FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Deep Learning for enhancing CFD methods

Nuno Gonçalo Lemos Oliveira

Mestrado em Engenharia Mecânica

Supervisor: Prof. Alexandre Miguel Prior Afonso

Co-Supervisor: Paulo Araújo da Cunha Sousa

July 30, 2024

Deep Learning for enhancing CFD methods

Nuno Gonçalo Lemos Oliveira

Mestrado em Engenharia Mecânica

July 30, 2024

Abstract

This document presents the work that led to the development and implementation of a deep learning model, specifically a multilayer perceptron (MLP), to predict the pressure field in a laminar flow past an object, between parallel plates. This study digged deep into the complexities of Computational Fluid Dynamics and the notable difficulties in accurately modeling pressure behaviors within confined flow scenarios. Traditional computational fluid dynamics approaches often face limitations by their expensive computational needs, since the grids required to capture the length and scale of fluid dynamics can be overwhelming. Therefore, this research endeavors to leverage the capabilities of deep learning to create a faster and accurate predictive model.

The proposed multilayer perceptron model is trained on a dataset comprising various confined flow configurations and associated pressure field computations. The training process involves optimizing the network architecture and learning parameters to improve the models performance and generalization capabilities.

Through rigorous evaluation and validation, the developed MLP model demonstrates promising predictive performance, showcasing its ability to generalize and make accurate predictions for different flow configurations.

The outcomes of this research not only offer a robust deep learning framework for addressing confined flow pressure calculation challenges but also provide valuable insights into the applicability of multilayer perceptrons in the domain of fluid dynamics. The developed model holds potential for enhancing predictive capabilities in various engineering and industrial applications where confined flow behaviors play a critical role.

Ackowledgments

I would like to say thank you to Professor Alexandre Afonso, my supervisor, and Paulo Soares, my co-supervisor, for the support and guidance that made this work possible.

I couldn't have reached this milestone without the support of my family. For everything they have done to give me the opportunities I had, I would like to show my deepest gratitude to my parents, Mr. Oliveira and Ms. Assunção. To my brother Pedro for keeping my head below the clouds, and my grandmother for dreaming by me years before I could dream, thank you.

Nuno Oliveira

"Shake off that erroneous notion that life is there and you're just gonna live in it. Versus embrace it, change it, improve it, make your mark upon it. (...) Once you learn that, you'll never be the same again"

Steve Jobs

Contents

List of Figures

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Networks
CFD	Computational Fluid dynamics
CNN	Convolutional Neural Networks
DL	Deep Learning
FDM	Finite Difference Method
FEM	Finite Element Method
FVM	Finite Volume Method
GAN	Generative Adversarial Networks
MLP	Multi Layer Perceptron
MSE	Mean-Squared Error
NN	Neural Network
N-S	Navier-Stokes
PC	Principal components
PCA	Principal component analysis
PDE	Partial Differential Equations
ReLU	Rectified Linear Unit
RMSE	Root-Mean-Squared Error
RNN	Recurrent Neural Networks
STDE	Standard-Deviation error

Chapter 1

Introduction

Artificial Intelligence (AI) is a rapidly expanding field and has been found to be useful in many scientific and engineering applications. Computational Fluid Dynamics (CFD) problems often have a strong optimization component (reducing drag, increasing lift), which is a work Neural Networks often excel at. Fluid mechanics has a long history of dealing with substantial amounts of data from large-scale numerical simulations, field measurements and other experiments. Recent advancements in measurement techniques and the surge of high-performance computing architectures have made the use of large amounts of data a reality in fluid mechanics research. This has led to the development and application of data-driven methods, as well as the integration of machine learning techniques to model and understand fluid dynamics. The field of fluid mechanics offers a fertile ground for the use of data-driven approaches, providing valuable insights and interpretations based on its strong connections to basic physics. Using AI algorithms that are developed to process large amounts of data a lot can be achieve in terms of reducing the computational cost of Fluid Dynamics simulations, improving turbulence closure modelling, and flow control. In this work Deep Learning (DL) methods were employed to improve CFD simulations by building a model that can replace the pressure prediction step in a CFD solver.

1.1 Outline

In Chapter 2 a CFD overview is given followed by a thorough description of Machine Learning (ML), finishing with a introduction to DL applied to CFD. In Chapter 3 the process of generating data and processing it is depicted, followed in 4 by the building of a pressure gradient surrogate model.

At last, in Chapter 5 the results are summarized and possible future works are discussed.

Chapter 2

State of the art

The work presented in this thesis brings together Artificial Intelligence with Computational Fluid Dynamics, two complex fields that have been exponentially growing in the past decades with the advance of computing power. In this Chapter, an overview of these themes will be provided so that the readers can fully grasp the work done and its purpose, as well as a review of the recent literature that merges Deep Learning and CFD together.

2.1 Introduction

Computational Fluid Dynamics consists of mathematically modeling complex fluid flow phenomena and solving them using numerical methods. It is a widely recognized and deeply rooted framework that enables the analysis of intricate physical systems. As analytical solutions are unavailable for the majority of flow scenarios, CFD plays an essential role in engineering and research.

A clear definition of AI has been the subject of extensive discussion and various perspectives trying to define it have been presented. The biggest difficulty comes from defining "intelligence" itself because there isn't a generally accepted definition of it nor tests that could be used to identify "intelligence" reliably. [5] The term Artificial Intelligence was first used in 1956 when a group of scientists made it the subject of an event. IBM describes it as "leveraging computers and machines to mimic the problem-solving and decision-making capabilities of the human mind". [7] . This field also overlaps with statistics since data preparation and analysis are always present and the use of statistical models is common. Artificial intelligence systems that improve through experience are the focus of Machine Learning (ML), which creates data-driven algorithms and models to enable automatic prediction and decision-making. There are several types of ML models, some common examples being Neural Networks (NNs), Decision Trees, Support Vector Machines and Linear Regression Analysis. NNs are inspired by the brain but use very simplified representations of neurons, far from the physiological complexity of the biological ones. Finally, Deep Learning arrives as a more specific subgroup of ML, consisting essentially in Neural Networks with multiple layers, which allows for high feature extraction from the data. The integration of deep learning in CFD methods has undeniable potential, since a lot of CFD problems boil down to optimization

(drag reduction, lift control), namely to accelerate simulations, improve turbulence modeling and develop enhanced reduced-order models.

2.2 Computational Fluid Dynamics

Independently of the application in mind, CFD always comprises these three steps:

1. Pre-processing: This initial phase involves defining the problem's physical boundaries and domain. It includes:
    - Creating a detailed geometric model of the fluid region;
    - Dividing this geometry into a mesh of smaller, discrete elements;
    - Specifying material properties and boundary conditions.
2. Simulation (or Processing): At this core stage, numerical methods are employed to solve the governing equations of fluid dynamics across the mesh. This typically involves:
    - Discretizing the equations to fit the mesh structure;
    - Iteratively solving these equations;
    - Monitoring convergence and adjusting parameters as needed.
3. Post-processing and results analysis: The final stage focuses on interpreting the simulation output. It usually encompasses:
    - Visualizing the results through various plots and animations;
    - Extracting quantitative data for specific variables of interest;
    - Conducting both qualitative and quantitative analysis to validate results and draw meaningful conclusions;
    - Comparing outcomes with experimental data or theoretical predictions when available.

The goal is to obtain the solutions of the governing equations of fluid flow and so these will be hereupon presented. The continuity equation, assuring there is conservation of mass:

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

Fluids motion can be described by Cauchy's momentum equation (here in Lagrangian form):

$$\frac{D\mathbf{u}}{Dt} = \frac{1}{\rho}\nabla \cdot \boldsymbol{\sigma} + \mathbf{f}_b \tag{2.2}$$

This equation can model the momentum transport of any continuum. Going further and using Newton's law of viscosity and Stoke's hypothesis we arrive at the Navier-Stokes equations for incompressible fluids and isothermal flow:

$$\rho \frac{D\mathbf{u}}{Dt} = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f}_b, \tag{2.3}$$

with $\frac{D\mathbf{u}}{Dt} = \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}$ representing the material derivative of velocity and $\mathbf{f}_b$ all the forces acting on the fluid, like gravity. Due to the second order nature of the N-S equations they can't be solved analytically so numerical methods are employed.

Numerical methods require the discretization of the spatial and temporal values to be carried in the transport equation. This discretization can be made with Finite Differences Method (FDM), Finite Volume Method (FVM) and Finite Element Method (FEM). For discretizing conservation laws the FVM is a good option since these laws can be integrated over volumes.

2.2.1 Pre-processing

The pre-processing consists of choosing the geometry and building the mesh. Boundary conditions should be defined in space and time. The mesh can be structured or unstructured, meaning having all the elements of the same shape and size or not respectively. Orthogonality, aspect ratio and resolution are important factors to evaluate in a mesh. When building unstructured meshes one should check the skewness of the cells because it slows down convergence and can also lead to incorrect fluxes near objects.

2.2.2 Simulation

The discretization method is applied to the computational mesh, transforming continuous differential equations into a system of algebraic equations. These discretized equations are then reformulated into a set of linear equations, facilitating numerical solution techniques. With a numerical solver we then can obtain a solution. One of the most widely used solvers in mechanical engineering applications, particularly in CFD, is PISO (Pressure-Implicit with Splitting of Operators). This algorithm was developed to solve the coupling between pressure and velocity by veryfing the satisfaction of both the Navier-Stokes equations and the continuity equation from the derivation of a pressure equation. PISO has been implemented in various solvers, such as pisoFoam, which is a transient solver for incompressible flow within the open-source CFD simulation software OpenFOAM. This solver uses the PISO algorithm to solve the relevant equations, making it suitable for simulating a wide range of fluid flow scenarios, namely laminar and turbulent, single-phase flows. The PISO algorithm is well known for its capability in handling transient simulations and became a fundamental tool for studying complex fluid dynamics problems. PISO splits the operator into an implicit momentum predictor and multiple explicit correction steps, instead of solving all the coupled equations iteratively. Usually there are needed only a few corrector steps to reach good accuracy.

In the PISO algorithm, the equation to be solved is the discretized pressure Poisson equation. This equation arises from forcing the continuity equation into the momentum equations and a similar procedure may be used to derive an analytical form by taking the divergence of equation

(2.3), yielding

$$\nabla \cdot \left(\rho \frac{D\mathbf{u}}{Dt} \right) = \nabla \cdot \left(-\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f}_b \right) \tag{2.4}$$

which after manipulation and neglecting the body forces, f_b, results in

$$\frac{\partial}{\partial t}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)+\left(\frac{\partial u}{\partial x}\right)^2+2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x}+u\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)+\left(\frac{\partial v}{\partial y}\right)^2+v\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)= -\frac{1}{\rho}\left(\frac{\partial^2 p}{\partial x^2}+\frac{\partial^2 p}{\partial y^2}\right)+\nu\left(\frac{\partial^2}{\partial x^2}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)+\frac{\partial^2}{\partial y^2}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)\right). \tag{2.5}$$

and considering the continuity equation for incompressible fluids, $\nabla \cdot u = 0$, which for 2D flows means that $\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$, one reaches the Poisson pressure equation for 2D flows:

$$-\frac{1}{\rho}\left(\frac{\partial^2 p}{\partial x^2}+\frac{\partial^2 p}{\partial y^2}\right)=\left(\frac{\partial u}{\partial x}\right)^2+2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x}+\left(\frac{\partial v}{\partial y}\right)^2. \tag{2.6}$$

The use of iterative methods to solve differential equations in CFD can be computationally expensive. However, employing a surrogate model to directly compute an approximate solution may help reduce this computational cost. Surrogate models, such as deep neural networks and active learning algorithms, have been proposed as efficient alternatives to high-fidelity numerical solvers for solving partial differential equations in various fields, including physics and engineering.

2.3 Machine Learning

In this section a thorough description of the fundamentals of machine learning will be provided, followed by the specificities of Deep Learning and what makes it a valuable asset for Computational Fluid Dynamics.

2.3.1 The Learning Algorithm

A ML algorithm is an algorithm that is capable to learn from data. How should we define learning? In its most broad definition, according to Mitchell [8] "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

2.3.1.1 Tasks

ML empowers us to address challenges beyond the scope of traditional, human-coded programs. Through the process of learning, systems acquire the capability to execute complex tasks effectively. Some of the most common ML tasks include [6]:

- Classification: involves assigning input data to specific categories. This is achieved by producing a function that maps the input to a category. The function can output a numeric code

identifying the category or a probability distribution over classes. Examples of classification tasks include object recognition, where an image is classified into specific objects.

- Regression: technique used to correlate between independent variables or features and a dependent variable or outcome. It is commonly employed for predictive modeling in machine learning, where algorithms are trained to predict continuous outcomes and requires labeled data.

- Transcription: involves converting unstructured data, such as speech or images, into discrete textual form. This process is widely used, for example in optical character recognition or speech recognition. Leverages automatic speech recognition machines to automate the conversion of speech into text, offering benefits such as accuracy, speed, and the ability to handle large volumes of transcription work.

- Structured output: involve generating a data structure with interconnected elements as the output. This encompasses various tasks such as the above mentioned transcription and translation but also parsing, pixel-wise image segmentation and image captioning.

- Anomaly detection: involves identifying unusual or atypical data points within a set of events or objects. This process is essential for various applications, such as fraud detection in credit card transactions and cybersecurity. The techniques used include density-based algorithms, cluster-based algorithms and collective outliers. Anomaly detection plays a crucial role in ensuring the security and integrity of systems.

- Synthesis/sampling: Synthetic data can be generated through various techniques, including random sampling from existing datasets and using generative models.

- Denoising: the algorithm is given a corrupted example and is required to predict the clean example. The goal is to train the algorithm to effectively distinguish between the corrupted and clean data, enabling it to generate accurate and reliable outputs from noisy inputs. Denoising techniques are also commonly applied in areas such as image recognition, speech processing, and data analysis to improve the quality and accuracy of the information being processed.

- Density estimation, or probability mass function estimation: involves training a ML algorithm to learn a function p_{model}, which represents a probability density function for continuous data or a probability mass function for discrete data. The algorithm must understand the structure of the data, including where examples cluster tightly and where they are unlikely to occur. This task is fundamental for capturing the underlying probability distribution of the data, enabling the algorithm to perform related tasks such as missing value imputation. Density estimation is achieved through parametric or non-parametric approaches such as kernel density estimation. It is widely used in, obviously, statistics but also anomaly detection and exploratory data analysis.

2.3.1.2 Performance Measure

Performance metrics are very important to evaluate the effectiveness of models and are specific to the task being performed by each system. For tasks such as classification, the most commonly used measures are accuracy and error rate. The choice of the performance measure is crucial and depends on the application and the desired behavior of the system. It can be challenging to select the most suitable metric, as it may be difficult to decide what should be measured or measuring it may be impractical in some cases.

A deeper explanation about a specific but common error metric will be made in the Neural Networks section.

2.3.1.3 Experience

Machine learning algorithms can be categorized by what kind of experience they are allowed to have during the learning process. This experience can be broadly defined as supervised or unsupervised.

Supervised Learning involves training a model on a labeled dataset, where each example is a pair consisting of an input object (typically a vector) and a desired output value. The algorithm learns to map the input to the output.

In Unsupervised Learning the model is given a dataset without explicit instructions on what to do with it. It must find structure within the data, like grouping or clustering of data points.

In Semi-Supervised Learning a small amount of labeled data and a large amount of unlabeled data are used. The algorithm learns from both to make predictions. One example that can be categorized by this approach are Generative Adversarial Networks (GANs). This type of neural network architecture is used for generative modeling. They consist of two neural network models: the generator, which produces synthetic data to imitate real data, and the discriminator, which distinguishes between genuine and generated instances. GANs are trained using adversarial training, where the generator and discriminator compete against each other, leading to the generation of realistic, high-quality outputs.

In Reinforcement Learning an agent learns to make decisions by taking actions in an environment to achieve maximum cumulative reward. Deep reinforcement learning enables an AI agent to acquire knowledge progressively through experimentation. By engaging with its surroundings and experiencing the consequences of its choices, the agent learns from both successes and failures, unlike the predefined datasets used in supervised learning. It is typically modeled as a Markov decision process.

Other types of learning methods are ensemble methods like boosting, bagging and stacking. This algorithms combine several ML models to improve predictive performance.

Each type of algorithm has its own characteristics and is suitable for different types of problems.

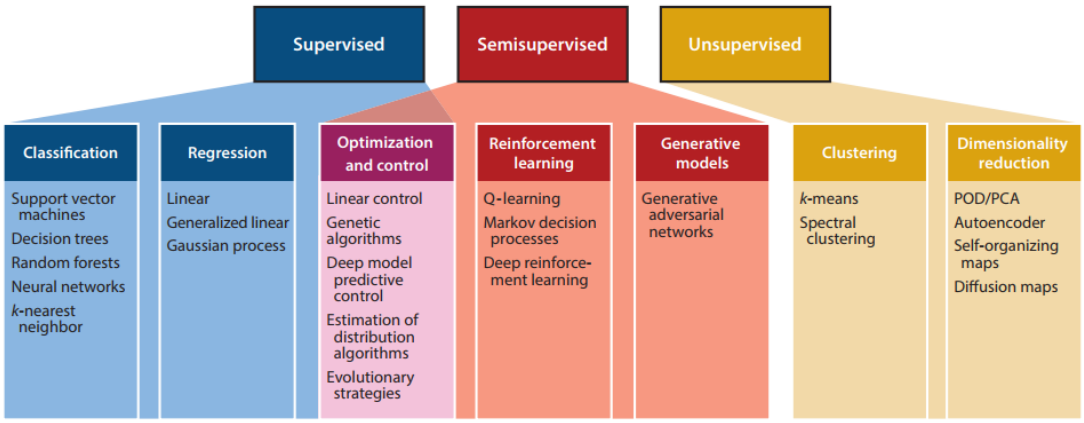

Figure 2.1: Machine Learning algorithms categorized by task and type of experience during training. From [2]

2.3.2 Training

A key goal in machine learning is developing models that can effectively handle novel, unfamiliar data. This ability to apply learned knowledge to new situations is referred to as generalization. This is distinct from a simple optimization problem, as it requires not only minimizing the error on the training set (training error) but also keeping the generalization error, or test error, low. The expected value of the error when applying the model to unseen data is the generalization error, and it is essential for the model to perform well on inputs drawn from the distribution it is expected to encounter in practice.

A machine learning model's effectiveness is measured by two key factors: how well it minimizes errors on training data, and how closely its performance on new data matches its training performance. Two common pitfalls can occur:

- Underfitting: The model fails to capture the underlying patterns in the training data, resulting in high error rates across both training and test sets.
- Overfitting: The model performs exceptionally well on training data but poorly on new data. This indicates that it has learned the noise and peculiarities of the training set rather than generalizable patterns.

Balancing these concerns is crucial for developing models that are both accurate and adaptable to new, unseen data. This are two of he main challenges in building a ML model.

What we can do to control this phenomena and achieve better generalization is alter the models capacity. Capacity in machine learning models refers to their ability to represent diverse functions. Low-capacity models may underfit by failing to capture complex data patterns, while high-capacity models risk overfitting by learning noise in the training data. Model training involves identifying appropriate risk functions or user-defined equivalents, then minimizing these functions using established optimization techniques. This process aims to find the optimal parameters that allow the model to perform its intended task effectively.

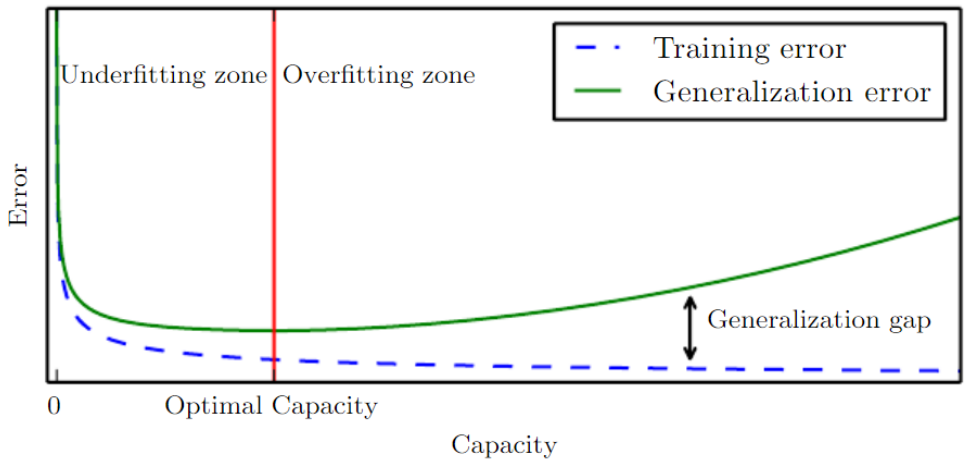

Figure 2.2: Typical relationship between capacity and error

2.4 Deep Learning for Computational Fluid Dynamics

The computational simulation of fluid dynamics is a crucial tool in science and engineering. However, numerically solving partial differential equations (PDEs) still faces challenges such as stability constraints, high computational costs, and inaccuracies in modeling turbulence terms. Deep learning involves using neural networks with multiple hidden layers, allowing them to learn complex data representations. Its naming comes from 1940's but only in the last few decades it became feasible to fully apply it. The increase in computing power and worlds digitalization, making huge datasets of all types available, lead to deeper exploration of this methodology and it has already achieved incredible performances in tasks such as image classification, speech recognition and natural language processing. In recent years, there has been a growing interest in applying deep learning techniques to fluid mechanics research. This approach is not only aimed at improving computational efficiency but also at enhancing the accuracy of various applications in the field. Some of the key areas where deep learning methods are being expanded include:

- Direct Numerical Simulation (DNS) and Turbulence Modeling: Deep learning models are being used to augment DNS and traditional turbulence models in CFD, aiming to improve their accuracy, finding new solutions to the closure problem or reducing computing time.
- Reduced-Order Modeling: There is a focus on developing enhanced reduced-order models using machine learning, which can potentially offer more efficient representations of fluid dynamics phenomena.
- Super-Resolution Techniques: Borrowed from computer vision and image recognition, super-resolution imaging refers to the class of techniques that aims at obtaining a high resolution image output from a low-resolution image [3] and is also being investigated for its potential applications in fluid mechanics research.

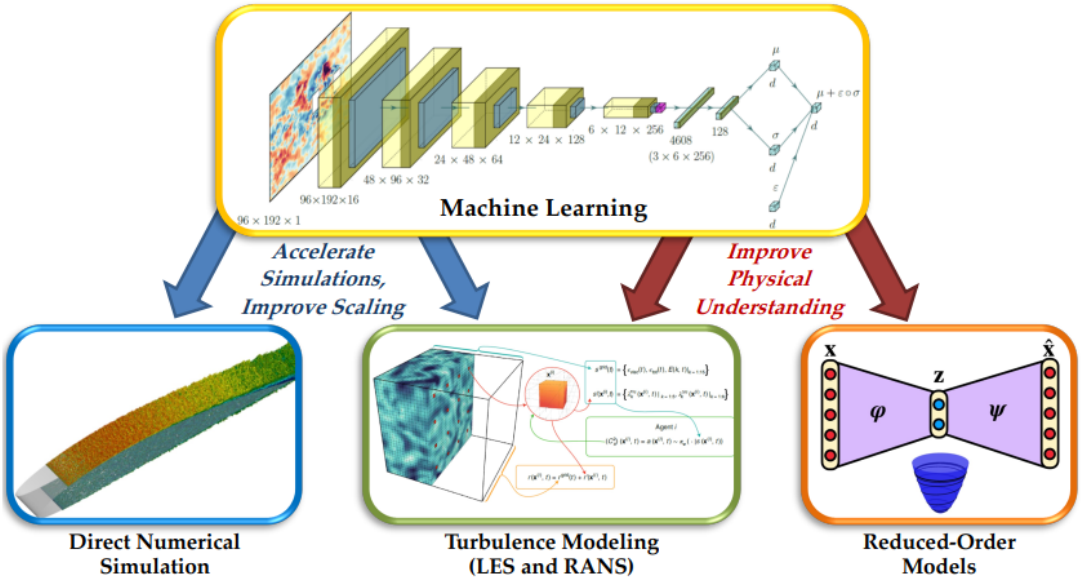

Figure 2.3: Most relevant areas where DL can enhance CFD, as per [9]

There are also several different approaches to leveraging DL for CFD applications. Surrogate modelling can be done to partially assist CFD simulations, the models can be Physics-Informed Neural Networks (PINN) or purely data driven neural networks. This interdisciplinary approach holds the potential to greatly influence the future of fluid mechanics research and its practical applications.

The most common types of architectures in CFD applications are:

- Multilayer Perceptron (MLP): Feedforward Neural Network with several layers, trained using the backpropagation method. Most "standard" neural network.
- Convolutional Neural Network (CNN): Regularized Feedforward Neural Network that use the convolution mathematical operation. This operation consists in sliding a filter (kernel) over an input (e.g., an image) to produce a feature map, capturing local patterns such as edges and textures. Excels in processing image and video.
- Recurrent Neural Networks (RNN): Characterized by the bidirectionallity of flow information between layers, this type of NN is optimal to analyze sequential data.

2.4.1 The Multilayer Perceptron

The basic unit of the MLP is a neuron, or node, that can be represented visually by fig, whose output is given by the expression:

$$y = f\left(\sum_{i=1}^{N} x_i w_i + b_i\right) \tag{2.7}$$

where f represents the activation function, w_i the weight of the neuron i and b_i the bias of the present neuron.

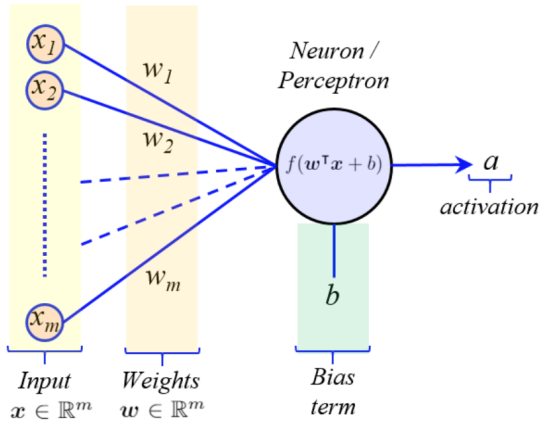

Figure 2.4: Neuron representation with the layer before, weights, bias and activation. From [1]

Neural networks consist of interconnected nodes organized into input, hidden, and output layers. The connections between nodes, called weights, are learned during training to minimize prediction errors. This type of algorithms approximate the relationship between input and output data by learning a function that maps the input data to the output data, working like an universal approximator. This function, usually denoted as $y = f(x)$, is determined by the weights and biases of the trained neural network, which are learned during training. The universal approximation theorem states that a feedforward network with a linear output layer, provided that the network is given enough hidden units, can be used to approximate any continuous function. This property makes neural networks valuable for function approximation, as they can learn to estimate an unknown underlying function using historical or available observations from the domain. The algorithm's goal is to develop a function f through training that can then be applied to new, unseen data to generate accurate predictions.

A general representation of a deep neural network with several connected layers follows:

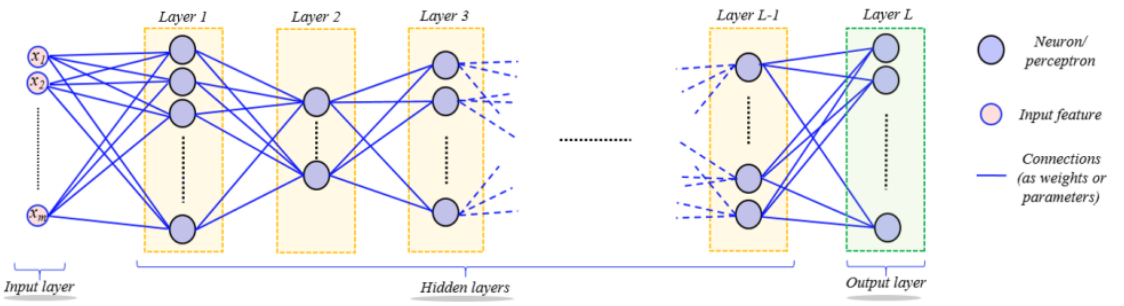

Figure 2.5: Deep Neural Network. From [1]

2.4.1.1 Feedforward propagation and backpropagation

The training of the MLP can be clearly divided into 4 steps, repeated over multiple epochs to gradually reduce the loss:

- Forward Pass: During this step, the input data is propagated through the network, and the output is computed based on the current weights and biases.
- Error Computation: The difference between the predicted output and the actual output is calculated using a loss function, such as mean squared error or cross-entropy loss.
- Backpropagation: The error is then propagated backward through the network using the backpropagation algorithm. This involves computing the gradient of the loss function with respect to the weights and biases of the network.
- Weight Update: The weights and biases of the network are updated using an optimization algorithm, such as stochastic gradient descent, based on the computed gradients.

2.4.1.2 Activation Functions

The activation function "links" the weighted sums of units in a layer to the values of units in the succeeding layer. The most important feature that these functions bring is adding non-linearity to the neural network. The most common are:

- Rectified Linear Unit (ReLU): returns 0 for negative inputs and the input value for positive inputs.
$$f(x) = max(0,x) \tag{2.8}$$
- Hyperbolic Tangent (tanh): squashes the input into the range (-1, 1).
$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.9}$$
- Sigmoid: squashes the input into the range (0, 1)
$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.10}$$
- Softmax: commonly used as the activation function in the output layer because it outputs a normalized probability distribution. Each element of the output layer represent,s for example, the probability that the input belongs to a particular class.
$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.11}$$

2.4.1.3 Loss Functions

The loss functions commonly used for training MLPs are:

- Mean Squared Error (MSE): The MSE loss is the default choice for regression problems, where the goal is to minimize the average of the squares of the errors between the predicted and actual values. It is suitable for regression tasks where the output is a continuous value

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y_i)^2 \tag{2.12}$$

  where y_i are the reference values and $\hat{y}$ the predicted values.

- Cross-Entropy Loss: is the default loss function for class classification problems. It measures the performance of a classification model whose output is a probability value between 0 and 1.

Chapter 3

Data Preparation

Data-driven models demand the generation of considerable amounts of data which can be an extensive and expensive work. The data was generated with software written in C++ and had to be translated to Python to be used by TensorFlow. TensorFlow was the software used to implement the MLP algorithm. In this chapter, the process of data generation and pre-processing will be presented.

3.1 Task formulation

The goal is to build a model that can learn from the velocity field how to predict the pressure gradient, and subsequentially the pressure, in the flow past an object for laminar channel flow. This is a regression task since we aim to map the known inputs to the known output.

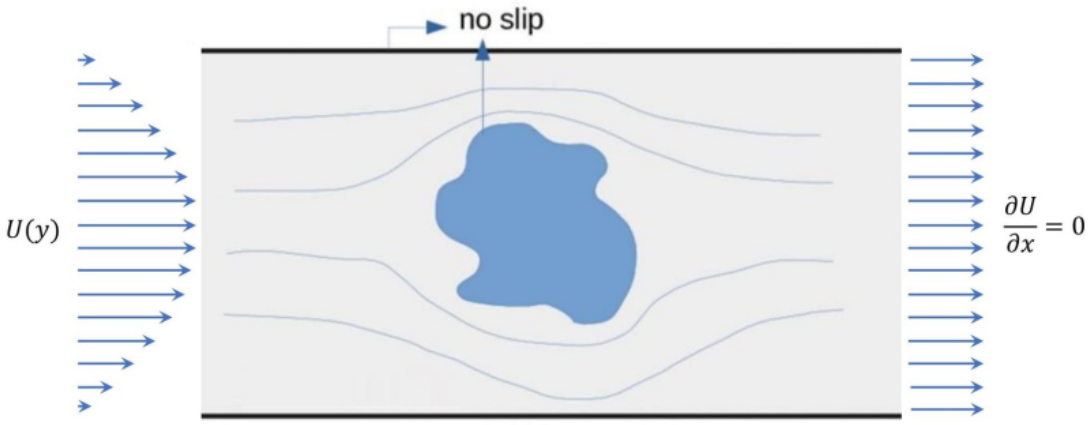

Figure 3.1: Problem representation. From [4]

3.2 Data Generation

The data was generated by performing simulations with the OpenFOAM software, an open-source CFD software made public in 2004, in its version v1812. A large number of distinct simulations were performed aiming to achieve the most diverse dataset possible. As these simulations can be time demanding to set up, a python script was used to generate meshes with different objects and different object sizes, altering its blockage ratio. The solver used was a modified version of pisoFOAM - a well known and established solver - that also writes the pressure gradients that we wanted to use for training. The simulations domain was defined as $x \in [0, 15]$ and $y \in [-1, 1]$, with y behind the dimension perpendicular to the flow. The different types of objects used to semi block the channel were a cylinder, a triangle pointing towards the flow and a inclined plate as shown in the figures below. The simulations were performed for 4 different Reynolds numbers: 1,10,100 and 500, and blockage ratios ranging in between 0.125 and 0.75 of the channel height. The boundary condition applied at the inlet, U(y), is a parabolic velocity profile coming from the analytical solution to fully developed laminar flow between parallel plates:

$$U(y) = \frac{3}{2}\overline{u}\left(1 - \left(\frac{y}{h}\right)^2\right) \tag{3.1}$$

where y is the distance from the centerline, h half of the distance between plates and $\overline{u}$ the mean velocity. In total 76 simulations were used, 64 to train the model and 12 to test its generalization capabilities.

Some results obtained from the pisoFOAM can be seen in the next figures:

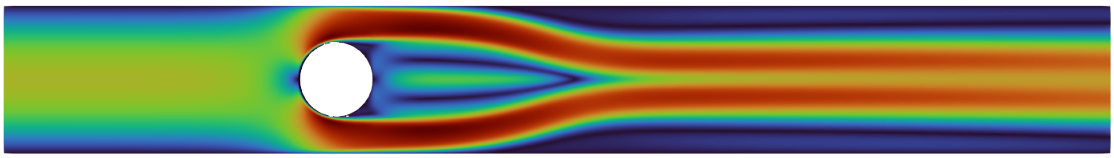

Figure 3.2: Velocity field magnitude of a flow past a cylinder with a 0.5 blockage ratio and Re=500.

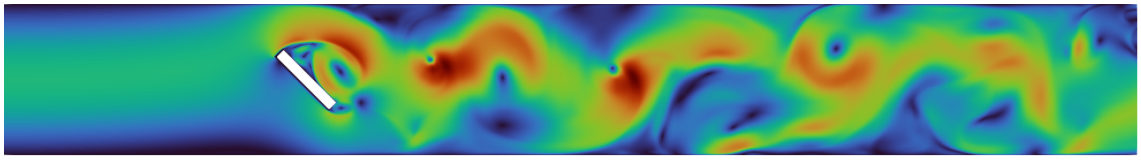

Figure 3.3: Velocity field magnitude of a flow past a plate with a 0.5 blockage ratio and Re=50.

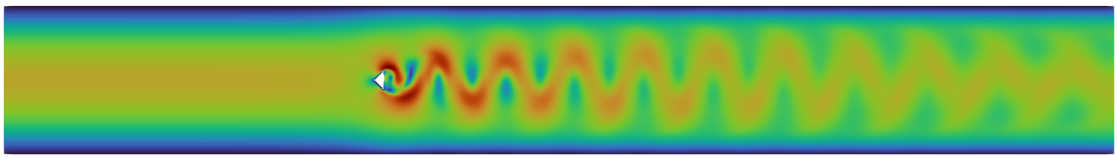

Figure 3.4: Velocity field magnitude of a flow past a triangle with a 0.125 blockage ratio and Re=100.

3.3 Data Pipeline

The initial phase of the project involved the successful extraction of data from the simulations. The ultimate objective was to transfer the simulation data to Python for utilization with the TensorFlow API. To achieve this, the simulation results needed to be exported into a Python array. This data pipeline encompassed several key steps:

- Acquiring Relevant Data: Retrieved computed quantities at grid cell centers and converted them to VTK files from simulations.
- Data Reading and Pre-processing: Read and pre-processed data from VTK files using the pyVista interface.
- Data Storage in Hierarchical Binary Format: Converted data into arrays, padded for consistent size, and stored in HDF5 format.

Upon completing these steps, we obtained an HDF5 file containing the full mesh information. The relevant training data arrays were then aggregated into a higher-dimensional array for use in training the MLP.

3.4 Data Pre-processing

The first step of pre-processing consists in nondimensionalizing the flow field quantities in relation to the maximum velocity found in the flow domain. The maximum velocity is calculated by

$$u_{max} = max\sqrt{u_x^2 + u_y^2} \tag{3.2}$$

the nondimensional velocity vector, $\mathbf{u}^*$ by

$$\mathbf{u}^* = \frac{\mathbf{u}}{u_{max}} \tag{3.3}$$

and the nondimensional pressure gradient fields, $\frac{\partial p}{\partial x}^*$ and $\frac{\partial p}{\partial y}^*$ by

$$\frac{\partial p}{\partial x}^* = \frac{\frac{\partial p}{\partial x}(x_{max} - x_{min})/\rho}{u_{max}^2} \tag{3.4}$$

$$\frac{\partial p}{\partial y}^* = \frac{\frac{\partial p}{\partial y}(y_{max} - y_{min})/\rho}{u_{max}^2} \tag{3.5}$$

respectively. It is widely recognized that neural networks perform better with normalized data, specifically when the data falls within the ranges of $[-1,1]$ or $[0,1]$. To apply this normalization, a field quantity ϕ, is normalized according to

$$\phi^* = \frac{\phi - min\phi}{max\phi - min\phi} \tag{3.6}$$

which bounds ϕ^* to [0,1].

3.5 Data Selection

To create a generic model applicable to any geometry, we followed Paulo's approach [4] by dividing the domain into fixed-shape square blocks and training with these blocks. This method allows us to represent any geometry within a rectangle by assembling the blocks and identifying which cells belong to the flow domain, as illustrated in Figure 3.5, where the darker grey region is outside the flow domain. The chosen study case was a confined flow past an obstacle, schematically represented in Figure 3.1, with the relevant boundary conditions applied.

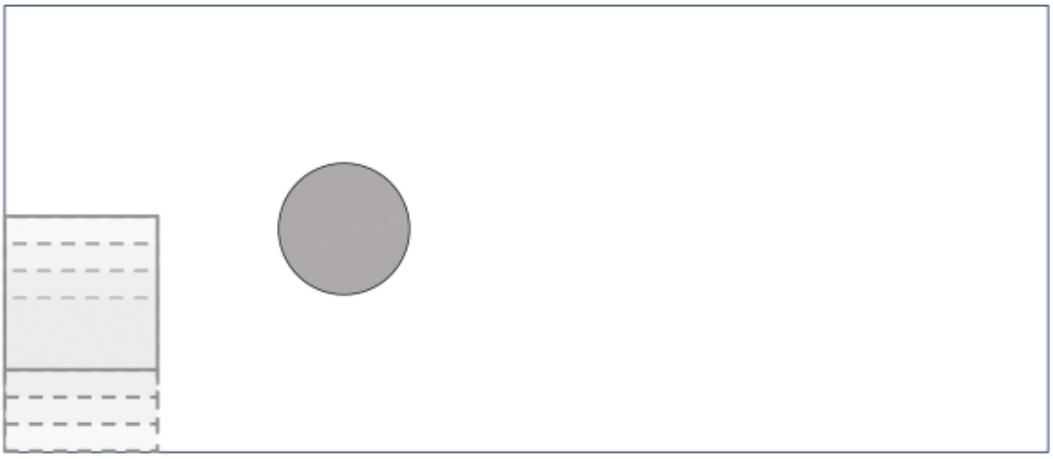

Figure 3.5: Illustration of the sampling method from the original domain.

Chapter 4

Pressure-gradient DL model

4.1 Architecture

The surrogate model's goal is to be able to map the pressure gradient with the x and y components of the velocity as input fields and an extra parameter to represent the geometry. The flow field quantities will go through Principal Component Analysis before being fed to the neural network. The chosen architecture for the neural network was a Multilayer Perceptron. The available data was divided into training and test sets, with 90% allocated to training and the remaining 10% to testing. Previous experiments indicated that using 10% of the data for testing was sufficient to evaluate the training performance.

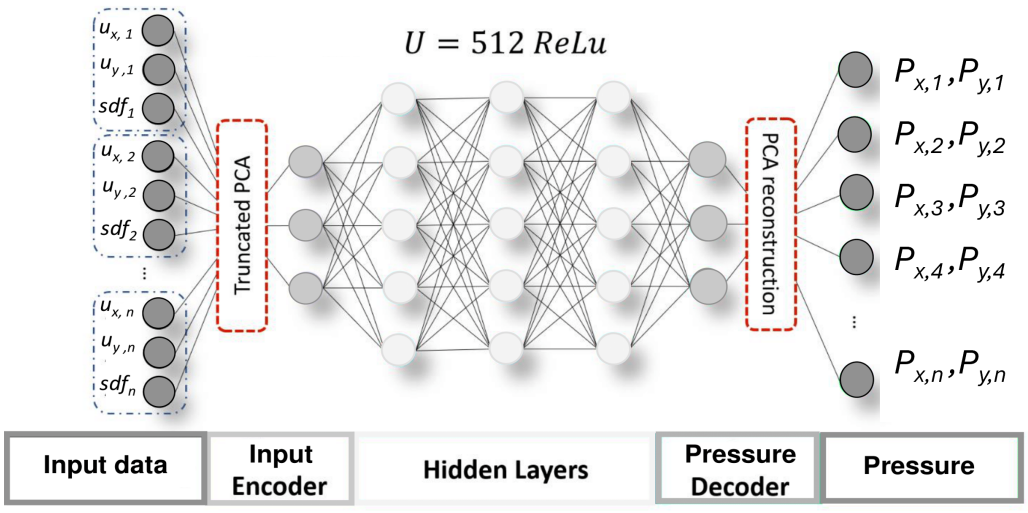

Figure 4.1: Deep learning model architecture schematized.

4.2 Loss Metric and Early Stopping

The cost function used is the mean squared error (MSE) defined as

$$MSE = \frac{1}{n_{cells}} \sum_{i=1}^{n_{cells}} \left(\hat{\theta} - \theta\right)^2 \tag{4.1}$$

where n_{cells} is the number of cells where an estimation is computed, θ the reference values, and $\hat{\theta}$ the predicted values. Early stopping is a technique used to halt the training of a neural network to prevent overfitting. It is based on monitoring a performance metric on a validation set and stopping the training process when the performance metric stops improving. The early stopping criterion was based on the test set loss, halting training if no 5% improvement was observed in the test set loss over 25 epochs.

4.3 Model predictions

The evaluation of the model's performance was made using the BIAS,

$$BIAS = \frac{1}{n_{cells}} \sum_{i=1}^{n_{cells}} \left(\hat{\theta} - \theta\right) \tag{4.2}$$

, the standard-deviation error (STDE),

$$STDE = \sqrt{\frac{1}{n_{cells-1}} \sum_{i=1}^{n_{cells}} \left(\hat{\theta} - \theta\right)^2}, \tag{4.3}$$

and the root mean squared error (RMSE),

$$RMSE = \frac{1}{n_{cells}} \sum_{i=1}^{n_{cells}} \left(\hat{\theta} - \theta\right)^2 \tag{4.4}$$

and afterward normalizing each of those with the range of values from the reference results with $norm = max(\phi) - min(\phi)$. $BIAS_{norm}$, $STDE_{norm}$ and $RMSE_{norm}$ come from dividing BIAS, STDE and RMSE by norm, respectively.

4.4 Hyper-parameters

The relevant hyper-parameters that were tuned to create the best conditions for training are:

- Loss metric, to lead the model in better predicting the Principal Components (PC).
- Number of PCs in the input and output layers, defining how much of the data is filtered.
- Batch size, which can significantly impact model accuracy. Larger batch sizes may compromise the model's generalization ability.

- Learning rate, $\alpha = 1 \times 10^{-4}$, and moving average, $\beta = 0.99$. Parameters required by the optimization algorithm used.
- Model depth, defined by the number of hidden layers in the network.
- Dropout was set to zero.

4.5 Results

Three models were successfully trained with 3 different datasets. The datasets with the cylinder and triangle as obstacles consisted of 20 simulations with different blockage ratios and Reynolds numbers and the dataset with the plate as obstacle consisted of 24 simulations, also with different blockage ratios and Reynolds numbers.

4.5.1 Predictions on the training dataset

The models were tested in their respective training datasets and the error metrics were calculated for each gradient independently as well as for both gradients combined. The combined errors can be seen in the tables below:

Table 4.1: Errors for $\mathbf{M}_{\circ}$ and $\mathbf{M}_{\triangleleft}$ on simulations with a 0.75 blockage ratio

Blockage Ratio = 0.75					
Model	Error Metric %	Re = 1	Re = 10	Re = 100	Re = 500
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	-1.38	6.60	-9.32	2.29
	$STDE_{norm}$	7.49	105.26	39.30	26.13
	$RMSE_{norm}$	7.36	105.05	38.18	26.03
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	-0.04	-0.45	-0.15	0.51
	$STDE_{norm}$	0.28	1.55	1.32	5.29
	$RMSE_{norm}$	0.28	1.49	1.31	5.27

Table 4.2: Errors for $\mathbf{M}_{/}$ on simulations with a 0.6 blockage ratio

Blockage Ratio = 0.6					
Model	Error Metric %	Re = 1	Re = 10	Re = 100	Re = 500
$\mathbf{M}_{/}$	$BIAS_{norm}$	-0.05	-0.21	-0.15	0.16
	$STDE_{norm}$	0.36	0.84	1.94	2.81
	$RMSE_{norm}$	0.36	0.81	1.93	2.80

Table 4.3: Errors for $\mathbf{M}_{\circ}$, $\mathbf{M}_{/}$ and $\mathbf{M}_{\triangleleft}$ on simulations with a 0.5 blockage ratio

Blockage Ratio = 0.5					
Model	Error Metric %	Re = 1	Re = 10	Re = 100	Re = 500
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	0.12	1.26	1.97	8.04
	$STDE_{norm}$	1.52	10.04	10.80	16.56
	$RMSE_{norm}$	1.52	9.96	10.62	14.74
$\mathbf{M}_{/}$	$BIAS_{norm}$	-0.39	-2.07	0.51	0.31
	$STDE_{norm}$	1.33	4.33	2.92	4.03
	$RMSE_{norm}$	1.28	3.80	2.88	4.02
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	0.15	0.53	-0.32	-0.03
	$STDE_{norm}$	0.36	1.51	1.55	2.92
	$RMSE_{norm}$	0.33	1.41	1.52	2.91

Table 4.4: Errors for $\mathbf{M}_{\circ}$, $\mathbf{M}_{/}$ and $\mathbf{M}_{\triangleleft}$ on simulations with a 0.25 blockage ratio

Blockage Ratio = 0.25					
Model	Error Metric %	Re = 1	Re = 10	Re = 100	Re = 500
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	0.60	0.09	0.34	0.74
	$STDE_{norm}$	1.18	3.20	2.82	3.63
	$RMSE_{norm}$	1.02	3.20	2.80	3.55
$\mathbf{M}_{/}$	$BIAS_{norm}$	0.06	-0.35	0.06	0.34
	$STDE_{norm}$	0.37	1.00	1.39	2.43
	$RMSE_{norm}$	0.36	0.94	1.39	2.41
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	1.66	0.21	-2.51	-0.70
	$STDE_{norm}$	3.85	1.62	5.48	4.59
	$RMSE_{norm}$	3.47	1.61	4.87	4.54

Table 4.5: Errors for $\mathbf{M}_{\circ}$ and $\mathbf{M}_{\triangleleft}$ on simulations with a 0.2 blockage ratio

Blockage Ratio = 0.2					
Model	Error Metric %	Re = 1	Re = 10	Re = 100	Re = 500
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	0.22	-0.59	-0.58	-0.78
	$STDE_{norm}$	0.91	1.68	2.43	3.10
	$RMSE_{norm}$	0.88	1.57	2.35	3.00
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	0.14	-0.27	-1.38	-0.08
	$STDE_{norm}$	0.47	1.13	4.51	3.57
	$RMSE_{norm}$	0.45	1.10	4.29	3.57

Table 4.6: Errors for $\mathbf{M}_{\circ}$, $\mathbf{M}_{/}$ and $\mathbf{M}_{\triangleleft}$ on simulations with a 0.125 blockage ratio

Blockage Ratio = 0.125					
Model	Error Metric %	Re = 1	Re = 10	Re = 100	Re = 500
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	0.18	-0.25	-0.83	-0.08
	$STDE_{norm}$	0.71	1.40	2.31	1.88
	$RMSE_{norm}$	0.69	1.38	2.15	1.88
$\mathbf{M}_{/}$	$BIAS_{norm}$	0.24	0.26	-0.24	0.31
	$STDE_{norm}$	0.55	0.94	1.38	1.65
	$RMSE_{norm}$	0.49	0.90	1.36	1.62
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	0.12	-1.45	-0.74	0.31
	$STDE_{norm}$	0.52	2.93	3.31	2.77
	$RMSE_{norm}$	0.50	2.54	3.22	2.75

Table 4.7: Errors for $\mathbf{M}_{/}$ on simulations with a thicker plate and 0.5 and 0.6 blockage ratios respectively

Blockage Ratio = 0.6 and 0.5 on a thicker plate					
Model	Error Metric %	Re = 1	Re = 10	Re = 100	Re = 500
$\mathbf{M}_{/}$	$BIAS_{norm}$	-0.06	-0.47	-0.52	-0.07
	$STDE_{norm}$	0.34	1.32	2.74	4.17
	$RMSE_{norm}$	0.33	1.23	2.69	4.17
$\mathbf{M}_{/}$	$BIAS_{norm}$	0.05	0.13	0.49	0.43
	$STDE_{norm}$	0.33	1.07	1.78	3.44
	$RMSE_{norm}$	0.33	1.06	1.71	3.41

The plotted results show considerable similarity between the simulations and the predicted values from the models:

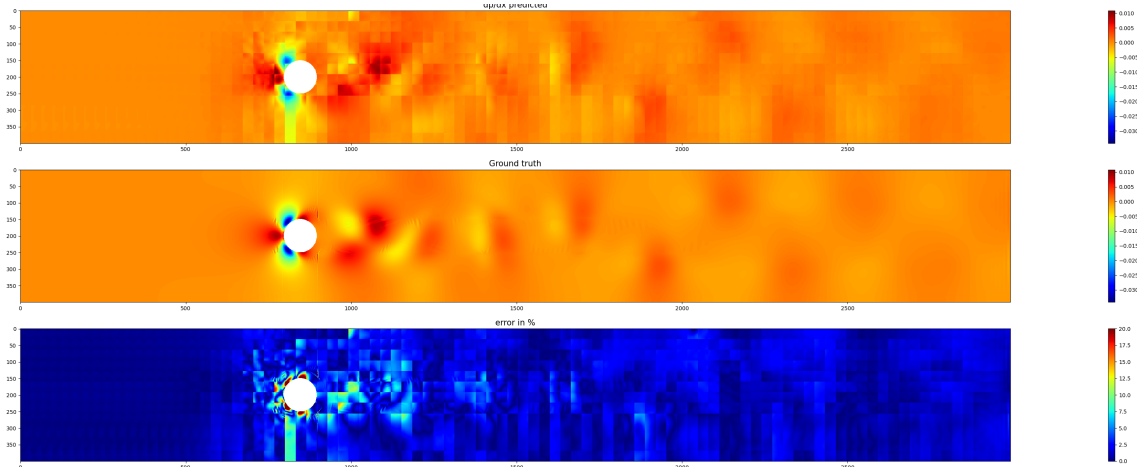

Figure 4.2: $\mathbf{M}_\circ$ prediction of $\frac{\partial P}{\partial x}$ for Re=100 at t=100s.

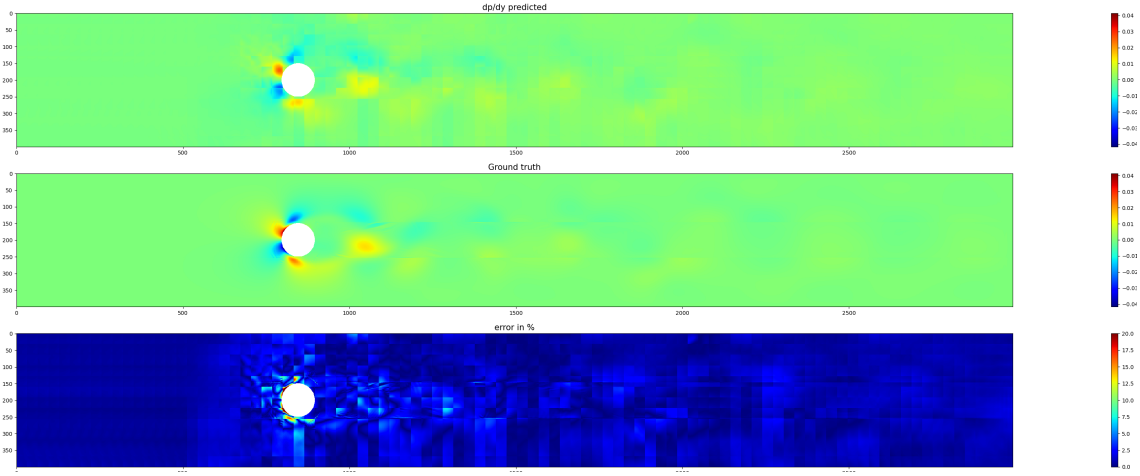

Figure 4.3: $\mathbf{M}_\circ$ prediction of $\frac{\partial P}{\partial y}$ for Re=100 at t=100s.

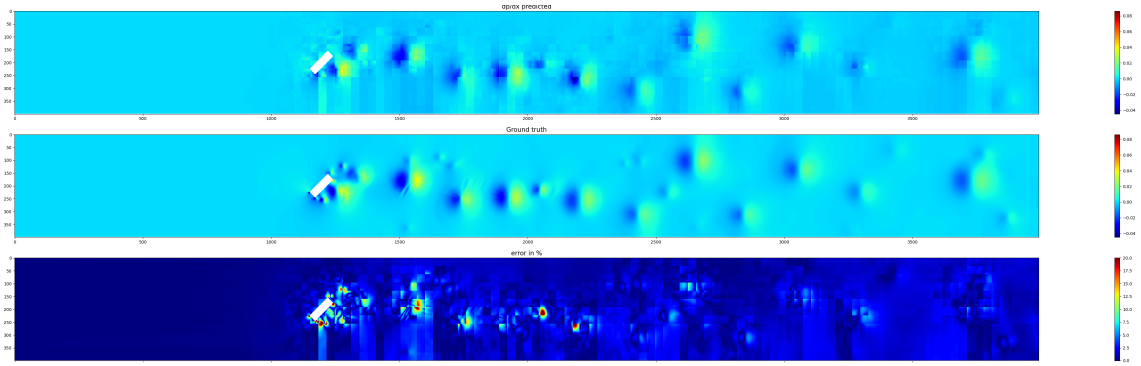

Figure 4.4: $\mathbf{M}_{/}$ prediction of $\frac{\partial P}{\partial x}$ for Re=500 at t=100s.

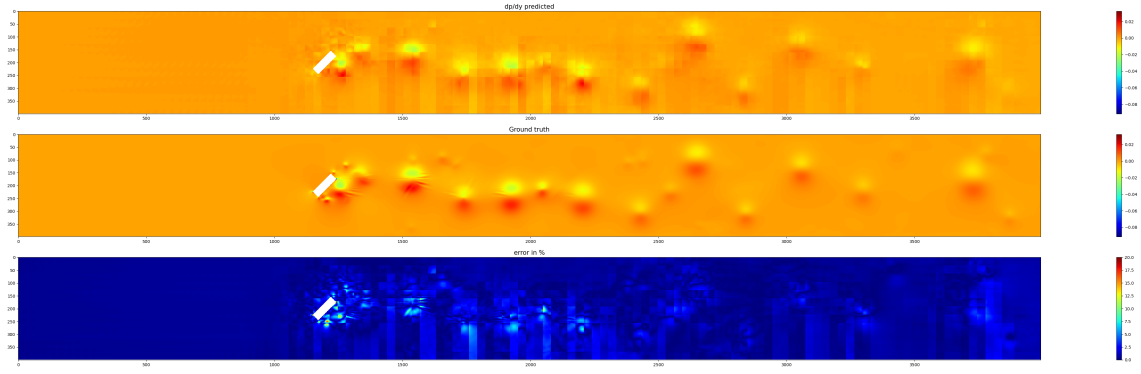

Figure 4.5: $\mathbf{M}_{/}$ prediction of $\frac{\partial P}{\partial y}$ for Re=500 at t=100s.

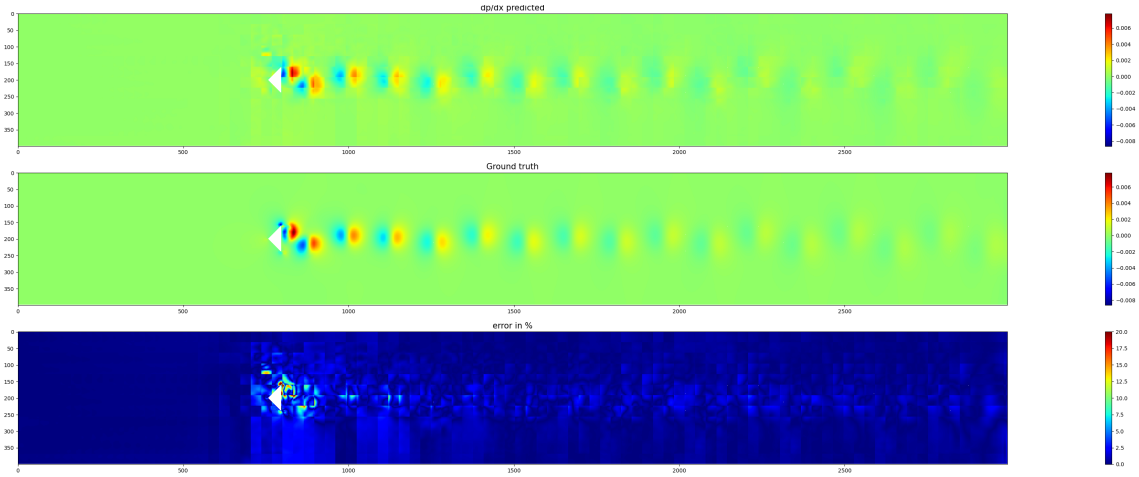

Figure 4.6: $\mathbf{M}_{\triangleleft}$ prediction of $\frac{\partial P}{\partial x}$ for Re=500 at t=100s.

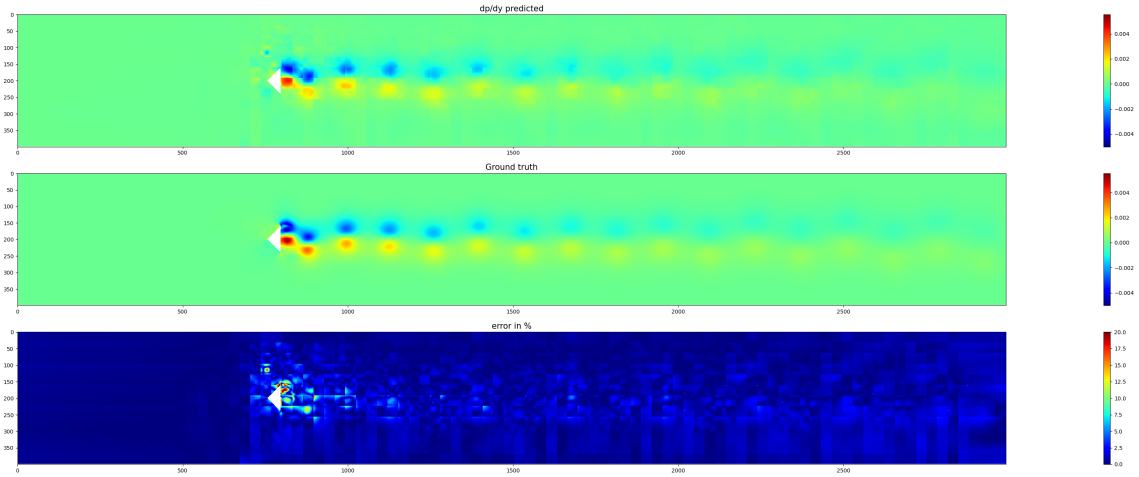

Figure 4.7: $\mathbf{M}_{\triangleleft}$ prediction of $\frac{\partial P}{\partial y}$ for Re=500 at t=100s.

As it can be observed, the models predict better the gradients in the y direction than in the x direction. This might be explained by the pressure changes being harder to predict in the main direction of the flow.

4.5.2 Extrapolation tests

The models were also tested in conditions previously unseen by them in the training phase. These simulations were done in a 0.5 blockage ratio channel for Re = 0.1, 50, 250 and 1000 and with the three different geometries as obstacles. The yielded results can be seen in the following table:

Table 4.8: Errors for $\mathbf{M}_{\circ}$, $\mathbf{M}_{/}$ and $\mathbf{M}_{\triangleleft}$ on extrapolation tests

Blockage Ratio = 0.5					
Model	Error Metric %	Re = 0.1	Re = 50	Re = 250	Re = 1000
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	0.79	-2.83	9.51	-4.42
	$STDE_{norm}$	3.14	17.19	19.74	33.91
	$RMSE_{norm}$	3.04	16.95	17.30	33.62
$\mathbf{M}_{/}$	$BIAS_{norm}$	0.07	0.49	0.49	0.46
	$STDE_{norm}$	0.45	3.22	3.22	3.35
	$RMSE_{norm}$	0.44	3.18	3.18	3.02
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	0.06	-0.77	-1.15	2.12
	$STDE_{norm}$	0.32	2.84	4.89	7.22
	$RMSE_{norm}$	0.31	2.73	4.75	6.90

One of the most notable results being a $RMSE_{norm}$ of 3.02 in $\mathbf{M}_{/}$ predicting for Re=1000. The prediction for t=100s can be seen below.

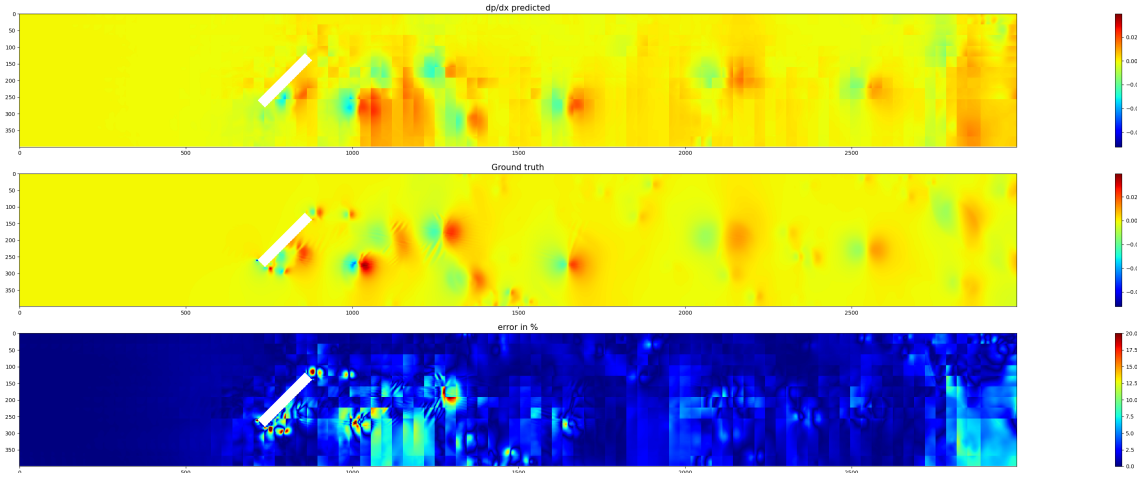

Figure 4.8: $\mathbf{M}_{/}$ prediction of $\frac{\partial P}{\partial x}$ for Re=1000 at t=100s.

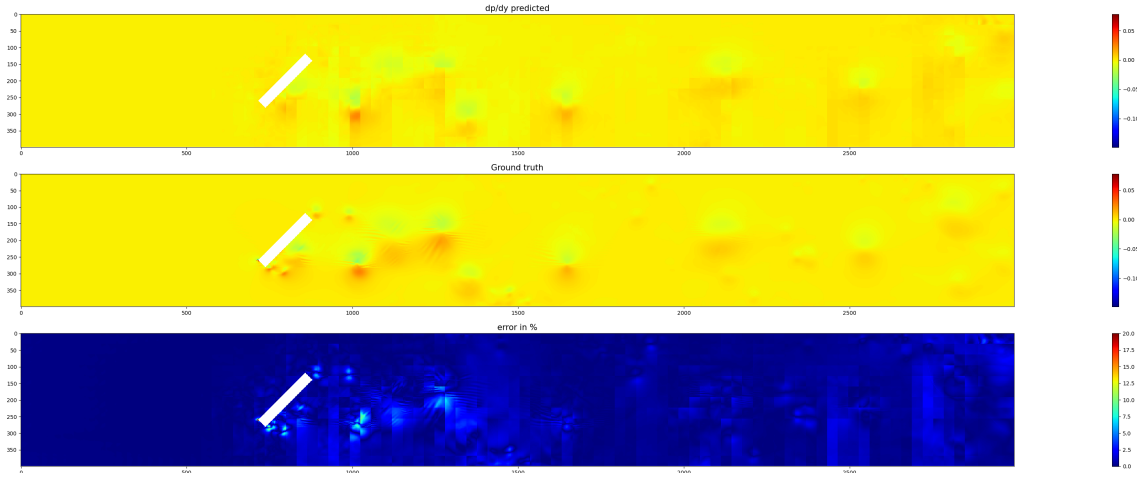

Figure 4.9: $\mathbf{M}_{/}$ prediction of $\frac{\partial P}{\partial y}$ for Re=1000 at t=100s.

Chapter 5

Conclusions and Future Works

The study aimed to develop a Multilayer Perceptron (MLP) to aid CFD software's pressure solvers. Principal Component Analysis (PCA) played a crucial role by replacing encoder and decoder layers with PCA transformations, simplifying the neural network's task. Surrogate models were successfully constructed to estimate pressure gradients for confined flows across various geometries.

- Given proper data generation and pre-processing, data-driven methods have demonstrated their viability for most applications, acting as effective interpolators and yielding reasonable results that closely resemble the training data.
- The combination of PCA and a simple MLP proved to be efficient and was trained with minimal computational resources, allowing computations to be performed without relying on GPUs.
- The proposed $\mathbf{M}_{\circ}$ and $\mathbf{M}_{\triangleleft}$ models revealed to be good interpolators, managing to predict with very low errors the pressure gradients in every test case from in training flow conditions.
- $\mathbf{M}_{\circ}$ predicted successfully for flows with smaller blockage ratios, in which the pressure drop around the cylinder wasn't too big. For blockage ratios around and above 0.5 the model wasn't able to present good results.
- In extrapolation for non-training data $\mathbf{M}_{\triangleleft}$ managed to predict the pressure gradients with very low errors in the four Reynolds numbers tested, the highest $RMSE_{norm}$ being 3.18, with emphasis for the prediction for Re=1000 in which the yielded $RMSE_{norm}$ was of 3.02.
- Greater diversity in the training dataset enhanced the model's overall performance.
- The training approach proved successful. For certain obstacles, predictions were visually indistinguishable from CFD results, occasionally achieving normalized root mean square errors ($RMSE_{norm}$) below 1%.

5.1 Future Works

The $\mathbf{M}_\circ$ model could probably be improved by tuning the hyper-parameters.
The obvious next step would be to build a model with all the simulations combined. Several tries were made but it wasn't possible to fix every setback and obtain enough computational memory to do so.
Another interesting step would be to implement the surrogate model in a pressure solver to evaluate its overall efficiency in replacing the traditional numerical methods that consume high amounts of computational time.

Bibliography

[1] Fouzia Altaf et al. "Going Deep in Medical Image Analysis: Concepts, Methods, Challenges, and Future Directions". In: *IEEE Access* 7 (2019), pp. 99540–99572. DOI: 10.1109/ACCESS.2019.2929365.

[2] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. "Machine Learning for Fluid Mechanics". In: *Annual Review of Fluid Mechanics* 52.1 (2020), pp. 477–508. DOI: 10.1146/annurev-fluid-010719-060214. eprint: https://doi.org/10.1146/annurev-fluid-010719-060214. URL: https://doi.org/10.1146/annurev-fluid-010719-060214.

[3] Giovanni Calzolari and Wei Liu. "Deep learning to replace, improve, or aid CFD analysis in built environment applications: A review". In: *Building and Environment* 206 (2021), p. 108315. ISSN: 0360-1323. DOI: https://doi.org/10.1016/j.buildenv.2021.108315. URL: https://www.sciencedirect.com/science/article/pii/S0360132321007137.

[4] Paulo Araújo da Cunha Sousa. *Solving Poisson's Equation through Deep Learning for CFD applications*. 2022.

[5] Yli-Harja O Emmert-Streib F and Dehmer M. "Artificial Intelligence: A Clarification of Misconceptions, Myths and Desired Status." In: *Front. Artif. Intell. 3:524339* (2020).

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

[7] IBM. *What is artificial intelligence (AI)?* URL: https://www.ibm.com/topics/artificial-intelligence.

[8] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997. ISBN: 0070428077.

[9] Ricardo Vinuesa and Steven L. Brunton. "Enhancing computational fluid dynamics with machine learning". In: *Nature Computational Science* 2.6 (June 2022), pp. 358–366. ISSN: 2662-8457. DOI: 10.1038/s43588-022-00264-7. URL: http://dx.doi.org/10.1038/s43588-022-00264-7.