Learning From the von Kármán Vortex Street

by
Anthony Morast

A thesis submitted to the Graduate Division
in partial fulfillment of the requirements for the degree of

Computational Sciences and Robotics, M.S.

South Dakota School of Mines and Technology
Rapid City, South Dakota

Date Defended: April 22, 2019

Approved by:

Major Professor — Larry D. Pyeatt, Ph.D., Department of Mathematics and Computer Science	Date
Graduate Division Representative — Randy C. Hoover, Ph.D., Department of Electrical and Computer Engineering	Date
Committee Member — Kyle A. Caudle, Ph.D., Department of Mathematics and Computer Science	Date
Committee Member — Jeffrey S. McGough, Ph.D., Department of Mathematics and Computer Science	Date
Interim Department Head — R. Travis Kowalski, Ph.D., Department of Mathematics and Computer Science	Date
Dean of Graduate Education - Maribeth H. Price, Ph.D.	Date

Abstract

With the recent success of deep learning algorithms, there has been a re-emergence of applying deep learning techniques to partial differential equations (PDEs). Much of the recent work has been focused on learning solutions to PDEs given training data and the general form of the PDE. One largely unexplored area is learning the parameterization of PDEs when provided information about their solutions.

In this work, data from simulations of Navier-Stokes fluid flow equations are used to infer the Reynolds number that would parameterize the PDE. Fluid flow around a cylinder creates a wake, known as the von Kármán vortex street, in which different structures form depending on the Reynolds number. Using multi-layer perceptrons and convolutional neural networks these structures are used to determine the Reynolds number governing the fluid flow system.

Additionally, the structures found in the von Kármán vortex street are used to determine the location of objects in the fluid flow. In this case, two cylinders are placed in different locations upstream of an observer in an area unseen by the deep learning methods. The deep learning methods take solution data from the area downstream of these objects and use the information to infer an (x,y) coordinate for each object with respect to a global coordinate system. This thesis shows that deep learning algorithms can successfully learn the parameterization of the Navier-Stokes equations from the von Kármán vortex street and predict object locations with a high accuracy.

Acknowledgments

First and foremost, I wish to thank my wife, Nikita, for her unwavering support and patience through the long hours I spent away from her and our family over the last three years. During this time I was focused on my education causing her and our children to sacrifice by not receiving my full attention.

Thanks to several in academia, particularly my advisor Dr. Larry Pyeatt for his ability and willingness to support and guide me over the past few years. With his support and guidance I've found ample opportunities during my graduate studies that I would have otherwise missed. I would also like to thank others on my committee for their guidance especially in critiquing my thesis work and adding depth and utility to my research topic. Additionally, I am grateful for many other professors in the Department of Mathematics and Computer Science at the South Dakota School of Mines and Technology for offering advice not only pertaining to my academic pursuits but on other professional and personal matters as well.

Lastly, I'd like to thank my employer, NISC, for sponsoring my master's degree and allowing me to maintain full-time employment during my graduate studies. Without their support and flexibility it would have been far more difficult to complete my research and coursework.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Fluid Dynamics

Fluid dynamics is a field in science and engineering that describes the flow of fluids, i.e. liquids and gases, with many subfields such as aerodynamics and hydrodynamics. To solve problems in fluid dynamics, physical properties of the fluids such as velocity, pressure, density, and temperature are taken under consideration as functions of space and time. There are many practical applications of fluid dynamics including the calculation of forces and moments on aircraft, the flow of fluids through pipes and around bluff bodies, prediction of weather patterns, and understanding nebulae in interstellar space.

The foundational axioms of fluid dynamics are the conservation laws, in particular the conservation of mass, conservation of linear momentum, and conservation of energy. In addition to these conservation laws, fluids are assumed to obey the continuum assumption. That is, in fluid dynamics the focus is not on the properties of a single particle but rather on a point in space (relative to some fixed coordinate system). As time proceeds different fluids flow past these points in space and a history of the point's physical properties are kept rather than tracking the individual particles as they travel through space and time.

1.1.1 Fluid Dynamics Definitions

There are many types of fluids that flow in different ways. Below are the definitions of some requisite terms to build a vocabulary necessary for discussing fluid flow.

1.1.1.1 Pressure

Pressure in a static fluid is defined as the normal compressive force per unit area acting on a surface immersed in the fluid. The pressure in a fluid at rest is isotropic, i.e. the force acts with equal magnitude in all directions (independent of direction of measurement). In a dynamical situation there may be other shear and stress forces but pressure is still isotropic and defined in the same way as the static case. However, the pressure must be measured as the normal stress on an area which moves along locally with the fluid.

1.1.1.2 Compressible Flow

All fluids are compressible to some extent. However, the density of a gas changes much more readily with fluctuations in temperature and pressure than does the density of a liquid. Thus, for most intents and purposes a gas is considered compressible while liquids, whose change in density is often negligible, are considered incompressible. Sound propagation in liquids is one of the few situations in which the compressibility of liquids must be considered.

1.1.1.3 Newtonian Fluids

As shown by Isaac Newton, in common fluids, such as air and water, the stress due to viscous forces from neighboring parcels of fluids moving at different velocities is linearly related to the strain rate. Fluids obeying this linear relationship are called Newtonian fluids and the coefficient of proportionality is the fluid's viscosity. Fluids with more complicated non-linear stress-strain behavior are called non-Newtonian fluids.

1.1.1.4 Inviscid, Viscous, and Stokes Flow

Stokes flow, or creeping flow, is fluid flow in which the viscous forces are very strong compared to the inertial forces. In such cases the inertial forces are sometimes neglected. In contrast, when inertial forces have more effect on the velocity field than viscous (friction) forces the flow is often modeled as inviscid. Inviscid flow is an approximation in which viscosity is completely neglected. In cases involving solid boundaries the no-slip condition generates a region of large strain rate (the boundary layer) in which viscosity effects dominate and vorticity is generated. In such cases viscous flow equations, such as Navier-Stokes equations, must be used to calculate the net forces on the bodies.

1.1.1.5 Steady Flow

A flow that does not depend on time is known as a steady flow. Contrarily, a time dependent flow is known as unsteady or transient.

1.1.1.6 Laminar vs. Turbulent Flow

A laminar flow is a purely viscous flow in which the fluid flows in laminae or layers. A stream of dye inserted into a laminar flow will streak out into a thin line and always be composed of the same fluid particles. In contrast, turbulent flow has velocity components with random turbulent fluctuations imposed upon their mean values. Dye inserted into such a flow would quickly become tangled and mixed in with the fluid as it flows along. Turbulence, which is caused by excessive kinetic energy in parts of a fluid flow, is one of the least understood natural phenomena.

1.1.2 Navier-Stokes Equations

The Navier-Stokes (NS) equations, given by

$$\rho\Big(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\Big) = -\nabla p + \nabla\cdot\mathbf{T} + \mathbf{f}, \tag{1.1}$$

are a set of nonlinear partial differential equations that describe fluid flow. The equations are an extension of Euler's equations which can only be used to model inviscid fluid flow as there is no term to account for a fluid's viscosity. The extension of Euler's equations was presented by Claude-Louis Navier, a French mathematician and engineer, in 1822. Although his equations were correct, there were flaws in the reasoning of Navier's mathematical argument. Later, in 1842, George Stokes, an Irish mathematician, began looking at problems in fluid dynamics and, among other things, provided a correct derivation of Navier's equations thus giving them their name.

The Navier-Stokes equations are really Newton's second law of motion in disguise. The left-hand side of Equation 1.1 is the density of the fluid (i.e. its mass) multiplied by its change in velocity (i.e. its acceleration). The right-hand side of Equation 1.1 includes the different types of forces acting on a small region of the fluid. Therefore the equations are synonymous to $F = ma$ (Newton's second law of motion) for a small region of the fluid. In their unsimplified form, the equations do not have a general closed form solution so they're primarily used in computational fluid dynamics. There are a number of ways to simplify the equations, some of which allow simple fluid dynamics problems to be solved in closed form, but in general the equations are very difficult to solve analytically.

The equations model many important phenomena of scientific and engineering interest including weather, ocean currents, water flow in a pipe, and air flow around an aircraft's wing. However, these equations are difficult to solve without making many assumptions or relying on numerical methods to provide approximations of their solution. In fact, it's yet to be shown whether smooth solutions even exist for the three-dimensional system of equations. Moreover, proving the existence and smoothness of solutions to the Navier-Stokes equations is one of the Millennium Prize Problems put forth by the Clay Institute of Mathematics which offers a $1,000,000 reward for anyone who can prove or disprove such solutions exist [1].

1.1.3 The Reynolds Number

The Reynolds number is given by

$$Re = \frac{\rho V D}{\mu} \tag{1.2}$$

where ρ is the fluid density, μ is the fluid viscosity, V is the velocity of the fluid, and D is the diameter of a cylinder (in the case of fluid flow around a cylinder).

This number is a dimensionless quantity which represents the ratio of inertial effects to viscous effects in a fluid flow. The Reynolds number was discovered in 1883 by Osborne Reynolds while investigating fluid flow through a pipe. Reynolds discovered the transition between laminar and turbulent flows depends on fluid density, viscosity, velocity, and the internal diameter of the pipe [2]. By applying dimensional analysis, Rayleigh extended the Reynolds number to flow around spheres and cylinders [3].

A low Reynolds number ($Re \ll 1$) indicates that viscous forces are very strong compared to inertial forces. In such cases, inertial forces are sometimes neglected; this flow regime is called Stokes or creeping flow. In contrast, high Reynolds numbers ($Re \gg 1$) indicate the inertial effects have more effect on the velocity field than the viscous effects. For very high Reynolds numbers, the effects of viscosity can be eliminated completely reducing the Navier-Stokes equations (Equation 1.1) to the Euler equations, given by

$$\rho\Big(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\Big) = -\nabla p, \tag{1.3}$$

for which solutions exist.

1.2 Problem Statement

One largely unexplored area is inferring the parameterization of partial differential equations (PDEs) given numerical data or visualizations of the solutions to PDEs. For example, one might want to infer the Reynolds number governing a system modelled by the Navier-Stokes equations given images or numerical data of the solution at different time steps. In this thesis modern deep learning methods, such as convolutional neural networks and multi-layer perceptrons (MLPs), are used with visualizations and numerical snapshots of simulations of Navier-Stokes equations to determine the Reynolds number governing the system. Additionally, datasets similar to those used to infer the Reynolds number are used to determine the locations of objects upstream of an observer. In this case, two objects are placed in the geometry and an MLP is used to determine their (x, y) coordinates with respect to a global coordinate system.

An accurate approximation of the Reynolds number governing a fluid flow can be valuable in a variety of ways. For example, the NS equations with no body force, i.e. $f = 0$ in Equation 1.1, can be written as

$$\rho\frac{\partial \mathbf{v}}{\partial t} + \rho\mathbf{v}\cdot\nabla\mathbf{v} = -\nabla p + \mu\nabla^2\mathbf{v}. \tag{1.4}$$

This equation can be rendered nondimensional so that it does not depend directly on physical sizes. To obtain the nondimensional form substitute

$$\begin{aligned}\mathbf{v}' &= \frac{\mathbf{v}}{V}\\ p' &= \frac{p}{\rho V^2}\\ \frac{\partial}{\partial t'} &= \frac{L}{V}\frac{\partial}{\partial t}\\ \nabla' &= L\nabla\end{aligned}$$

where V and L are the characteristic velocity and length, respectively. This allows the Navier-Stokes equations to be rewritten as

$$\frac{\partial \mathbf{v}'}{\partial t'} = -\nabla' p' + \frac{\mu}{\rho L V}\nabla'^2\mathbf{v}' \tag{1.5}$$

where $\frac{\mu}{\rho LV} = \frac{1}{Re}$. Dropping the primes for readability and substituting the Reynolds number gives

$$\frac{\partial \mathbf{v}}{\partial t} = -\nabla p + \frac{1}{Re}\nabla^2\mathbf{v}. \tag{1.6}$$

Thus, knowing the Reynolds number allows for the parameterization of the Navier-Stokes equations. Additionally, for some large-scale systems where the Reynolds number might be valuable but cannot easily be determined, e.g. distant nebula or weather systems such as the Madden-Julian oscillation (MJO), providing images to a model similar to that proposed in this thesis could provide useful data for such systems.

Moreover, the ability to predict the onset of turbulent flow is an important design tool for many applications, making it a well studied problem. For example, predicting the transition in blunt-body flows while accounting for many different physical effects is handled by Wilcox [4] and methods for predicting the transition in pipe flow are studied by Soumerai [5, 6]. Turbulence transition around a circular cylinder in particular has also been studied extensively [7, 8, 9, 10, 11, 12]. Variations of the present work could be used as monitoring systems wherein data from a fluid flow is gathered throughout time and used to determine when the system is approaching its critical Reynolds number, i.e. when it's transitioning to turbulence.

The data used here is comprised of solutions of Navier-Stokes equations for fluid flow around a cylinder which creates a wake structure known as the von Kármán vortex street, discussed further in subsection 2.2.3. This provides additional application of this work as the patterns appearing in these wakes could perhaps provide information about objects in the fluid flow, e.g. the number of objects or their positions. This type of object detection has applications in many domains, e.g. on underwater and marine vehicles. In similar work by Bouffanis, Wymouth and Yu [13], pressure sensing is used to a similar effect.

The remainder of this work provides further detail on machine learning, fluid dynamics, and the proposed models. Chapter 2 gives the theoretical fundamentals of machine learning and fluid dynamics that are required to understand the models and experiments but might already be familiar to the reader. Chapter 3 discusses previous work applying machine learning to PDEs and how machine learning has been used in fluid dynamics. The details of the proposed models, data generation, and results are given in Chapter 4 while Chapter 5 discusses the results and future research directions.

Chapter 2

Theoretical Fundamentals

2.1 Machine Learning

Machine learning is an application of artificial intelligence used to automate model building, that is to extract patterns from raw data. To do so machine learning algorithms are designed to iteratively learn from data by minimizing error or maximizing the likelihood of their predictions being true. A subset of machine learning, deep learning, deals particularly with neural networks consisting of many layers of neurons. The interested reader is directed to the work of Goodfellow, Bengio, and Courville [14] for an in-depth introduction and reference to deep learning and machine learning principles.

Machine learning has allowed AI systems, that were once hard-coded with knowledge, to acquire knowledge on their own and continue to learn as new data becomes available. Machine learning algorithms have successfully, been applied to a broad range of disciplines including computer vision, speech and audio processing, natural language processing, robotics, bioinformatics and chemistry, video games, search engines, online advertising, finance, and recommender systems. Although deep learning was introduced to the machine learning community decades ago, its recent increase in popularity has improved the state-of-the-art in many of the fields mentioned above.

The machine learning algorithms discussed in this thesis are primarily supervised learning algorithms. Supervised learning algorithms are algorithms that learn to associate an input with a particular output when given a set of training data. Occasionally, outputs are hard to gather automatically so must be created by a human "supervisor", hence this class of algorithm's name. This is opposed to unsupervised learning algorithms which, typically, attempt to extract information from a distribution without the need of human labor to annotate the examples.

2.1.1 Artificial Neural Networks

Artificial neural networks (ANNs) are popular supervised learning models and the focal point of deep learning. The structure of these networks is based on the human brain where billions of nerve cells called neurons connected by axons consume external stimuli and produce a reaction, if stimulated beyond some threshold. Similarly ANNs are composed of many nodes which imitate neurons to which stimuli are carried via weighted connections.

The most popular ANN model is the feedforward ANN or multilayer perceptron (MLP). In this network architecture, information flows from the network input (represented as a vector $\mathbf{x}$), through intermediate computations (called hidden layers), and finally to the output (also represented as a vector $\mathbf{y}$). The example, $\mathbf{x}$, is passed from the input layer to the output layer where the gradient of a cost function is computed and the network weights are adjusted. The hidden layers of the network sum their weighted inputs, which are passed from a previous layer, and pass the result into an activation function which is typically nonlinear. Popular activation functions are the logistic sigmoid

$$f(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}}$$

which maps values to the range $(0, 1)$, and the hyperbolic tangent (tanh)

$$\tanh(\mathbf{x}) = \frac{\sinh(\mathbf{x})}{\cosh(\mathbf{x})} = \frac{1 - e^{-2\mathbf{x}}}{1 + e^{-2\mathbf{x}}}$$

producing values in the range $(-1, 1)$. Recently the rectified linear unit (ReLU), given by

$$f(\mathbf{x}) = \max\{0, \mathbf{x}\},$$

has become the activation function of choice since it offers many advantages over the others including being less prone to the vanishing gradient problem, being easier to

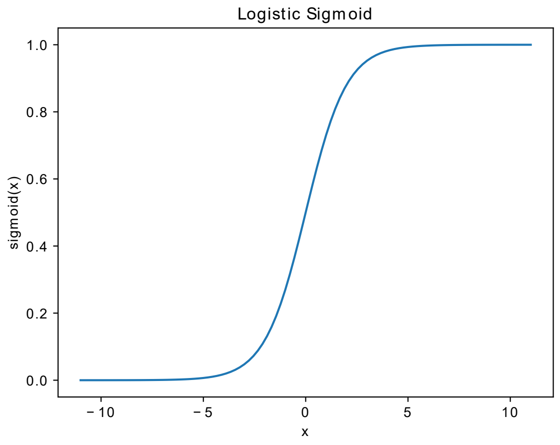

Figure 2.1: A graph showing logistic sigmoid from x=-10 to x=10.

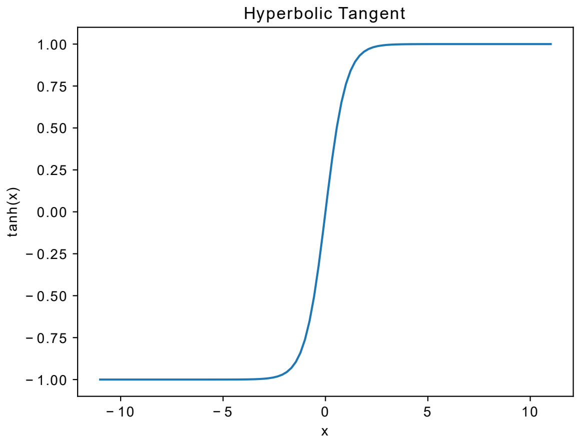

Figure 2.2: A graph showing hyperbolic tangent from x=-10 to x=10.

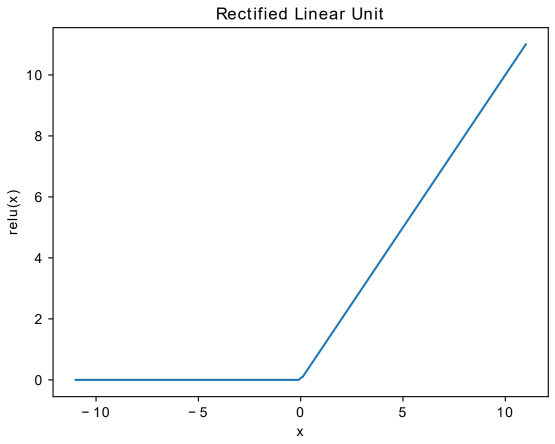

Figure 2.3: A graph showing rectified linear unit from x=-10 to x=10.

compute, and, in practice, networks using ReLU tend to show better convergence performance. Graphs of logistic sigmoid, hyperbolic tangent, and rectified linear unit can be seen in Figure 2.1, Figure 2.2, and Figure 2.3, respectively.

More technically, an MLP attempts to approximate a function $\mathbf{y} = f(\mathbf{x})$ as $\mathbf{y} = f^*(\mathbf{x})$ provided a set of data examples $\mathbf{X}$ and a set of labeled outputs $\mathbf{Y}$ where each $\mathbf{y} \in \mathbf{Y}$ corresponds to an input $\mathbf{x} \in \mathbf{X}$. The network is composed of an input layer, hidden layer(s), and an output layer. The input layer is determined by the number of features in the dataset and it is left to the network's designer to determine the amount and width of hidden layers and how the output will be represented. Neural networks with two or more hidden layers are known as deep neural networks.

The input vector, $\mathbf{x}$, is fed forward, through each layer, from the input layer to the output layer. Each connection between neurons is assigned a weight w_{ij} where i is the output of the previous neuron and j is the input of the next. The output of the k^{th} neuron is calculated as

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

where φ is the activation function.

The output layer produces an estimate of the desired response, $\hat{\mathbf{y}}$, which is used in the back-propagation algorithm to optimize the weights of the network. The output of the network, $\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{w})$, is fed into an objective function, E, which calculates some cost between the estimated and desired response. By finding the gradient of the cost function with respect to the weights,

$$\nabla E\left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_l}\right)$$

the weights of the network can be updated via

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \text{ for } i = 1, 2, ..., l$$

where η is the learning constant determined at implementation. Continuously feeding (new) examples into the network and adjusting the weights will, hopefully, find a near

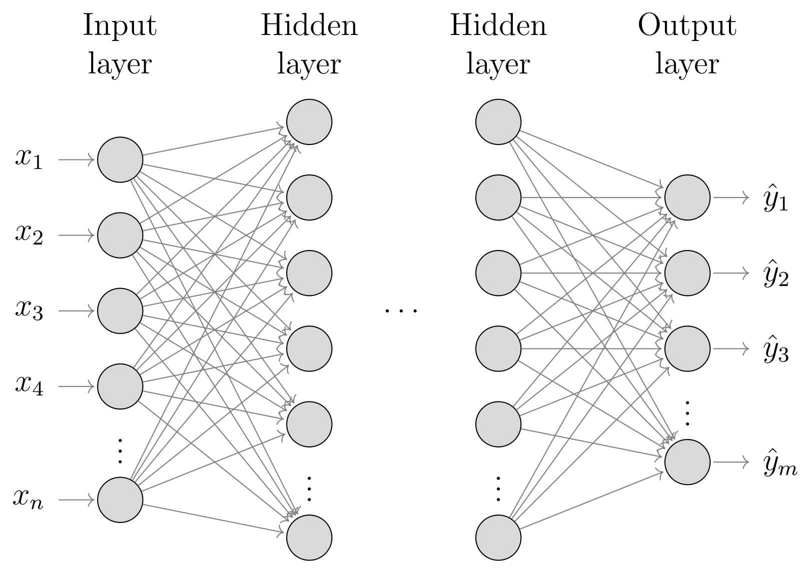

Figure 2.4: A model of the artificial neural network. The **x** vector is fed forward from the input layer to the output layer. The network can have zero to many hidden layers. Each hidden neuron takes a weighted sum of its inputs before passing the sum to a (typically) nonlinear activation function. Proper selection of activation functions, coupled with sufficiently wide ANNs, make ANNs universal approximators [15].

minimum value of the cost function, a process known as training. Once an ANN is trained with a sufficient amount of data, the network can be used for out-of-sample predictions. The general architecture of an ANN is depicted in Figure 2.4.

2.1.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a special type of ANN used to process data with a grid-like topology. Usually this refers to image data since the rows and columns of pixels form a 2-D grid. However, CNNs have also been applied in other domains such as time-series forecasting wherein the time-series is treated as a 1-D grid. CNNs get their name from the mathematical operation of convolution which is a special kind of linear operation. A CNN is just an ANN in which convolution replaces matrix multiplication in at least one the network's layers.

Convolution is a mathematical operation on two functions producing a third function that expresses how one is modified by the other. The operation is defined as the integral of the product of the two functions after one function is reversed and shifted, such as

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da. \tag{2.1}$$

In the case of CNNs, the function $x(t)$ in Equation 2.1 is the input to the neural network or convolutional layer, the function $w(t)$ is called the kernel, and the resulting function $s(t)$ is referred to as the feature map. In most machine learning applications the input is usually a multidimensional array of data and the kernel is a multidimensional array of parameters which are optimized by the learning algorithm.

Intuitively, a kernel of size n can be thought of as an $n \times n$ grid of parameters that 'slides over' the grid-like structure of the layer's input. At each location the kernel is slid over, a part of the kernel is aligned with a data value on the input, e.g. a pixel value of an image. The $n \times n$ values of the kernel are piecewise multiplied with the corresponding input data which are then summed to provide a value for $s(t)$, which is the feature map resulting from the operation. This is done for all potential positions of the kernel and the result is another image, reduced in dimensionality, created with the values of $s(t)$.

Typically, each layer of a CNN consists of three steps: several convolutions with different kernels, running each linear activation through a nonlinear activation function (e.g. ReLU), and finally applying a pooling operation. A pooling operation replaces the output of the layer at certain locations with summary statistics of nearby outputs. For example, a max pooling operation produces the maximum output within a rectangular neighborhood. In essence, pooling does something similar to the kernels described above but rather than changing the output values the purpose is to reduce the size of the feature map(s).

After the convolutional layers, the final feature maps are flattened into a one dimensional array to be used as input to an ANN. The final layers of the CNN have an architecture like that generalized in Figure 2.4. This network takes the values from the final feature maps and applies the operations described in subsection 2.1.1

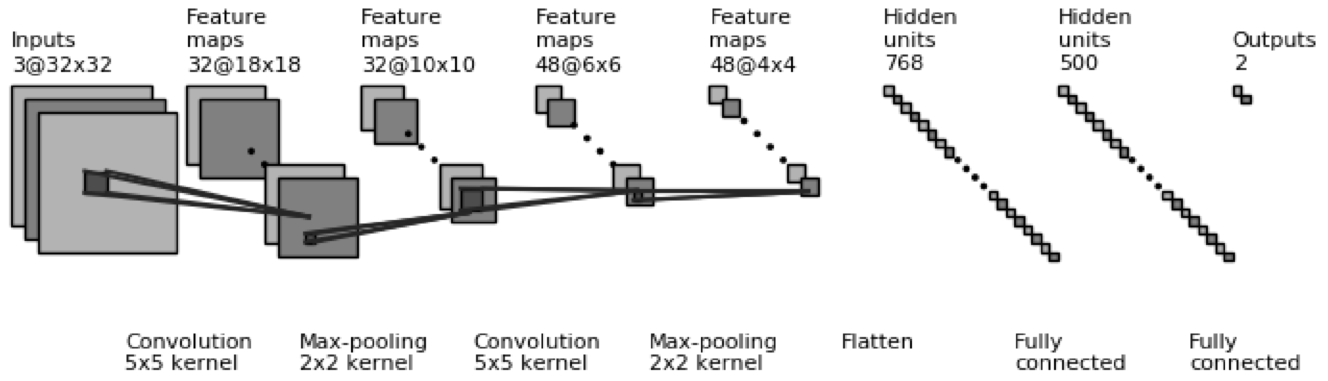

Figure 2.5: An example of a convolutional neural network. The network is given an input of three 2-D grids (for example the red, green, and blue channels of an image). These grids are operated on by a series of 5×5 kernels to create smaller dimensional feature maps. This is done until the final layer where the feature maps are flattened and input to an ANN.

to produce the output of the neural network. Note that, even though the final layers of the CNN are essentially an ANN ‘tacked on’ to the rest of the architecture, every layer, from the input to the final output, is trained with the same loss function and metrics. An example of a CNN is given in Figure 2.5.

There are three primary benefits to using CNNs over other neural network architectures: sparse interactions, parameter sharing, and equivariant representations. Sparse interactions, i.e. sparse connectivity, is accomplished by setting the kernel sizes smaller than the input size. This leads to storing fewer parameters which reduces memory requirements and improves the statistical efficiency of the model. Parameter sharing is achieved because each parameter member of the kernel is used at every position in the input, as opposed to ANNs where each element of the weight matrix is used exactly once. A final advantage of CNNs is that, due to their particular form of parameter sharing, the layers have a property called equivariance to translation. A function is equivariant if, when the input changes, the output changes in the same way.

2.1.3 Principal Component Analysis

Principal component analysis (PCA) is an unsupervised machine learning algorithm that learns a representation that is in a lower dimension than its input data and whose elements have no linear correlation. PCA can be used as an effective dimensionality reduction technique which preserves as much of the original information as possible (as measured by least-squares reconstruction error). PCA transforms the input data to a new coordinate system such that the greatest variance lies on the first coordinate, the second greatest variance on the second coordinate, etc.

Consider an $n \times m$ data matrix $\mathbf{X}$ with a mean of 0 (if the data matrix has non-zero mean it can easily be centered) where each row represents an observation and each column a feature of the dataset. The components of PCA are given by the eigenvectors of $\mathbf{X}^T\mathbf{X}$ with the first component being the eigenvector corresponding to the largest eigenvalue. The weight matrix $\mathbf{W}$ created with the eigenvalues of $\mathbf{X}^T\mathbf{X}$ (i.e. each eigenvector is a column in $\mathbf{W}$) provides a linear transformation taking each $x \in \mathbf{X}$ to the lower-dimensional space. This linear transformation is a rotation of the input space that aligns the principal axis of variance with the basis of the new representation.

2.2 Fluid Dynamics

2.2.1 Navier-Stokes Equations

As shown in subsection 1.1.2 the Navier-Stokes (NS) equations are given by

$$\rho\Big(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\Big) = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f} \tag{2.2}$$

where ρ is the density of the fluid, p is the pressure, $\mathbf{v}$ is a velocity field, $\mathbf{T}$ determines the stresses, and $\mathbf{f}$ represents body forces, i.e. forces that act throughout the entire region. The NS equations can be written in their nondimensional form with no body forces (i.e. $f = 0$ in Equation 2.2) as

$$\frac{\partial \mathbf{v}}{\partial t} = -\nabla p + \frac{1}{Re}\nabla^2 \mathbf{v}. \tag{2.3}$$

These equations were derived by Claude-Louis Navier, a French engineer and mathematician, in 1822. Navier amended Euler's equations (Equation 1.3) for fluid flow to cover more realistic cases in which a fluid has a non-zero viscosity. However, there were flaws in the mathematical reasoning put forth by Navier leading George Stokes, an Irish mathematician, to provide a correct derivation decades later in 1842. Thus, the equations came to be known as the Navier-Stokes equations.

A primary innovation of Euler, Navier, and Stokes is to consider the velocity field of a fluid flow rather than the movements of the individual particles in the fluid. Thus solutions to the NS equations and Euler's equation describe the velocity field of the fluid flow from which good approximations of actual flow patterns can be calculated. Moreover, this view of fluid flow causes the solutions to these sets of equations to be velocity fields or flow fields describing the fluid velocity at particular points in space and time.

The NS equations were derived from basic physical laws, in particular Newton's second law of motion which states that the rate of change of momentum of a body is directly proportional to the force applied, or, in terms of acceleration, force equals mass times acceleration ($F = ma$). In fact, Equation 2.2 is really Newton's second law in disguise with the LHS, $\rho\Big(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\Big)$, being the fluids mass (density) multiplied by its acceleration and the RHS, $-\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f}$, being the forces acting on the fluid. Combining the forces acting on the fluid (i.e. elastic stress caused by the fluids viscosity and pressure and body forces due to the acceleration of the fluid) and relating them to the movement of a fluid particle via Newton's second law produces the NS equations which, in this context can be seen as a statement of the law of conservation.

NS equations are especially useful in many different fields of science and engineering because they describe the physics of many different phenomena. Use of the equations is widespread and may be used to model the weather, ocean currents, fluid flow in pipes, flows around aircraft wings or automobiles, blood flow in the human body, and many other phenomena. In practice, since solutions to NS equations are difficult to obtain analytically, computational solvers are implemented to model these systems. Because of this, and due to the widespread and important applications of NS equations, a lot of work is done on finding solutions to NS equations and developing better numerical solvers so they can more easily be simulated in a computer.

Despite their wide range of practical applications, it has not been proven whether solutions to the NS equations always exist and, if they do, whether or not they are smooth (i.e. infinitely differentiable at all domain points). This problem is known as the Navier-Stokes existence and uniqueness problem and was deemed one of the seven most important open problems in mathematics making it a millennium prize problem [1]. Because of its high applicability, most solutions to the NS equations, in practice, are found via numerical approximations using algorithms like SIMPLE, SIMPLER, and MAC. However, for turbulent fluid flows this introduces new problems since computers can't handle infinitely complex calculations. Accurately modelling turbulence requires an impossibly fine computational grid to numerically solve the NS equations. Because of this statistical models are often used to model turbulence.

2.2.2 The Reynolds Number

The Reynolds number is the dimensionless quantity

$$Re = \frac{\rho V D}{\mu},$$

where ρ is a fluids density, V the fluid velocity, D the diameter of a circular bluff body, and μ the fluid viscosity. The Reynolds number represents the ratio of inertial forces to viscous forces. The quantity was discovered by Osborne Reynolds in 1883 who was studying the flow of fluids through pipes [2]. In particular Reynolds was investigating when the flow of fluids in pipes would transition from laminar flow to turbulent flow using dyed water flowing through a glass pipe at different velocities. Reynolds discovered the transition was dependent on the fluid density, viscosity, velocity, and pipe diameter, i.e. the quantities represented in the Reynolds number. This concept would later be extended to fluid flows around cylinders and spheres where the diameter becomes the external diameter of the body by British scientist John Strutt or Lord Rayleigh as he's more commonly called.

When the Reynolds number is low, $Re \ll 1$, the viscous forces are very strong compared to inertial forces. In this case, the inertial forces can be neglected; such a flow regime is known as Stokes flow. Contrarily, high Reynolds numbers, i.e. $Re \gg 1$, result from a fluid flow in which the velocity field is primarily affected by inertial forces. In this case the the viscosity can be ignored and simple equations, such as Euler's equations 1.3, can be used in place of the NS equations.

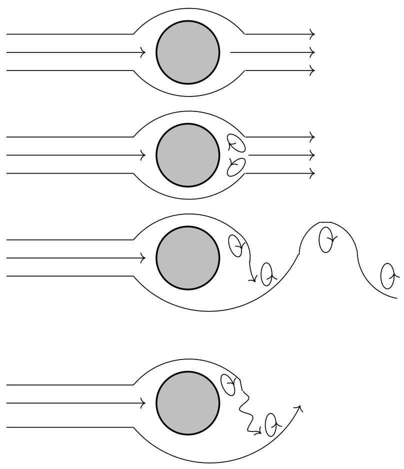
$Re < 5$: Regime of unseparated flow
$5-15 < Re < 40$: A fixed pair of Föppl vortices in cylinder wake
$40 < Re < 150$: Two regimes in which vortex street is laminar
$150 < Re < 300$: Transition range to turbulence in vortex
Transition to turbulence
$300 < Re < 3*10^5$: Vortex street is fully turbulent

Figure 2.6: Regimes of fluid flow across a smooth cylinder. For Reynolds numbers in the range $3 * 10^5 < Re < 3.5 \times 10^6$ the cylinder wake is narrow and disorganized do to turbulence. At $3.5 * 10^6 < Re$ a turbulent vortex street is re-established. These ranges are not considered thus not shown in this figure.

At certain Reynolds numbers the fluid flow system is transitioning from a laminar to a turbulent flow. Such Reynolds numbers are known as critical Reynolds numbers and have important implications in science and engineering. The ability to predict the laminar-turbulent transition is an important design tool for equipment such as piping systems and aircraft wings. The Reynolds number can also be used to determine dynamic similitude between different fluid flow cases. For example, scaling factors can be developed using the Reynolds number to relate the airflow over a model airplane's wing to a full-scale implementation.

2.2.3 The von Kármán Vortex Street

The von Kármán vortex street is a repeating pattern of vortices in the wake of a bluff body caused by a process known as vortex shedding. The vortex street is named after Theodore von Kármán , who discovered the pattern while working on his PhD dissertation. Von Kármán was inspired by another graduate student to investigate the cause of an oscillating wake behind a cylinder as fluid flows past. Von Kármán studied the stability of two configurations of the vortices. In one configuration the vortices formed in parallel rows and were also aligned vertically. In the other configuration, the vortices formed in parallel rows but were staggered vertically. Of the two, von Kármán found that only the latter resulted in a stable configuration, thus giving the vortex street its name.

The von Kármán vortex street only forms at certain Reynolds numbers and various Reynolds numbers will also create different types of wakes. As shown in Figure 2.6, for $Re < 5$ the flow around the cylinder remains unseparated, flowing smoothly around the cylinder. For higher Reynolds numbers, $5 < Re < 40$, two vortices spinning in opposite directions form directly behind the cylinder, such vortices are known as Föppl vortices. At Reynolds numbers of $40 < Re < 150$ a fully formed von Kármán vortex street develops and for higher Reynolds numbers the street shortens as faster moving vortices dissipate more quickly as seen in Figure 2.6 for $150 < Re < 2 \times 10^5$. Not shown in Figure 2.6 are two other configurations of the vortex street. At $3 \times 10^5 < Re < 3.5 \times 10^6$, the wake of the cylinder is fully turbulent causing it to be narrower and disorganized. However, for $3.5 \times 10^6 < Re$ the wake remains turbulent but a von Kármán vortex street is re-established. The latter two configurations are not considered in this thesis since the Reynolds numbers used were between 1 and 4,000.

The von Kármán vortex street has practical applications as engineers must take into account the vortex shedding when designing a wide range of structures, e.g. submarine periscopes and industrial chimneys and towers. For example, in the case of Ferrybridge Power Station C in 1965, the shedding of vortices caused the collapse of three concrete cooling towers that were clustered together. The vortex streets also can occur naturally in many large and small scale systems such as that shown in Figure 2.7. The von Kármán vortex street is the central theme of this thesis. The ideas introduced here stem from using machine learning to discern information about a fluid flow or the objects obstructing a fluid flow based on the patterns formed in the vortex street. Thus the patterns seen in Figure 2.6 are of particular importance as those are the patterns the deep learning models are intended to learn.

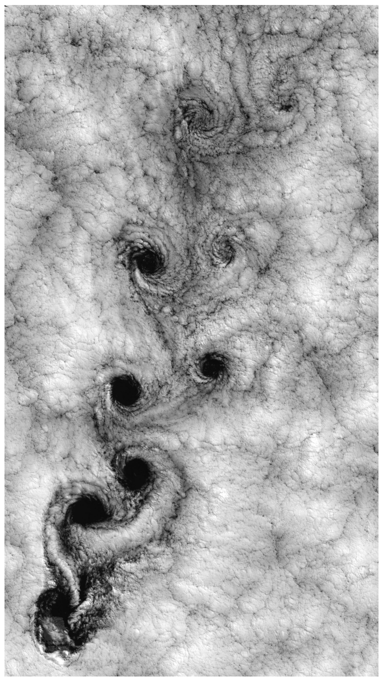

Figure 2.7: A von Kármán vortex street occurring naturally from wind flowing around the Juan Fernández Islands.

2.3 Error Measurement

The evaluation of machine learning models usually considers how close the predicted values are to the actual or observed values. Introduced in this section are the common measurements of errors that are used to help determine the goodness of a model in this thesis. Below, $\hat{\mathbf{y}}$ is the predicted value, $\mathbf{y}$ is the observed value, and n is the number of observations in the dataset.

Mean Absolute Error

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

Mean Squared Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n} \left(\hat{y}_i - y_i\right)^2$$

Chapter 3

Literature Review

3.1 Machine Learning and Partial Differential Equations

Some early work on using machine learning algorithms to approximate differential equations consists of transforming an ordinary differential equation into a linear system of equations via discretization of the domain. The solution of the linear system is then mapped onto the architecture of a Hopfield neural network. The minimization of this network's energy function provides the solutions to the system of differential equations [16, 17, 18]. Similar approaches are derived by using certain types of splines as basis functions to solve ordinary differential equations. Using splines, such as B-splines, the solution of a differential equation is found by solving a system of equations, not necessarily linear, to determine the parameters of the splines. This form of solution can be mapped onto the architecture of a feedforward neural network by replacing each spline with the sum of piecewise linear activation functions [19, 20]. These methods are used for solving ordinary differential equations and cannot easily be adapted to partial differential equations.

Lagaris, Likas, and Fotiadis [21] present a general method for solving both ODEs and PDEs using ANNs to approximate the differential equations' solutions resulting in a differentiable, closed analytic form solution. This method works by combining a term $A(\mathbf{x})$ having no adjustable parameters and meant solely to satisfy boundary or initial conditions, with the term $F(\mathbf{x}, N(\mathbf{x}; \mathbf{p}))$ which employs an ANN parameterized by $\mathbf{p}$ to approximate the solution giving the trial solution

$$\psi_t(\mathbf{x}) = A(\mathbf{x}) + F(\mathbf{x}, N(\mathbf{x}; \mathbf{p})). \tag{3.1}$$

Lagaris, Likas, and Papageorgiou [22] extend this work to boundary value problems with irregular (complex) boundary geometries. This is done with a system of two neural networks: an ANN to approximate the solution and a radial basis function neural network to satisfy the complex boundary which replaces $A(\mathbf{x})$ in Equation 3.1 [21].

Although sparsely studied between 2000 and 2017 there was a strong reemergence in 2017-2018 of using statistical learning techniques (deep learning, reinforcement learning, sparse regression, etc.) to solve problems modelled by PDEs. Recent work has largely been focused on solving two types of problems i) learning solutions to PDEs given training data and the form of the PDE and ii) inferring PDE parameters given training data but having no foreknowledge of the system dynamics. Much work in this area has been generalized to handle arbitrary differential equations but some is focused on solving a particular problem within a field.

Recent problem-centric work has been in applying machine learning methods to Navier-Stokes equations [23, 24]. Miyanawala and Jaiman [23] use a CNN for model reduction of the Navier-Stokes equations for fluid flow past different types of bluff-bodies. Baymani, Kerayechian, and Effati [24] extend the method presented by Lagaris, et al. [21] by applying it to a linearized form of Navier-Stokes known as Stokes equation, which produced significantly better results when compared to the Aman-Kerayechian method (a linear programming approach to the problem).

Weinan, Han, and Jentzen [25] developed *deep backwards stochastic differential equation (BSDE) solver* to approximate semilinear heat equations (a class of nonlinear PDEs). This methodology reformulates the nonlinear PDE into a stochastic control problem. The nonlinear Feyman-Kac formula is used to transform the PDE into an equivalent BSDE. The BSDE is viewed as a stochastic control problem with the gradient of the solution as the policy function. This high-dimensional policy function can be approximated by a deep neural network, as is done in deep reinforcement learning. This work is extended by Fuji, Takahashi, and Takahasi [26] who use asymptotic expansion as prior knowledge for the deep neural network which reduces the error of the approximations and improves the speed of convergence for deep BSDE solver.

Another extension to Weinan et al. [25] is provided by Beck, Becker, Grohs, Jaafari, and Jentzen [27] who apply the approximation algorithm to the special case of linear Kolmogorov PDEs. Khoo, Lu, and Ying [28] use CNNs to extract the states of a system governed by a PDE using the previous (predicted) state and the coefficients of the PDE. In this case, uncertainties in the system are built into the PDE as random coefficients. This method is applied to the Nonlinear Schrödinger equation subjected to a random potential field.

Sirignano and Spiliopoulos [29] use a neural network architecture inspired by the long short-term memory recurrent neural network to solve high-dimensional free boundary PDEs. The architecture and algorithm is named *Deep Galerkin Method (DGM)* since it is similar, in spirit, to Galerkin methods. DGM is meshfree, as meshes become infeasible in high-dimensional spaces, and instead randomly samples points in space and time. The training of the architecture uses not only the solution to the PDE but also the initial and boundary conditions in the objective function. DGM is applied to American and European option pricing PDEs with up to 200 dimensions (individual stocks considered) as well as a high-dimensional Hamilton-Jacobi-Bellman PDE and Burger's equation with satisfactory results.

Rudy et al. [30] consider equations of the form $u_t = \Theta(U, Q)\xi$ where U is spatial time series data of the PDE, Q is additional input data (e.g. known potential for Schrödinger's equations), and $\Theta(U, Q)$ is a matrix of candidate functions each consisting of linear and nonlinear combinations of the solution and its time and space derivatives. That is $\Theta(U, Q)$ could have the form

$$\Theta(U, Q) = \begin{bmatrix} 1 & U & U^2 & \dots & Q & \dots & U_x & UU_x & \dots \end{bmatrix}.$$

Each nonzero entry in the vector ξ corresponds to a term in the PDE. Thus, solving for ξ via sparse regression provides the terms in the PDE.

In a two-part treatise Raissi, Perdikaris, and Karniadakis [31, 32] introduce *physics informed neural networks (PINNs)*, a new paradigm for solving two types of problems: data-driven discovery of PDEs and data-driven solutions of PDEs. This is done by utilizing prior knowledge as a regularization agent that constrains the space of admissible solutions. The constraints are imposed via automatic differentiation [33] of a neural network with respect to its input coordinates and model parameters. PINNs can operate in the small data regime wherein most state-of-the-art machine learning algorithms lack robustness and fail to provide guarantees of convergence.

The first problem addressed by PINNs is that of computing data-driven solutions to PDEs of the general form

$$u_t + \mathcal{N}[u] = 0, x \in \Omega, t \in [0, T], \tag{3.2}$$

where $u(x, t)$ is the hidden solution, $\mathcal{N}$ is a nonlinear operator, and Ω is a subset of $\mathbb{R}^D$ [31]. This is done by training a neural network on data from the PDE's initial and boundary conditions to approximate a solution of the PDE, $u(x, t)$. A function

$$f := u_t + \mathcal{N}[u] \tag{3.3}$$

is constructed by taking the partial derivatives of the ANN with *Tensorflow's* [34] built-in automatic differentiation. Using f as part of the loss function to train $u(x, t)$ ensures the PINN adheres to the structure imposed by the PDE. The aforementioned setup is used for continuous time models but a similar derivation is given for discrete time models via Runge-Kutta methods.

In the second part of the PINN treatise the question of data-driven discovery of PDEs is investigated [32]. That is, given a small set of scattered and potentially noisy observations of the latent solution, what are the parameters that best describe the observed data? Again equations of the form given in Equation 3.2 are considered and a PINN constrained by Equation 3.3 is constructed. Because f, Equation 3.3, is constructed via automatic differentiation to model the PDE in question, the parameters to be learned are built into the NN. A derivation for both continuous and discrete time models is provided [31]. This method for learning the parameters of a PDE is applied to Burger's Equation, 2D Navier-Stokes, and Kortewegde Vries equation and shows promising results even given data with 10% noise corruption.

Raissi [35] extends the previous work [30, 31, 32] by incorporating a second neural network to approximate the nonlinear operator $\mathcal{N}$ (Equation 3.3) as a function of $u(x, t)$ and its derivatives. When applying the algorithm provided by Rudy et al. [30], the library of candidate functions will likely be insufficient when the dynamics are unknown. Additionally, this algorithm can only be used for PDEs with parameters appearing as coefficients. Approximating $\mathcal{N}$ with an ANN replaces the library of candidate functions with a universal approximator. Combining this second ANN with a PINN addresses the aforementioned issues and others introduced by Rudy et al. [30]. Further work by Raissi extends PINNs to forward-backward stochastic differential equations [36].

3.2 Machine Learning in Fluid Dynamics

As mentioned above, Miyanawala and Jaiman [23] and Baymani, Kerayechian, and Effati [24] apply deep learning to different problems involving the Navier-Stokes equations. Guo, Li, and Iorio [37] use CNNs to predict the velocity field of 2D and 3D non-uniform steady laminar flow. Their model successfully improves the efficiency of testing multiple designs that are quickly iterated over during the early stages of development when compared to traditional, computationally expensive fluid dynamics solvers. Farimani, Gomes, and Vijay [38] use generative adversarial networks (GANs) [39] to generate solutions to heat conduction and incompressible fluid flow without knowledge of the underlying equations. Rather than using ANNs to approximate the equations the GANs are used to directly generate solutions. This approach was able to predict temperature, velocity, and pressure fields with significant test accuracy.

Although promising for modeling high-dimensional complex systems, deep learning has been scarcely applied to modeling turbulent flows, as highlighted in a recent review of deep learning in fluid dynamics by Kutz [40]. Some work by Ling, Kurzawski, and Templeton [41] uses deep neural networks to improve Reynolds Averaged Navier-Stokes (RANS) models for turbulence. ANNs have been used to learn improved representations of the Reynolds stress anisotropy tensor from simulation data to create highly sought after improved RANS models [41]. Shallow ANNs are considered by Zhang and Duraisamy [42] to examine the efficacy of machine learning methods when applied to data-driven turbulence and transition modeling techniques introduced by Duraisamy, Zhang, and Singh [43].

Wang, et al. [44] used deep learning techniques to identify reduced order fluid dynamic models. In this work, recurrent neural networks are used for non-intrusive reduced order modelling (NIROM). NIROMs, in general, offer the potential to simulate dynamic systems with increased computational efficiency while maintaining accuracy. Duraisamy, et al. [45] provide an in-depth look at improving fluid system models, in particular turbulence models, via many different methods including advances in deep learning. The focus of this article is on improving Reynolds-averaged Navier-Stokes (RANS) models via bounding uncertainties in the models. The review covers many uses of machine learning including various approaches to predicting different features of the Reynolds stress tensor.

Yiu, et al. [46] use CNNs to detect extreme weather events such as tropical cyclones, atmospheric rivers, and weather fronts. The data used were images of several continuous spatial variables (pressure, temperature, precipitation, etc.) stacked into image patches. For example, in some example images a colormap is used to represent sea level pressure while solid vector lines show near surface wind distribution. The

CNNs used to detect patterns in these images achieve 89%-99% accuracy at classifying whether or not extreme weather events were present in the images. That is, the response of the CNNs was continuous and used to determine if the extreme weather event was present in the image or not. Recently Stöfer, Wu, Xiao, and Paterson [47] also used CNNs to extract features of interest from fluid flow fields using images of the fluid flows similar to those used in the present work.

Chapter 4

Method and Results

4.1 Overview

As shown in Chapter 3, deep learning has been extensively applied to solving partial differential equations. One largely unexplored area is inferring the parameterization of a PDE given data from its solution. Some applications are inferring Reynolds number for Navier-Stokes, the growth rate for Fisher's equation, or the thermal diffusivity constant of the heat equation. Additionally, for Navier-Stokes equations in particular, it is yet to be explored what further information can be determined from the von Kármán vortex street formed by fluid flowing past a bluff body. For example, by using the structures recurring in the vortex street one could attempt to determine the number of objects in a certain unseen area or the x and y coordinates of the objects in reference to some global coordinate system.

In the present work, data from simulations of fluid flows governed by the Navier-Stokes equations and characterized by a range of Reynolds numbers are used to train deep learning models for the tasks of parameter inference and object detection. In the case of object detection, the deep learning models will be trained to determine the x and y coordinates of two objects in the fluid flow. This chapter introduces the machine learning method, explains how the data was collected and prepared, and discusses the setup of the experiments.

4.1.1 Software

To create the datasets for the machine learning methods, two software packages were used: ANSYS Fluent [48] and Gerris [49]. To generate the dataset to infer the Reynolds number, a single geometry with an unchanging mesh was needed, as only the physical parameters of the fluid flow changed. Thus, ANSYS Fluent was used since

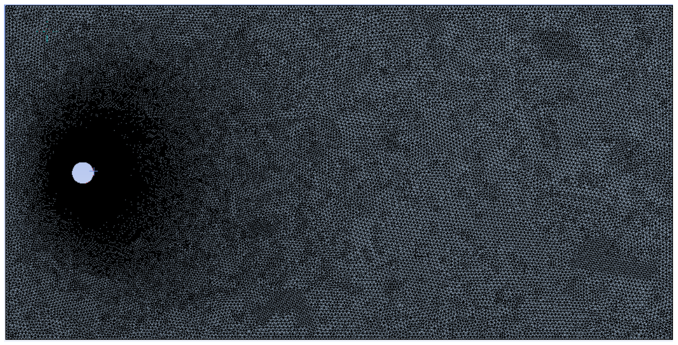

Figure 4.1: The geometry/mesh used to generate the data for inferring the Reynolds number.

it provides, perhaps, more accurate simulations as well as better performance with its GPU acceleration. When generating the dataset for object detection, the cylinders and mesh needed to be updated for each run of the simulation, but the physical parameters need not change. Therefore, Gerris was used due to its ability to easily place objects at any location in the domain and its adaptive meshing capabilities.

The models will be implemented in the Python programming language using the Keras high-level neural networks API [50]. The backend of Keras can be configured to use Theano, CNTK, or Tensorflow which are open-source symbolic tensor manipulation frameworks. In this case, due to its popularity and ease of use, Tensorflow [34] was used as the Keras backend. The Pandas data analysis library was used to reformat the data generated by Gerris and Fluent. Due to the large size of the resulting CSV files, the data was again reformatted into the HDF5 format, which is a file format designed to store and organize large amounts of data. Easily and efficiently reading and writing the HDF5 formatted files was accomplished with the h5py library.

4.2 Inferring the Reynolds Number

4.2.1 Collecting and Preparing Data

To create the dataset for parameter inference of the NS equations, ANSYS Fluent was used to simulate fluid flow around a circular cylinder at Reynolds numbers 1, 2, 4, 40, 100, 150, 250, 300, 400, 600, 800, 1000, 1200, 1400, 2900, and 4000. To easily

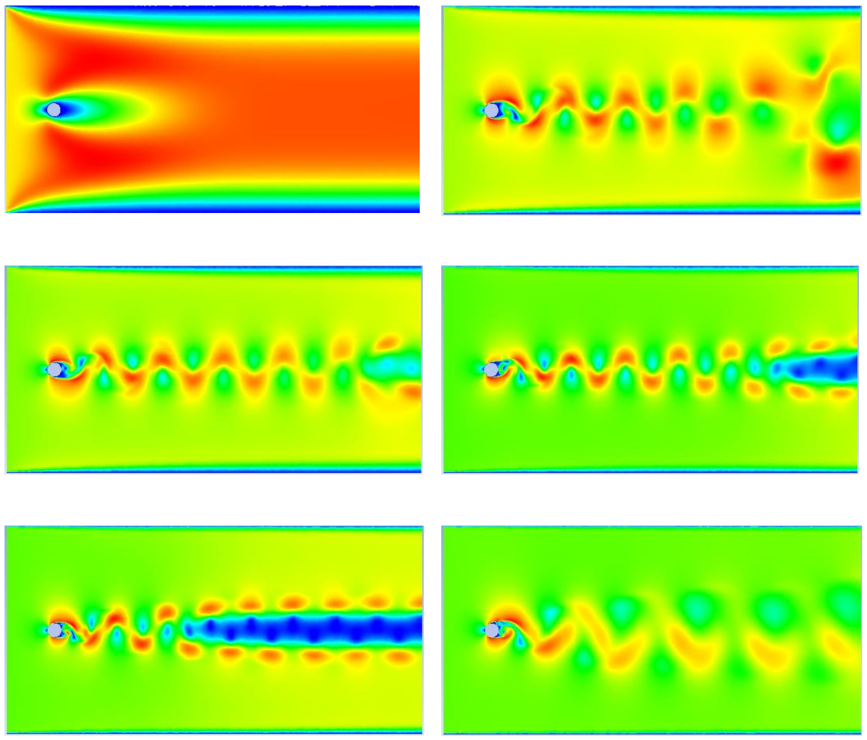

Figure 4.2: Velocity magnitude of the fluid flow for different Reynolds numbers at the same time step. From left to right and top to bottom the Reynolds numbers are 4, 100, 400, 1000, 2900, and 4000.

calculate the Reynolds numbers the diameter of the cylinder, fluid pressure (ρ), and fluid viscosity (μ) were all equal to 1. In this scenario the Reynolds number is equal to the fluid's velocity at the geometry's inlet since

$$\begin{aligned} Re &= \frac{\rho D U}{\mu} \\ &= \frac{1 \times 1 \times U}{1} \\ &= U \end{aligned}$$

The geometry/mesh in Figure 4.1 was used to generate this dataset. The geometry is 15 by 30 meters with a 1 meter cylinder near the left boundary and centered vertically. The left boundary is an inlet with the velocity set depending on the current Reynolds number being simulated. The right boundary is an outlet, and the top and bottom boundaries are both no-slip walls.

The mesh in Figure 4.1 has $\approx 80,000$ nodes for which time, pressure, x and y coordinates, and x and y velocities are saved every 10 time steps in CSV format. Images of the velocity magnitude are generated as well, examples are shown in Figure 4.2. The simulations were run for 3000 time steps producing 300 images and rows of numerical data. From the 16 simulations, this provides two datasets, one of images and one of numerical data, with 4,800 observations each. In the case of the numerical data, five variables are saved for each of the $80,000$ nodes giving a $400,000$ column dataset. However, because the data near the wall boundaries is not of interest, the dataset can be trimmed to include only the von Kármán vortex street. Trimming the numerical dataset reduces the column count to $\approx 240,000$. In both cases, the response variable of the datasets was the Reynolds number corresponding to the timestep and image or numerical data.

4.2.2 Experiments

For the images generated by ANSYS, two experiments were set up: using all of the data, and using data from matured systems. In the former, all 4,800 images were used to train the deep learning methods. Of these observations, 80% were used to train the neural networks and the remainder were used for validation. To create the dataset for the mature systems, only observations after time step 500 were used. The reason for using data only from mature systems is that better results can be expected from systems with a fully formed von Kármán vortex street as the simulations will be more distinguishable from one another. Again, this dataset is split using 80% for training and 20% for testing.

The numerical data was used to train MLPs using both the entire dataset and a rotated dataset created via principal component analysis (PCA) [51]. Time is highly represented in this dataset as the same time value is included for each node. This was done since time is an important feature of the dataset and having one time input among the 200,000+ other inputs would likely conceal the feature. Although the

Table 4.1: Best results for the 4 different datasets used for inferring the Reynolds numbers. The datasets are defined in subsection 4.2.2.

Method	**Dataset**	**Train MAE**	**Train MSE**	**Test MSE**	**Test MSE**
CNN	Images All	8.1162	140.2229	27.6371	15621.8274
CNN	Images After 500	52.3148	62318.4469	54.6317	56562.9193
20x20 ANN	Numerical PCA	7.0122	141.9751	28.6968	28525.9391
15x100 ANN	Numerical All	1.6917	6.7892	1.8000	8.6977

dataset was trimmed to exclude the regions near the geometry's border the input to the neural networks was still very high dimensional ($\approx 250,000$ inputs). Because of this, PCA was used to reduce the dimensionality of the dataset which produced a dataset with the first 61 principal components which explained 99.5% of the variance in the dataset. Reducing the dataset dimensionality via PCA can lead to better predictive capability and will certainly lead to faster training times.

4.2.3 Methods and Results

The image dataset generated by ANSYS Fluent was split into two, an **"Images All"** dataset and an **"Images After 500"** dataset as described in subsection 4.2.2. CNNs were trained to detect features in the image datasets that provide information about the Reynolds number. A satisfactory CNN architecture (after the input layer) for both datasets was found to be:

1. a convolutional layer with 128, 5×5 kernels and ReLU activation,
2. a convolutional layer with 128, 3×3 kernels and ReLU activation,
3. a max pooling layer using a 3×3 neighborhood,
4. a convolutional layer with 64, 3×3 kernels and ReLU activation,
5. a convolutional layer with 128, 3×3 kernels and ReLU activation, and
6. a max pooling layer using a 4×4 neighborhood.

The output of the final max pooling layer was flattened and fed into a fully connected ANN with ten hidden layers having ten nodes each having ReLU activation functions. Finally, the last hidden layer is fed into the single neuron output layer with no activation function.

Because this data is dependent on time, a data format for the CNN input was constructed which uses time as a fourth dimension to the images. That is, if the RGB values are read into three $m \times n$ arrays, giving a $3 \times m \times n$ tensor, a fourth $m \times n$ array is appended to the tensor with all entries being the time-step value. Thus, essentially, each pixel will store red, green, blue, and time information. This results in the input of the CNN being a tensor of size $4 \times m \times n$.

The results of using this CNN architecture to infer the Reynolds number from the image datasets are shown in Table 4.1. As seen in this table, using the entire dataset gives satisfactory results, having an average error of only 27.6371 on the test dataset which, in most situations is a reasonable approximation of the Reynolds number. That is, a system with a Reynolds number of 1100 will behave similarly to one with a Reynolds number of 1127. However, when using the dataset consisting of only mature systems, i.e. ones in which the vortex street should have already formed, results in the much higher error rate of 54.6317. Two potential reasons for this increased error could be the systems become more similar as time progresses making it more difficult to distinguish between different Reynolds numbers, or the reduced dataset had an insufficient amount of data to effectively train the CNN.

Many ANN architectures were considered for the numerical datasets generated by Fluent, as well as the dataset rotated via PCA as explained in subsection 4.2.2. The results for the best neural network architectures are summarized in Table 4.1. This table shows that when using the entire dataset (**“Numerical All”**), a neural network with 15 hidden layers and 100 nodes per layer is able to very accurately predict the Reynolds number with an average error of only 1.8 when predicting values in the unseen test dataset. For most purposes, this provides a good approximation, as systems with similar Reynolds numbers would be behave similarly.

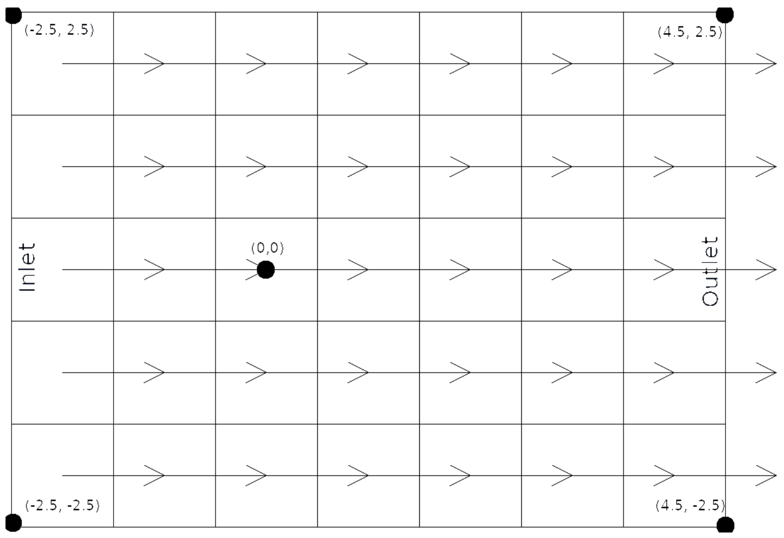

Figure 4.3: General geometry used by the Gerris simulations. The inlet is on the left of the geometry, the outlet is on the right, and the top and bottom boundaries are no-slip walls. Everything left of the origin (negative x values) is in the 'blind zone' meaning the data isn't seen by the machine learning algorithm.

When using the rotated dataset (**"Numerical PCA"**), a neural network having 20 hidden layers with 20 nodes in each hidden layer produces results on par with those of the CNN trained on the entire image database. Although this neural network performed worse than the neural network trained on the original dataset, the estimated Reynolds numbers are very close to the actual Reynolds numbers and would be sufficient in most cases. However, when using the rotated dataset it is important to note that the training and predicting times are much faster since the neural network has only 61 inputs compared to more than 250,000 inputs of the neural network using the original dataset. Additionally, the ANN trained on the rotated dataset was smaller, having 400 nodes in its hidden layers compared to 1500 for the ANN using the entire dataset. This allows the the neural network to more easily be loaded into GPUs with fewer computational resources, such as those on the NVIDIA Jetson.

Table 4.2: List of x and y coordinates for the two cylinders used in the Gerris geometries.

x_1	y_1	x_2	y_2
-2.1	0	-1.1	0
-1.5	0	-1.3	0
-2	0.5	-2	-0.5
-1	0.5	-1	-0.5
-2	1.5	-1	-1.5
-1.5	-1.5	-1	1.5
-1	-1.5	-1	1.5
-1.5	0	-1	0.1
-2	0	-1.5	0.1
-2	1	-1.8	1.1
-0.2	-1	-0.5	-1.1
-1.292	0.644	-0.707	1.914
-2.428	-2.128	-1.282	-1.679
-0.560	0.045	-0.123	0.323
-1.564	-0.708	-0.566	1.404
-1.447	1.541	-0.102	1.316
-1.142	-1.075	-1.810	1.903
-1.805	0.050	-1.468	0.431
-2.071	-1.788	-0.077	1.576

4.3 Object Detection

4.3.1 Collecting and Preparing Data

When using the Gerris software package, a geometry is defined by a sequence of boxes connected to each other. The origin is in the center of the first box defined and the others are placed with respect to the first box or boxes connected to the original box. Everything in Gerris is dimensionless and each box has unit length. In this particular case, 35 boxes are connected in a 5×7 grid, giving a geometry that is seven units long and five units wide. The geometry has an inlet as its left boundary, an outlet as its right boundary, and two no-slip walls for the top and bottom boundaries. For these simulations, everything to the left of the origin (negative x values) is considered 'in the blind zone', meaning the data is not used by the neural networks. The general geometry is shown in Figure 4.3

The geometry also contains two cylinders with diameter 0.1 units at varying x and y coordinates. Table 4.2 shows a complete list of the cylinder locations used.

Table 4.3: ANN architectures for object detection with results. The dimensions of the ANN are given as *hidden layers*×*nodes per hidden layer*, i.e. a 10×10 ANN has 10 hidden layers with 10 nodes in each layer.

Method	Train MAE	Train MSE	Test MAE	Test MSE
10x10 ANN	0.42856	0.3300	0.4525	0.3796
10x25 ANN	0.2614	0.1600	0.2054	0.2953
10x50 ANN	0.2125	0.1555	0.2261	0.1808
10x100 ANN	**0.1429**	**0.0773**	**0.1649**	**0.1173**
20x50 ANN	0.3847	0.3187	0.3999	0.3422
20x100 ANN	0.3847	0.3387	0.3998	0.3596

Initially, the velocity at each grid point is set to 2 units per second in the x-direction with fluid flowing into the geometry via the inlet at 2 units per second, also in the x-direction. Both the initial velocity and inlet velocity are 0 in the y-direction. The physical parameters of the fluid are $\mu = 0.0008$ for the viscosity and $\rho = 1$ for the pressure. With this setup, the Reynolds number is calculated as

$$\begin{aligned} Re &= \frac{\rho D U}{\mu} \\ &= \frac{1 * 0.1 * 2}{0.0008} \\ &= 250 \end{aligned}$$

An example of the initial mesh generated by Gerris is shown in Figure 4.4. In this case, the 35 separate squares (Figure 4.3) are refined to have 128 smaller sections for a total of 4,480 nodes in the mesh. Figure 4.4 also shows two cylinders at locations $(-2.1, 0)$ and $(-1.1, 0)$ with respect to the coordinate system shown in Figure 4.3. The simulation is run from time $t = 0$ seconds to $t = 5$ seconds. The x and y velocities, x and y locations, pressure, and time values are saved every 0.02 seconds at 56,726 (x, y) locations. This results in a dataset with 250 rows and 283,630 columns for each simulation. As shown in Table 4.2, 19 different simulations were run providing a dataset with 283,630 columns and 4,750 rows, of which 80% is used for training and 20% is used for validation/generalization testing.

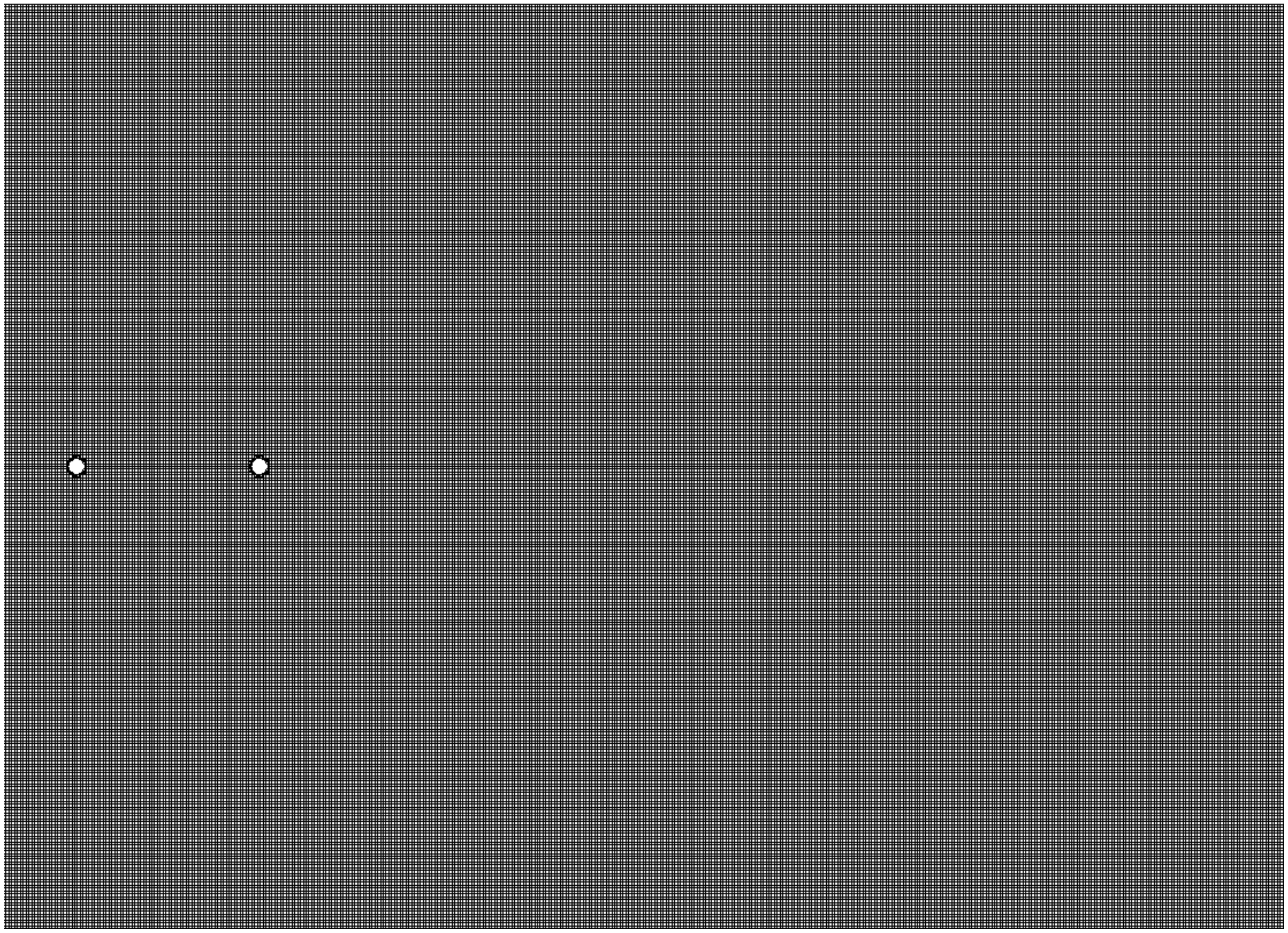

Figure 4.4: An example geometry wherein the two cylinders are inline w.r.t. the y-axis but slightly offset in the x-direction. This image also shows the initial grid used by Gerris.

Table 4.4: ANN architectures for object detection with results when trained on the rotated dataset using 400 components. Here, the dimensions of the ANN are given as *hidden layers×nodes per hidden layer*, i.e. a 10 × 10 ANN has 10 hidden layers with 10 nodes in each layer.

Method	**Train MAE**	**Train MSE**	**Test MSE**	**Test MSE**
10x10 ANN	0.0349	0.0047	0.1488	0.1336
10x25 ANN	0.0008	$1.6915\text{x}10^{-6}$	0.0213	0.0284
10x50 ANN	0.0008	$1.2974\text{x}10^{-6}$	0.0203	0.0315
10x100 ANN	**0.0003**	$\mathbf{3.8043\text{x}10^{-7}}$	**0.0273**	**0.0338**
20x50 ANN	0.0020	$2.2648\text{x}10^{-5}$	0.0278	0.0308
20x100 ANN	0.0041	0.0001	0.0154	0.0148

4.3.2 Methods and Results

Since the dataset for object detection contains only numerical data, i.e. there are no images or time-series data, an MLP was chosen as the machine learning method. Results from the most promising MLP architectures are presented in Table 4.3. In this table the MLP architecture is represented as *hidden layers×nodes per hidden layer*. A 10 × 10 MLP has 10 hidden layers with 10 nodes in each layer. As seen in Table 4.3, an MLP trained on the entire object detection dataset doesn't perform very well. This is likely due to an insufficient number of observations with respect to the complexity of the numerical data. Another factor at play could be the similarities between different simulations. That is, two separate simulations can have similar solutions at different times but have drastically different responses. However, the latter would likely apply to a modified version of the dataset as well.

Due to the disappointing results when training on the entire dataset, a dimensionality reduction technique known as truncated singular value decomposition (TSVD) was used. TSVD was chosen over other techniques due to its effectiveness on especially large datasets. When applying TSVD, 400 components were found to explain $\approx 97.99\%$ of the variance. Those components were used to produce a rotated dataset. Table 4.4 shows results from training MLPs with the same architectures as were used on the entire dataset, but with the rotated dataset.

As shown by Table 4.4, the MLPs trained on the rotated dataset perform significantly better than those trained on the entire dataset. In addition to performing better, the MLPs trained on the rotated dataset were faster to train since there were far fewer inputs. In both Table 4.3 and Table 4.4, the highlighted row represents the MLP with the best results on the training dataset. On the rotated dataset, the testing data reaches an MAE of 0.0273 units, meaning that, on average, the difference between the estimated and actual x and y components was ≈ 0.0273 units. However, the 10×100 MLP is likely overfit as the 20×100 MLP gets even better performance on the testing dataset. Putting the results into perspective, since the objects have a radius of 0.05 units, the estimated values were within the boundaries of the objects, but off-center.

Chapter 5

Conclusions and Future Work

5.1 Future Work

One interesting future direction is the application of this inverse problem to other partial differential equations. That is, using the methodology here to infer the parameterization of other partial differential equations some possibilities are Fisher's equation, the heat equation, or other population growth equations such as those used in medicine. Additionally, the data used here was produced via numerical approximation rather than using data from real physical systems. It would be interesting to see if the success of the results would carry over to noisy real-world data. Lastly, as seen in Figure 4.2 and Figure 2.6, the frequency of the vortices in the von Kármán vortex street differs based on the Reynolds number. The difference in vortex shedding frequency could be used to predict a different dimensionless number known as the Strouhal number which is commonly used as a measure of the predominant shedding frequency.

From a deep learning perspective, different methods or CNN architectures could be applied to these same datasets in an attempt to get better results. For example, the time component of the dataset could be dealt with in other ways when training the CNNs. Moreover, the data here represented time as an input to the machine learning algorithms rather than viewing the the simulations as a sequence. Convolutional recurrent neural networks could be used to learn how the system evolves over time and, potentially, give better predictions of the Reynolds number.

The object detection aspect could also be extended. In this thesis, two objects were placed upstream of an observer and the machine learning techniques attempted to predict their locations. It would be interesting to see if the methods would be as successful when more objects were placed in the fluid flow. Another unexplored area is determining the number of objects in the flow. In most cases the machine learning methods should be able to easily determine the number of objects, since there are patches of changing velocity created downstream of the objects. However, in some cases this would be more difficult, for example if the objects are aligned horizontally or if they are very close together. Furthermore, the only objects used here were circular cylinders in a channel. A prospective extension is to try different object shapes or different geometries which might affect the fluid flow and von Kármán vortex street.

5.2 Conclusion

In this thesis it has been shown that the structures in the von Kármán vortex street can be used to learn properties of the fluid flow and the objects in a fluid flow system. Using state-of-the-art machine learning techniques and solution data from the Navier-Stokes equations the Reynolds number and object locations were successfully determined for different fluid flows. The deep learning methods used were able to predict these properties to a high degree of accuracy that would be sufficient for most intents and purposes.

Being able to infer the Reynolds number to a high degree of accuracy allows for the parameterization of a dimensionless form of the NS equations, provides information about the physical properties of the fluid, and could be used to determine when a system is transitioning from laminar to turbulent flow. Although the images like those generated by ANSYS Fluent would be difficult to capture in a real-world system they could be generated given physical properties of the system as is done in [46]. Additionally, the von Kármán vortex street occurs naturally in many large scale systems which can be captured via satellite. Physical properties can be discerned from images like these, e.g. an approximation of the land mass shown in Figure 2.7. Finally, the work in this thesis, as well as that done by Liu, et al. [46], shows that a technique similar to the one developed here could be used to discern features in large-scale meteorological systems to predict the onset of extreme weather events.

In the case of object detection, the applications are more obvious. It has been shown here that the position of objects upstream of an observer can be accurately discerned given velocity data of the fluid flow. Additionally, a method similar to the one used here could be repurposed to predict other features of the objects in the fluid flow system. For example, a neural network could be used to determine the number of objects upstream of an observer, as well as an approximation of their positions.

Bibliography

[1] C. L. Fefferman, “Existence and smoothness of the navier-stokes equation,” *The millennium prize problems*, vol. 57, p. 67, 2006.

[2] O. Reynolds, “Xxix. an experimental investigation of the circumstances which determine whether the motion of water shall he direct or sinuous, and of the law of resistance in parallel channels,” *Philosophical Transactions of the Royal Society of London*, vol. 174, pp. 935–982, 1883.

[3] M. Zdravkovich, “Flow around circular cylinders volume 1: fundamentals,” *Oxford University Press, Oxford*, vol. 19, p. 185, 1997.

[4] D. C. Wilcox, “Turbulence-model transition predictions for blunt-body flows,” tech. rep., DCW INDUSTRIES SHERMAN OAKS CA, 1974.

[5] H. Soumerai, “On the application of an entropy maximizing principle in flow regime predictions,” *International communications in heat and mass transfer*, vol. 14, no. 3, pp. 303–312, 1987.

[6] H. Soumerai and B. E. Soumerai-Bourke, “Analytical method of predicting turbulence transition in pipe flow,” *Scientific reports*, vol. 2, p. 214, 2012.

[7] D. Okhotnikov, V. Molochnikov, A. Goltsman, and A. Mazo, “Mechanism of transition to turbulence in a circular cylinder wake in a channel,” 2017.

[8] A. Roshko, “On the development of turbulent wakes from vortex streets,” 1954.

[9] M. S. Bloor, “The transition to turbulence in the wake of a circular cylinder,” *Journal of Fluid Mechanics*, vol. 19, no. 2, pp. 290–304, 1964.

[10] C. Williamson, “The existence of two stages in the transition to three-dimensionality of a cylinder wake,” *The Physics of fluids*, vol. 31, no. 11, pp. 3165–3168, 1988.

[11] H. Persillon and M. Braza, “Physical analysis of the transition to turbulence in the wake of a circular cylinder by three-dimensional navier–stokes simulation,” *Journal of Fluid Mechanics*, vol. 365, pp. 23–88, 1998.

[12] H.-Q. Zhang, U. Fey, B. R. Noack, M. König, and H. Eckelmann, “On the transition of the cylinder wake,” *Physics of Fluids*, vol. 7, no. 4, pp. 779–794, 1995.

[13] R. Bouffanais, G. D. Weymouth, and D. K. Yue, “Hydrodynamic object recognition using pressure sensing,” in *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, p. rspa20100095, The Royal Society, 2010.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* The MIT Press, 2017.

[15] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[16] H. Lee and I. S. Kang, "Neural algorithm for solving differential equations," *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, 1990.

[17] L. Wang and J. Mendel, "Structured trainable networks for matrix algebra," in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pp. 125–132, IEEE, 1990.

[18] R. Yentis and M. Zaghloul, "Vlsi implementation of locally connected neural network for solving partial differential equations," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 43, no. 8, pp. 687–690, 1996.

[19] A. J. Meade Jr and A. A. Fernandez, "The numerical solution of linear ordinary differential equations by feedforward neural networks," *Mathematical and Computer Modelling*, vol. 19, no. 12, pp. 1–25, 1994.

[20] A. J. Meade Jr and A. A. Fernandez, "Solution of nonlinear ordinary differential equations by feedforward neural networks," *Mathematical and Computer Modelling*, vol. 20, no. 9, pp. 19–44, 1994.

[21] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[22] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou, "Neural-network methods for boundary value problems with irregular boundaries," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–1049, 2000.

[23] T. P. Miyanawala and R. K. Jaiman, "An efficient deep learning technique for the navier-stokes equations: Application to unsteady wake flow dynamics," *arXiv preprint arXiv:1710.09099*, 2017.

[24] M. Baymani, A. Kerayechian, and S. Effati, "Artificial neural networks approach for solving stokes problem," *Applied Mathematics*, vol. 1, no. 04, p. 288, 2010.

[25] E. Weinan, J. Han, and A. Jentzen, "Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations," *Communications in Mathematics and Statistics*, vol. 5, no. 4, pp. 349–380, 2017.

[26] M. Fujii, A. Takahashi, and M. Takahashi, "Asymptotic expansion as prior knowledge in deep learning method for high dimensional bsdes," *arXiv preprint arXiv:1710.07030*, 2017.

[27] C. Beck, S. Becker, P. Grohs, N. Jaafari, and A. Jentzen, "Solving stochastic differential equations and kolmogorov equations by means of deep learning," *arXiv preprint arXiv:1806.00421*, 2018.

[28] Y. Khoo, J. Lu, and L. Ying, "Solving parametric pde problems with artificial neural networks," *arXiv preprint arXiv:1707.03351*, 2017.

[29] J. Sirignano and K. Spiliopoulos, "Dgm: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339–1364, 2018.

[30] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Science Advances*, vol. 3, no. 4, p. e1602614, 2017.

[31] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10561*, 2017.

[32] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part ii): data-driven discovery of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10566*, 2017.

[33] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, "Automatic differentiation in machine learning: a survey," *CoRR*, vol. abs/1502.05767, 2015.

[34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[35] M. Raissi, "Deep hidden physics models: Deep learning of nonlinear partial differential equations," *Journal of Machine Learning Research*, vol. 19, no. 25, pp. 1–24, 2018.

[36] M. Raissi, "Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations," *arXiv preprint arXiv:1804.07010*, 2018.

[37] X. Guo, W. Li, and F. Iorio, "Convolutional neural networks for steady flow approximation," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 481–490, ACM, 2016.

[38] A. Barati Farimani, J. Gomes, and V. Pande, "Deep Learning Fluid Mechanics," in *APS Meeting Abstracts*, p. E31.004, Nov. 2017.

[39] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[40] J. N. Kutz, "Deep learning in fluid dynamics," *Journal of Fluid Mechanics*, vol. 814, pp. 1–4, 2017.

[41] J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *Journal of Fluid Mechanics*, vol. 807, pp. 155–166, 2016.

[42] Z. J. Zhang and K. Duraisamy, “Machine learning methods for data-driven turbulence modeling,” in *22nd AIAA Computational Fluid Dynamics Conference*, p. 2460, 2015.

[43] K. Duraisamy, Z. J. Zhang, and A. P. Singh, “New approaches in turbulence and transition modeling using data-driven techniques,” in *53rd AIAA Aerospace Sciences Meeting*, p. 1284, 2015.

[44] Z. Wang, D. Xiao, F. Fang, R. Govindan, C. Pain, and Y. Guo, “Model identification of reduced order fluid dynamics systems using deep learning,” *International Journal for Numerical Methods in Fluids*, vol. 86, 07 2017.

[45] K. Duraisamy, G. Iaccarino, and H. Xiao, “Turbulence modeling in the age of data,” *Annual Review of Fluid Mechanics*, vol. 51, no. 1, pp. 357–377, 2019.

[46] Y. Liu, E. Racah, J. Correa, A. Khosrowshahi, D. Lavers, K. Kunkel, M. Wehner, W. Collins, *et al.*, “Application of deep convolutional neural networks for detecting extreme weather in climate datasets,” *arXiv preprint arXiv:1605.01156*, 2016.

[47] C. M. Ströfer, J.-L. Wu, H. Xiao, and E. Paterson, “Data-driven, physics-based feature extraction from fluid flow fields using convolutional neural networks,” *COMMUNICATIONS IN COMPUTATIONAL PHYSICS*, vol. 25, no. 3, pp. 625–650, 2019.

[48] ANSYS, “Ansys fluent - cfd software — ansys,” 2016.

[49] S. Popinet, “Gerris: A tree-based adaptive solver for the incompressible euler equations in complex geometries,” *J. Comp. Phys*, vol. 190, pp. 572–600, 2003.

[50] F. Chollet *et al.*, “Keras.” `https://keras.io`, 2015.

[51] K. Pearson, “Liii. on lines and planes of closest fit to systems of points in space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

Vita

Anthony Morast graduated from the South Dakota School of Mines and Technology in 2015 with bachelor of science degrees in Computer Science and Applied and Computational Mathematics. During his undergraduate studies he worked as a software developer intern at CHR Solutions which led into his career at NISC upon graduating.

After spending some time at NISC he returned to the South Dakota School of Mines and Technology to pursue a Computational Sciences and Robotics M.S. with a graduation date of May 2019. His master's thesis combined ideas in fluid dynamics and partial differential equations with those in machine learning. While working on his master's degree Anthony also worked as a data analyst for SDSMT's Mathematics and Computer Science department, a graduate research assistant under Dr. Kyle Caudle, and a software developer for NISC.