and thus more representative of $\mathcal{D}_3$ than what $\mathcal{D}_1$ is. The general lack of improved predictive performance is unsurprising considering our discussion in section 6.3.1, where we saw that underrepresented samples in the training data made it hard for both models to generalize to the validation data. Thus, generalizing to a completely unseen data distribution should be a challenge for both models.

The Wilcoxon signed-rank test shows that there are no statistically significant differences in the predictions from the UNet and Ph-UNet trained on $\mathcal{D}_1$, where the p-value is 0.874. The same holds for the two $\mathcal{D}_2$-models, where the p-value is 0.093. We therefore cannot conclude that the inclusion of a physics-guided loss term improved the generalization capabilities of the UNet architecture. This is in line with the results in section 6.5 where we concluded that there are no significant differences in the predictive performance of the Ph-UNet and the UNet.

6.6.2 Interpretability

In Figure 6.18, we present the VD distributions on the test set of $\mathcal{D}_3$ for UNet and Ph-UNet trained on $\mathcal{D}_1$, and $\mathcal{D}_2$, and a baseline Ph-UNet. We compare the Ph-UNet and UNet trained on all training data from $\mathcal{D}_1$ (left) and $\mathcal{D}_2$ (middle) and the baseline Ph-UNet trained on all training data from $\mathcal{D}_3$.

As we can see in Figure 6.18, the inclusion of the physics-guided loss terms allows the models to better predict a fluid flow that adhere to the continuity equation. The differences in the test VD losses for the predictions from the UNets and the Ph-UNets are statistically significant. For the models trained on $\mathcal{D}_1$, the p-value is on the order of 10^{-7}, while for the models trained on $\mathcal{D}_2$ the p-value is on the order of 10^{-15}. In addition, the Ph-UNet trained on 4 buildings is able to achieve test losses that approach the baseline's test losses. The p-value for the Wilcoxon signed-rank test performed on the test loss distribution from the Ph-UNet trained on $\mathcal{D}_2$ and the test loss distribution of the baseline Ph-UNet is 0.00091. The two models are thus not likely to have approximated the same mapping. Even so, the four-building Ph-UNet has generalized its representation of the VD very well. In general, the inclusion of the physics-informed loss term allows the neural network to predict flow fields for an unseen data distribution that better adhere to the continuity equation, while not negatively affecting the prediction of the fluid flow.

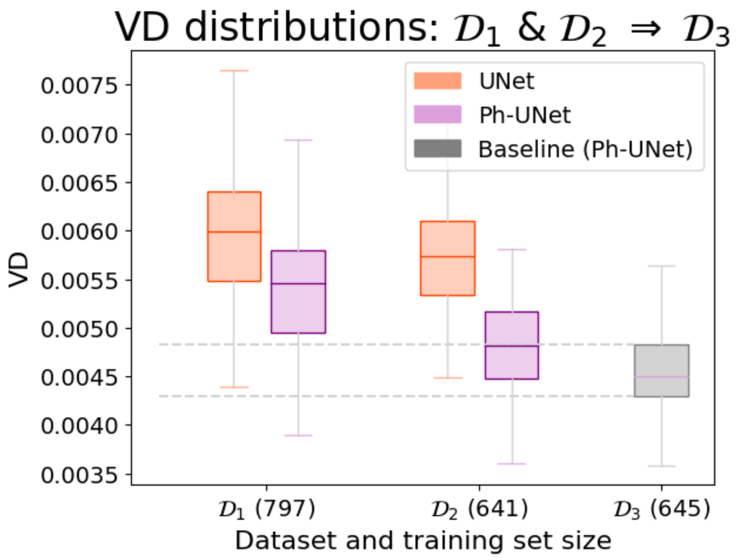

Figure 6.18: Distributions of the VD for UNet and Ph-UNet trained on all training data from $\mathcal{D}_1$ (left) and $\mathcal{D}_2$ (middle). All four models are then evaluated on the test set from $\mathcal{D}_3$. As a baseline, the test set VD for the UNet trained on all training data from $\mathcal{D}_3$ is used.

6.7 Training Time and Prediction Time

The computer used for training all 44 models is a shared resource, so there is no accurate measure of runtime for the training. However, the runtime for all 600 epochs of training for each model was recorded. These runtimes were used to calculate the relative difference in cumulative training time for Ph-UNets compared to UNets:

$$\text{Relative Difference} = \frac{T_{\text{Ph-UNets}} - T_{\text{UNets}}}{T_{\text{UNets}}}, \tag{6.1}$$

where $T_{\text{Ph-UNets}}$ is the cummulative training time for all 22 Ph-UNets and T_{UNets} is the cummulative training time for all 22 UNets. Using Ph-UNet resulted in a relative increase of 8.1% in the cummulative training time. By utilizing early stopping[6], this increase in training time can be reduced – or even removed – due

[6]Early stopping is, as mentioned in section 2.2.5, the process of stopping the training before all scheduled epochs are concluded.

to the faster convergence of Ph-UNet.

While the additional calculation of the velocity divergence increases the overall training time of Ph-UNet, this calculation is not required during inference. Thus, predicting the wind flow for an example takes approximately 0.09 s when using either Ph-UNet or UNet. For comparison, the corresponding CFD simulation takes approximately 30 s. With the use of our PINN, it is thus possible to generate physically consistent approximations of wind flow around complex geometries in a fraction of the time required for CFD simulations.

Chapter 7

Conclusion

As part of an ongoing research project working to utilize deep learning for urban wind prediction, we have explored how injecting physics into neural networks can improve the performance of the neural network. With this goal in mind, we have successfully trained a Physics-Informed Neural Network (PINN) that exhibits better sample efficiency during training and consistently increased interpretability compared to a purely data-driven model.

We conclude this thesis with answers to our research questions, posed in section 1.2, followed by recommendations for future work. Finally, we include takeaways from the state of the art that may serve as a basis for general improvements for using our framework to predict real-life wind flow.

7.1 Concluding the Research Questions

We implemented a physics-informed loss function in the state-of-the-art UNet architecture, creating a PINN. Through the use of our PINN, we have coupled elegant deep learning principles with a physical constraint. We explored our framework on a range of turbulent wind flows – including wind flows of higher complexities than those explored by the research community. While the underlying physics of the system is very complex, our physics loss term only required the velocity components of the fluid flow around 2D buildings. This is a substantial difference from modelling the second conservation law for fluid flow, which requires one equation per dimension as well as the pressure of the system.

7.1.1 RQ1: How Does the Addition of Physics Knowledge Affect a Neural Network's Training Process?

We evaluated the training process of a substantial number of PINNs against corresponding data-driven models. The addition of the physics loss term yielded significant training improvements, both in terms of stability and faster convergence. Our analysis thus suggests that the PINN is able to utilize training samples better, in line with what has been reported by the research community.

7.1.2 RQ2: How Does the PINN's Performance Differ as the Amount of Training Data Changes?

With sufficient training, both the fully data-driven model and the PINN were able to attain similar predictive performance, even when analysing different regions of the images. Additionally, our PINN was consistently able to better adhere to the physical law modeled, removing the trade-off between the two metrics which was reported by Wang et al. 2020. The improvements when using the PINN were apparent for all training dataset sizes.

The analysis of how training dataset size affects the model's performance showed how much data was needed for the models to converge. For the non-convergent region, our PINN's improvement in terms of adhering to the physical law was even higher than in other regions. This indicates that the use of PINNs is a promising direction for future research into applications with scarce training resources. However, we also highlight the fact that both the PINN and the data-driven UNet were able to converge on all three datasets, showing that, in general, neural networks are viable models for approximating the fluid flow of complex systems. This finding, in combination with the interpretability of our PINN, further motivates the use of PINNs for urban wind flow prediction.

7.1.3 RQ3: How Well Can the PINN Generalize to a More Complex Data Distribution?

On the unseen distribution, our PINN outperformed the data-driven model in terms of obeying the physical law while maintaining similar predictive performance. Our results suggest that the physics-informed loss term is more generalizable across distributions, compared to the purely data-driven loss term. This opens for further research in this area, and we believe there is a lot of potential for improving neural networks' generalizability on turbulent flow prediction by using several physics-informed loss terms.

In terms of the datasets, the complexity of the training data impacted the generalization capabilities. Compared to the UNet, the PINN improved a lot more on the Velocity Divergence (VD) when trained on the dataset of medium complexity than when it was trained on the simplest dataset. However, while our PINN needed to be trained on data that is somewhat representative of the correct underlying distribution, the addition of several physics-informed loss terms might soften this requirement.

7.2 Future Work

Given the modest performance improvements from modeling the continuity equation in the UNet, there is room for future improvements. We suggest two lines of research that focus on a tighter coupling of fluid dynamics and deep learning, and one line of research for increasing the richness of the dataset.

7.2.1 Momentum Conservation

The most obvious area of improvement is to integrate more physics into the neural network. We have seen that adding one of the Navier–Stokes equations is beneficial to the neural network's training, and also consistently increased its interpretability. We have no reason to expect that including the second Navier–Stokes equation will not provide similar benefits.

Modeling the second Navier–Stokes equation, the conservation of momentum, could potentially have an even larger effect than modeling the continuity equation. The momentum conservation is more extensive in terms of modelling physics, compared to the continuity equation, explicitly enforcing physical constraints for each dimension of the system.

In addition to stronger regularization from the added physics constraints, modeling the momentum conservation requires the pressure of the system. The downside of this is the need for new datasets. However, including the pressure of the system provides more information about the fluid flow state and thus could contribute to improved predictive performance of the network.

As our framework is built in a modular fashion, expanding it to include the momentum conservation for training, and as metrics, is highly recommended.

7.2.2 Boundary Conditions

The final step for full integration between deep learning and fluid flow is the addition of boundary conditions. Adding these equations encompasses a complete description of the fluid flow system in the physics-informed loss terms. Using each physics loss term as a metric, one could also fully investigate how well the neural network obeys the physical constraints of the system.

We guided our PINN towards solutions that obeyed the continuity equation, even though these solutions might disobey the other physical constraints describing the system. Guiding a model towards solutions that obey *all* physical constraints of the system could improve predictive accuracy in complex scenarios. Our PINN was able to generalize well in terms of the VD. A physics-informed loss function that describes the complete system could lead to an overall improvement of the generalization capabilities of the neural network.

7.2.3 Geometries

While our datasets model geometries that are complex compared to those used in previous research, where the datasets normally contain $1-2$ geometries, they are simplified compared to real-life building geometries. As we are interested in the wind trajectories on a macroscopic level, some level of simplification is justified. However, our generalization experiments have highlighted the importance of representative training data, even for PINNs. It would therefore be interesting to investigate how well a PINN performs on a dataset containing a wider range of geometries, such as triangles or circles, as well as combinations of these geometries. It would also be interesting to assess how well a PINN performs on real-life buildings, using a dataset depicting different areas of a city.

7.3 Towards Urban Wind Prediction with PINNs

In the previous section, we suggested three lines of research as a continuation of this thesis. There are two additional lines of research that are worth commenting on, as they are influential to achieving state-of-the-art urban wind flow prediction. The first line of research is related to the baseline architecture used in the PINN, while the second concerns moving from two dimensions to three dimensions.

7.3.1 Architectural Elements

Enhancing the baseline architecture is vital for achieving state-of-the-art performance on fluid flow, also when using PINNs. In chapter 3, we presented several architectural elements that might improve on the baseline UNet's performance, and thus also on the PINN's performance.

The model developed by Wandel et al. 2020 could be very beneficial for generalizing to urban wind flow prediction. As mentioned in chapter 3, their spatially local model was able to train on basic geometric shapes and predict more complex shapes. Applying similar principles to a PINN could enhance the model's ability to predict wind flow in cities around the world, as building designs differ substantially between international cities.

In Høiness et al. 2021, using attention[1] improved the performance of Generative Adversarial Networks (GANs) for predicting fluid flow. In one of these GANs, introduced by Isola et al. 2016, UNet architecture was applied. This suggests that enhancing the UNet with attention may further improve the fluid flow predictions.

7.3.2 Moving to 3D

The most substantial limitation of our PINN in terms of urban wind prediction, is that we have focused on two-dimensional wind flow. For the real-life three-dimensional scenario, the wind flow would differ from the two-dimensional case both for short and tall buildings. As described in Hågbo et al. 2020, the wind flow around short buildings will change a lot in the z-direction. For tall buildings, a lot of wind will be dragged down the side of the building and down to the ground, increasing the wind speed at the pedestrian level.

Using a PINN to predict the real-life wind flow requires a neural network that supports three-dimensional data. The PINN must be able to output pressure and velocity for all three dimensions of the fluid flow system. The model developed by Musil et al. 2019 was a data-driven UNet with 3D convolutions, encouraging the use of UNet as the base architecture for a PINN in the three-dimensional domain.

In three dimensions, the PINNs ability to converge faster will be beneficial. A three-dimensional expansion of the UNet architecture will greatly increase the number of parameters, so increased training efficiency will be much more impactful.

While a line of research regarding three-dimensional PINNs is very interesting, one should first determine the impact of including more physics in the two-dimensional case. Until we can determine how substantial the benefits of using a PINN instead of a data-driven neural network can be, the complexity of the neural network architecture should not be increased.

[1] A technique to make neural networks focus on the most important aspects of the data, originally used for language models.

Bibliography

S. Brunton, B. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. 52(1):477–508, 2020. ISSN 0066-4189, 1545-4479. version: 3.

S. Chen and G. D. Doolen. Lattice Boltzmann method for fluid flows. 30(1): 329–364, 1998.

C. Cheng and G.-T. Zhang. Deep learning method based on physics informed neural network with resnet block for solving fluid flow problems. *Water*, 13, 02 2021.

M. W. M. G. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

S. Flores. Variational autoencoders are beautiful, April 2019. URL `https://www.compthree.com/blog/autoencoder/`.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)*, 313:504–7, 08 2006.

H. Høiness, K. Gjerde, L. Oggiano, K. E. T. Giljarhus, and M. Ruocco. Positional encoding augmented GAN for the assessment of wind flow for pedestrian comfort in urban areas. *CoRR*, abs/2112.08447, 2021.

T.-O. Hågbo, K. E. T. Giljarhus, and B. H. Hjertager. Influence of geometry acquisition method on pedestrian wind simulations, 2020.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

H. Ma, Y. Zhang, N. Thuerey, X. Hu, and O. J. Haidn. Physics-driven learning of the steady Navier-Stokes equations using deep convolutional neural networks, 2021.

W. Malalasekera and H. Versteeg. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method.* Pearson/Prentice Hall, 2007.

J. Musil, J. Knir, A. Vitsas, and I. Gallou. Towards sustainable architecture: 3D convolutional neural networks for computational fluid dynamics simulation and reverse designworkflow. *CoRR*, abs/1912.02125, 2019.

S. Pawar, O. San, B. Aksoylu, A. Rasheed, and T. Kvamsdal. Physics guided machine learning using simplified theories. *CoRR*, abs/2012.13343, 2020.

T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. *CoRR*, abs/2010.03409, 2020.

M. Raissi, P. Perdikaris, and G. Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. 11 2017.

M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378: 686–707, 2019. ISSN 0021-9991.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia. Learning to simulate complex physics with graph networks. *CoRR*, abs/2002.09405, 2020.

J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Mavriplis. CFD vision 2030 study: A path to revolutionary computational aerosciences. 2014. Document ID: 20140003093.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

N. Thuerey, K. Weissenow, H. Mehrotra, N. Mainali, L. Prantl, and X. Hu. Deep learning methods for Reynolds-Averaged Navier-Stokes simulations of airfoil flows. *CoRR*, abs/1810.08217, 2018.

N. Wandel, M. Weinmann, and R. Klein. Learning incompressible fluid dynamics from scratch - towards fast, differentiable fluid models that generalize. *CoRR*, abs/2006.08762, 2020.

R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu. Towards physics-informed deep learning for turbulent flow prediction. pages 1457–1466, 08 2020.

F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. ISSN 00994987.

J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar. Integrating scientific knowledge with machine learning for engineering and environmental systems, 2021.

Appendix A

Image Generation Process

The CFD simulations are performed using the commercial CFD software Simcenter STAR-CCM+ from Siemens PLM Software using the finite volume method on an unstructured grid. All datasets are provided by Nabla Flow.

In some simulation cases, an asymmetrical vortex shedding pattern in the wake appeared which results in a transient oscillating solution. To remove these patterns from the dataset, the time-averaged of the velocity field was used to approximate a time-independent solution.

Each building geometry was automatically generated by random uniform selection of attributes, constrained by the following conditions:

- Building length $\in [5, 25]$
- Building rotation $\in [0, 45]$
- Building placement for x-direction $\in [-30, 30]$
- Building placement for y-direction $\in [-30, 30]$

Three datasets were generated, with different number of geometries generated: $\mathcal{D}_1$ (2 geometries), $\mathcal{D}_2$ (4 geometries), and $\mathcal{D}_3$ (6 geometries). The geometries were allowed to overlap, so the number of buildings represented in a dataset may be less than the number of geometries generated. Each dataset originally contain 1000 examples of input-target pairs. Input-target pairs where the CFD simulation did not converge were removed, resulting in the following dataset attributes:

- $\mathcal{D}_1$: 997 examples, maximum 2 buildings
- $\mathcal{D}_2$: 841 examples, maximum 4 buildings
- $\mathcal{D}_3$: 845 examples, maximum 6 buildings

Appendix B

Analysis of Image Regions

To determine if there are significant and consistent qualitative differences between the Ph-UNets and the UNets predictions, we perform a quantitative analysis on different regions of each image in the test set. Pfaff et al. 2020 showed that convolutional neural networks undersamples the area around the obstacle and the wake region. For fluid flow prediction, these two areas are the most interesting regions as this is where the changes in velocity are highest. To determine if inclusion of the physics-informed loss term mitigates this undersampling effect, we compare how UNet and Ph-UNet perform when evaluated for three different parts of the image:

- **Building**: The area of the buildings and the surrounding 15 pixels in each direction.
- **Wake**: The area to the right of the building-area, with an additional padding of 5 pixels to ensure the wake is contained in this region.
- **Other**: All regions that are not included in the Building-region or the Wake-region.

An example showing the three regions for the x-channel of a target image is shown in Figure B.1.

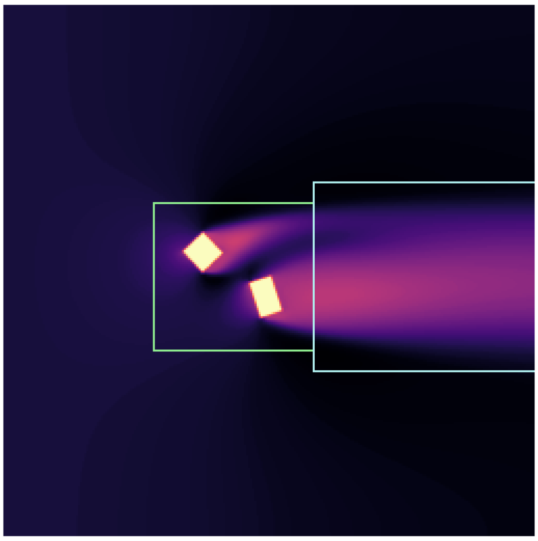

Figure B.1: Example of how the target images are split into regions for further evaluation, illustrated here with the channel containing the x-component of the velocity. The green frame encompasses the Building region, and the blue frame encompasses the Wake region. The remaining pixels lie in the Other region.

The L1 between the target image and the predicted image is then computed for each of these regions separately. The distributions of the L1 test loss for all three regions, as well as for the full image, are shown in Figure B.2 for $\mathcal{D}_1$, in Figure B.3 for $\mathcal{D}_2$, and in Figure B.4 for $\mathcal{D}_3$. Statistically significant differences between the test loss for Ph-UNet and UNet have a darker border.

For all three datasets, the three areas follow a similar distribution to what we saw when calculating the L1 for the entire image. The Building area might have slightly more similar distributions when comparing the Ph-UNets and the UNets. Overall, there appear to be no significant improvement in the predictive performance for any of the three image regions when including the physics-informed loss-component.

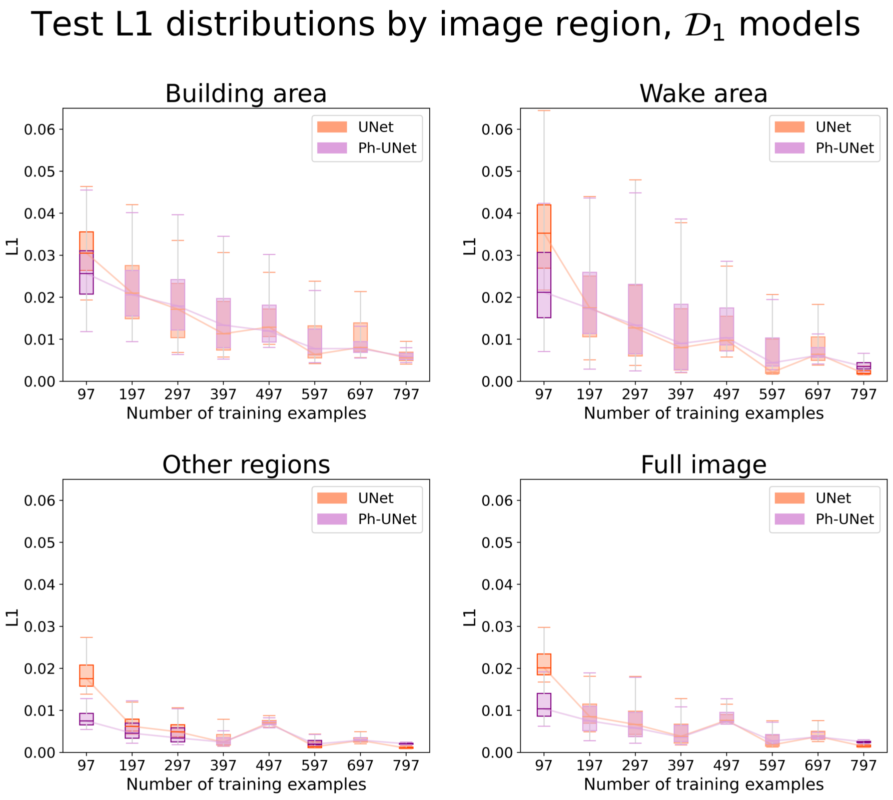

Figure B.2: Test L1 distribution for models trained on $\mathcal{D}_1$. When evaluating interesting regions separately, the Ph-UNets and UNets do not differ substantially in their predictions of the flow field.

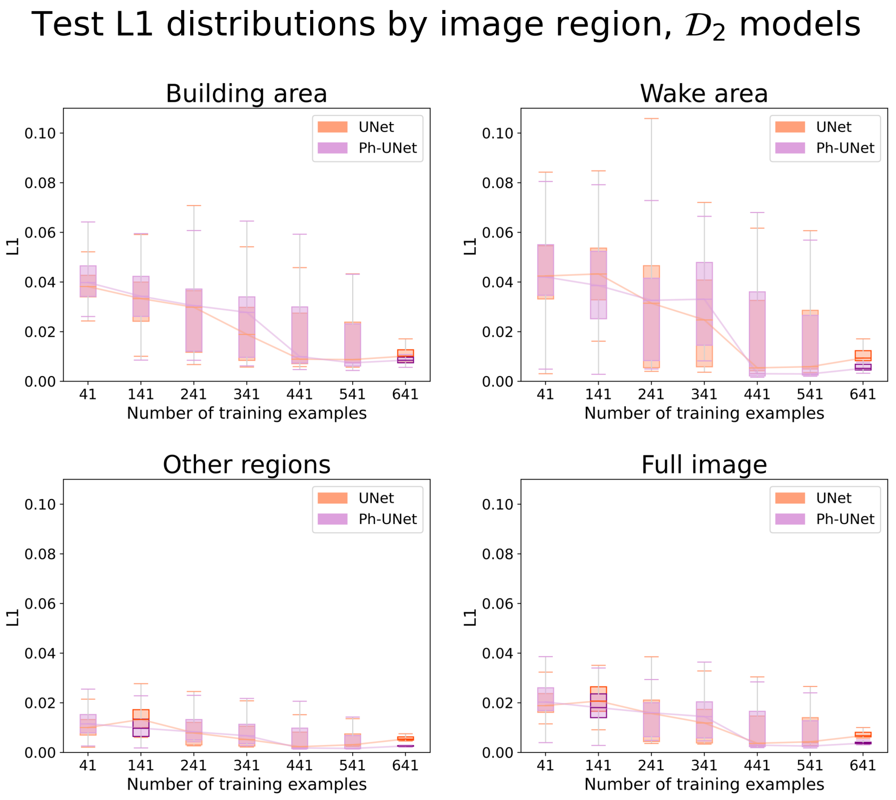

Figure B.3: Test L1 distribution for models trained on $\mathcal{D}_2$. When evaluating interesting regions separately, the Ph-UNets and UNets do not differ substantially in their predictions of the flow field.

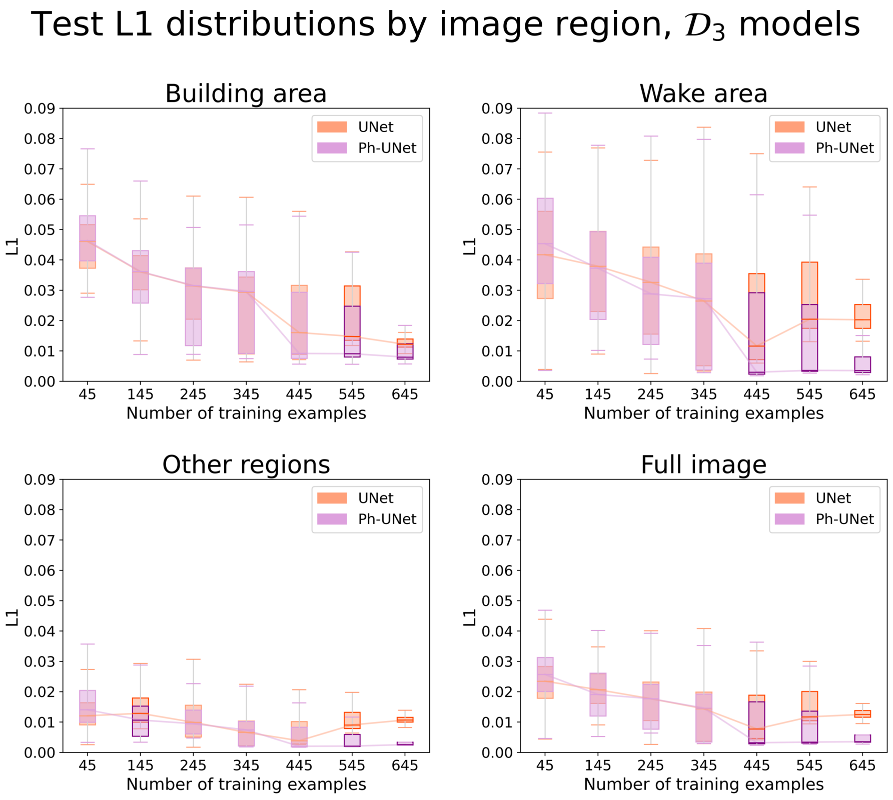

Figure B.4: Test L1 distribution for models trained on $\mathcal{D}_3$. When evaluating interesting regions separately, the Ph-UNets and UNets do not differ substantially in their predictions of the flow field.

Appendix C

Large Figures

We include scaled-up version of some result figures, to convey additional details.

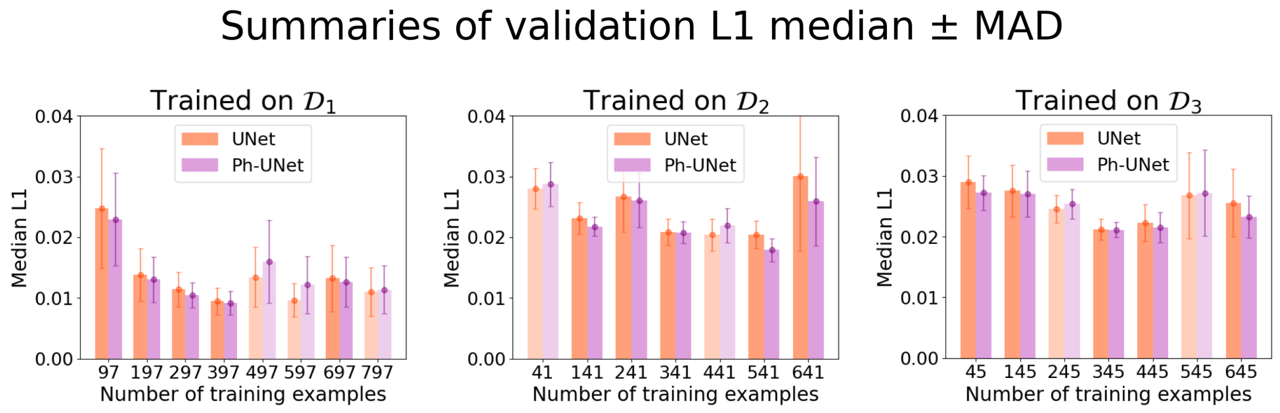

Figure C.1: Enlarged version of Figure 6.4 showing the median and MAD validation L1 from training for all models, by dataset. The 8 Ph-UNets and UNets trained on $\mathcal{D}_1$ are shown in the leftmost plot, the 7 pairs of models trained on $\mathcal{D}_2$ are in the middle plot, and the 7 pairs of models trained on $\mathcal{D}_3$ are to the right.

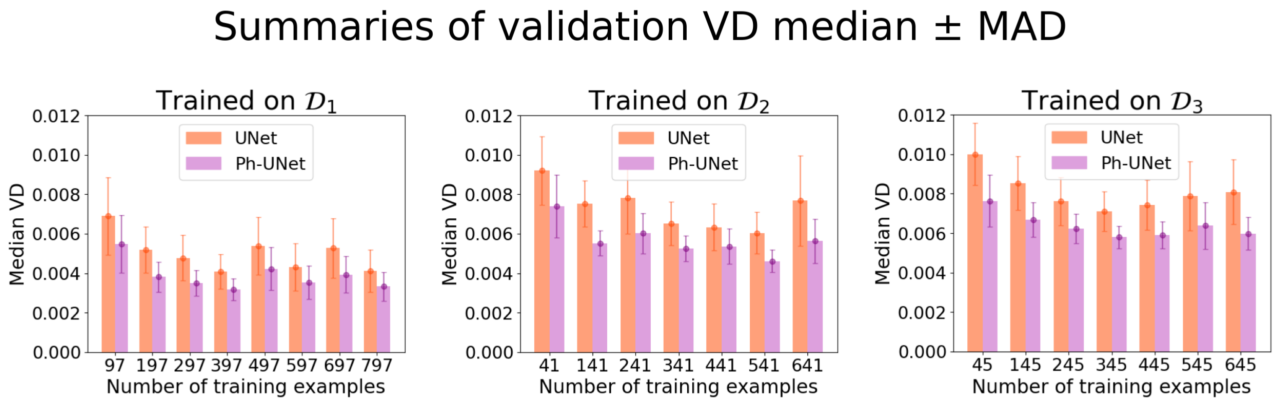

Figure C.2: Enlarged version of Figure 6.6 showing the median and MAD validation VD from training for all models, by dataset. The 8 Ph-UNets and UNets trained on $\mathcal{D}_1$ are shown in the leftmost plot, the 7 pairs of models trained on $\mathcal{D}_2$ are in the middle plot, and the 7 pairs of models trained on $\mathcal{D}_3$ are to the right.

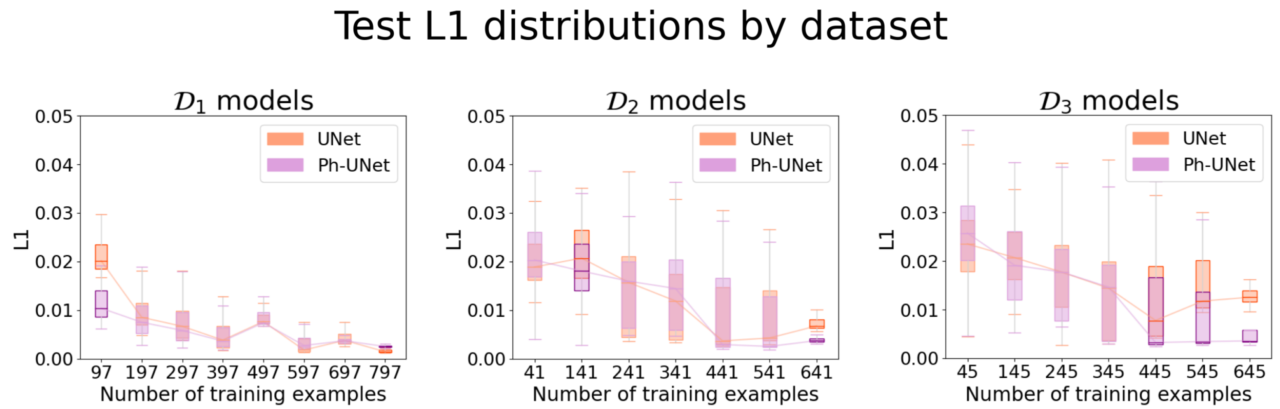

Figure C.3: Side-by-side comparison of Figure 6.12 - 6.14. The distributions of the test set L1 for $\mathcal{D}_1$ (left) are generally lower than those for $\mathcal{D}_2$ (middle) and $\mathcal{D}_3$ (right). This suggests that the transition from 2 to 4 buildings adds more complexity than the transition from 4 to 6 buildings.

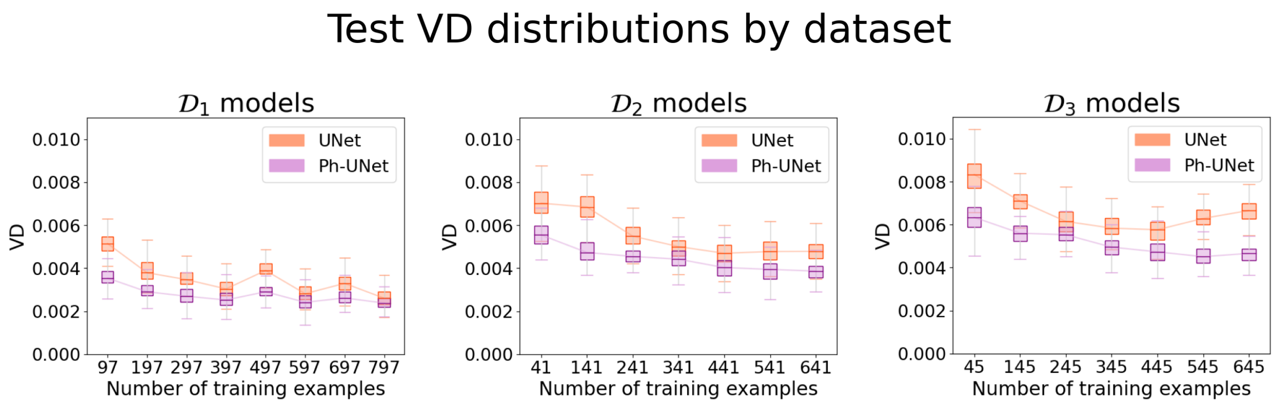

Figure C.4: A larger version of Figure 6.16. The distributions of the test set VD increase with increasing complexity.

NTNU
Norwegian University of
Science and Technology

NABLAFLOW