UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

MEC4110W - Project 41

Daniel Ferrini - FRRDAN014

13/11/2020

Plagiarism Declaration

I, Daniel Ferrini, hereby declare that:

i. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.

ii. I have used the IEEE convention for citation and referencing. Each significant contribution to, and quotation in, this report / project from the work(s) of other people has been attributed and has been cited and referenced.

iii. This report/project is my own work.

iv. I have not allowed and will not allow anyone to copy my work with the intention of passing it off as his or her own work.

I am henceforth presenting this report under examination for the degree of:

B.Sc Eng (Mechanical & Mechatronics).

D.S. Ferrini
13/11/2020

This page intentionally left blank

Abstract

The practice of applying machine learning as a parallel aid to improve the applicability of CFD systems is a relatively understudied field. Due to the associated computational efficiencies, these machine learning programs are ideal for in-situ implementation in engineering systems that require accurate near real-time fluid flow data. By providing these trained learning algorithms with a large CFD dataset, the ML models can potentially be used to reconstruct the high dimensionality, non-linear data of seemingly non-algorithmic fluid systems (particularly in the case of fluid mixing). This report details the development, testing, and evaluation of three machine learning regression models (Neural Network, Random Forest, and Autoencoder) on their ability to accurately reconstruct the CFD data trends of a two dimensional mixing T-piece. From literary investigation the correct requirements for both the CFD simulation configuration, and algorithm development were determined. By generating 1000 CFD data points, each of the three machine learning models could be trained and validated appropriately. From this training, it was found that a 128 neurone per layer Neural Network, 150 tree Random Forest, and a 32 dimensional intrinsic space autoencoder were the optimal model architectures for generalising on the provided CFD data. These selected models were then evaluated on their prediction performance, and computational efficiency's. From these model evaluations, it was found that the Random Forest produced the best prediction results for turbulent fluid flow. Although the Neural Network was more capable of generalising over a wider CFD data range, as well as being the most ideal model for implementation in existing fluid based systems.

Acknowledgements

The formulation and compilation of this research project would not have been possible without the help and support of the remarkable people around me:

Foremost;

I would like to express my gratitude to my research supervisor *Dr Ryno Laubscher*, for his support in the development of this research project. Dr Laubscher's involvement and support has guided me throughout the course of this tempestuous year. His wisdom, and expertise has motivated me to complete this report to the best of my abilities. Without your assistance this project would not have been as enjoyable. Thank you for introducing me to the world of machine learning, this experience has sparked a new and lasting interest.

Secondly;

I would like to share my thanks to *Simon Krone* for taking his own time to develop and share the cover page templates used in both this report and the interim report. Thank you for your generosity as the compiling of these cover pages saved me priceless project time.

Thirdly;

I would like to acknowledge *Brandon Reabow* and *Christopher Mailer* for their assistance in the provision of machine learning tutorial resources, the sharing and brainstorming of ideas, and the identification of errors relating to the computational aspect of the project. Thank you for these contributions, as without them, the completion of this project would not have been possible.

Lastly;

I would like to thank *Dean Brand* and *Iain Colley* for their general support, motivation, and humour which constantly drives me to better myself throughout the course of this project and in every other challenge I have undertaken.

Contents

Abstract **iii**

Acknowledgements **iv**

Glossary **x**

Nomenclature **xii**

List of Abbreviations **xiv**

Introduction **1**

Problem Statement **4**
Scope 4
Limitations 6

Literature Review **7**
Conservation Equations and CFD Setup 7
Machine Learning 11
Machine Learning in Fluid Mechanics 14

Materials and Methods **16**
CFD and Data Preparation 16
Neural Network 21
Random Forest 26
Autoencoder 29

Results and Discussion **34**
Prediction Error and Comparison 34
Algorithm Runtime and Memory 39
Lower-Bound Error and Comparison 42
Upper-Bound Error and Comparison 47
Discussion 51

Conclusion **54**

References **56**

Appendix I - Project Structure **59**

Appendix II - Experimentation Theory **61**
Mesh Modelling . . . 61
Data Processing . . . 62
Regularisation . . . 64

Appendix III - Model Selection **65**
Neural Network Selection Process . . . 65
Random Forrest Selection Process . . . 68
Autoencoder Selection Process . . . 71

Appendix IV - Detailed Model Analysis **79**
Turbulence Prediction Results . . . 79
Restricted Prediction Contour Resolution . . . 81
Detailed Prediction Error Analysis . . . 82
Lower-bound Prediction Error Analysis . . . 84
Upper-bound Prediction Error Analysis . . . 86

Appendix V - Implementation **89**

Appendix VI - Interim Report **91**
Nomenclature . . . iii
List of Abbreviations . . . iii
Introduction . . . 1
Problem Statement . . . 2
Scope . . . 2
Limitations . . . 3
Literature Review . . . 4
Conservation Equations and CFD Setup . . . 4
Machine Learning . . . 7
Project Planning . . . 10
Appendix . . . 13
EBE Ethics Form . . . 13
Risk Identification Form . . . 14
Impact of Technology Statement . . . 15
Work Breakdown Structure . . . 16
Gantt Chart . . . 17

List of Figures

1 Scale diagram of dimensioned T-piece to be analysed 4
2 Standard wall function meshing for near wall treatment 10
3 Machine learning categories and subclasses 11
4 Machine learning training profiles for varying model structures 13
5 Initial CFD simulation contour solution 18
6 Functional operation of an artificial neurone 21
7 Standard Model of a Neural Network 22
8 Gradient descent path of three dimensional surface 23
9 Annotated structure of a Decision Tree Regressor 26
10 Flow diagram of Random Forest 28
11 Standard Model of an Autoencoder 30
12 Flow diagram for Autoencoder training process 32
13 Mean temperature prediction and error contour for Neural Network . . . 35
14 Mean velocity prediction and error contour for Neural Network 36
15 Mean temperature prediction and error contour for Random Forest . . . 36
16 Mean velocity prediction and error contour for Random Forest 37
17 Mean temperature prediction and error contour for Autoencoder 38
18 Mean velocity prediction and error contour for Autoencoder 38
19 Interpolated algorithm runtime projections 40
20 Random access memory usage for varying data sample sizes 41
21 Mean temperature and velocity contours for lower-bound data range . . . 42
22 Lower-bound temperature prediction and error contour for Neural Network 43
23 Lower-bound velocity prediction and error contour for Neural Network . 44
24 Lower-bound temperature prediction and error contour for Random Forest 44
25 Lower-bound velocity prediction and error contour for Random Forest . . 45
26 Lower-bound temperature prediction and error contour for Autoencoder . 46
27 Lower-bound velocity prediction and error contour for Autoencoder . . . 46
28 Mean temperature and velocity contours for upper-bound data range . . 47
29 Upper-bound temperature prediction and error contour for Neural Network 48
30 Upper-bound velocity prediction and error contour for Neural Network . 49
31 Upper-bound temperature prediction and error contour for Random Forest 49
32 Upper-bound velocity prediction and error contour for Random Forest . . 50
33 Upper-bound temperature prediction and error contour for Autoencoder 50
34 Upper-bound velocity prediction and error contour for Autoencoder . . . 51
35 High level project structural layout and summary 59
36 Process flow diagram for algorithm development and evaluation 60
37 Grid mesh components and layout within the T-piece section 62
38 Neural Network training and validation convergence graphs 67
39 Validation curve for simple Random Forest structures 69

40 Validation curve for Random Forests with added random state 70
41 Convergence graphs for variable Autoencoder structures 71
42 16 neurone Autoencoder convergence for Dropout, Batch, and combination 73
43 32 neurone Autoencoder convergence for Dropout, Batch, and combination 73
44 64 neurone Autoencoder convergence for Dropout, Batch, and combination 74
45 16 neurone encoding space Neural Network training and validation graphs 75
46 32 neurone encoding space Neural Network training and validation graphs 76
47 Autoencoder CFD temperature prediction contour comparison 77
48 Autoencoder CFD velocity prediction contour comparison 77
49 T-piece turbulence kinetic energy (K) and dissipation rate (ε) distribution 79
50 Turbulence Reynolds cell number contour for mixing T-piece model . . . 80
51 Restricted mean Autoencoder temperature and error prediction contour . 81
52 Restricted mean Autoencoder velocity and error prediction contour . . . 82
53 Logarithmic cell prediction error frequency distribution for Neural Network 83
54 Logarithmic cell prediction error frequency distribution for Random Forest 83
55 Logarithmic cell prediction error frequency distribution for Autoencoder . 84
56 Lower-bound cell prediction error frequency distribution for Neural Network 85
57 Lower-bound cell prediction error frequency distribution for Random Forest 85
58 Lower-bound cell prediction error frequency distribution for Autoencoder 86
59 Upper-bound cell prediction error frequency distribution for Neural Network 87
60 Upper-bound cell prediction error frequency distribution for Random Forest 87
61 Upper-bound cell prediction error frequency distribution for Autoencoder 88
62 Diagram of dimensioned T-piece to be analysed 2
63 Diagram of machine learning categories 7
64 Diagram showing machine learning training profiles 9
65 Simplified flow chart showing general process for machine learning algorithms 11
66 Work Breakdown Structure displaying the scope for machine learning project 16
67 Gantt chart showing machine learning research project lifespan 17

List of Tables

1 Initial design point boundary parameters 17
2 Inlet boundary ranges used in mixing T-piece 18
3 Mean absolute errors for each algorithm prediction 34
4 Runtime measurements for varying instance sample sizes 39
5 Static and resident set size memory usage for varying instance sample sizes 41
6 Lower-bound inlet boundary ranges used in mixing T-piece 42
7 Mean absolute prediction errors for lower-bound CFD data samples . . . 43
8 Upper-bound inlet boundary ranges used in mixing T-piece 47
9 Mean absolute prediction errors for upper-bound CFD data samples . . . 48
10 Neurone quantities per layer for varying Neural Network structures . . . 66
11 Neurone quantities per layer for varying Autoencoder structures 71
12 Neural Network structures for differing encoding spaces 75
13 Python based microcontrollers and corresponding performance metrics . 89
14 Estimated maximum continuous batch sample size capable of being predicted 90

Glossary

Autoencoder	-	A specialised variation of Neural Networks, commonly used for non-linear dimensionality reduction.
Backpropagation	-	The process of modifying/updating the relevant weights of a Neural Network.
Convergence	-	The tendency for a time dependant function to stabilise on a finite solution.
Computational Fluid Dynamics	-	The study of modelling fluid flow with the aid of computational/digital systems.
Decision Tree	-	A type of machine learning algorithm with a branched architecture that resembles the structure of a tree.
Epoch	-	A distinct period/event in time.
Hyper-Parameter	-	Parameters that can be modified to change the learning outcome of a machine learning algorithm.
Latin-Hypercube	-	A multidimensional sampling method that is used to generate almost random data distributions.
Machine Learning	-	The study of computer based prediction algorithms that are capable of improving their prediction performance through experience.
Matplotlib	-	A data graphing library for Python.
Meshing	-	The discretisation of a continuous surface into finite elements/volumes for the application in computational simulations.
Neural Network	-	A class of machine learning algorithms with internal architectures that resemble that of the human brain.
Non-Volatile Memory	-	A form of memory, primarily used for data storage and transfer, that can operate after being power cycled.

NumPy	-	A Python based multi-dimensional linear algebra computing library.
Overfitting	-	The phenomena where a machine learning algorithm overgeneralises on the provided training dataset.
Pandas	-	A data handling Python extension.
PyTorch	-	A machine learning development package for python.
Random Forest	-	An ensemble of Decision Trees that make predictions based on the cumulative result of the constituting models.
Regularisation	-	The process of constraining/restricting a machine learning model in order to reduce overfitting.
Scikit-Learn	-	A Python based data processing and machine learning development package.
Synapses	-	The structures within biological neurones that enable the communication between adjacent neurones.
T-Piece	-	A common fluid pipe geometry with a characteristic orthogonal junction forming a shape that resembles that of the character *'T'*.
Underfitting	-	The phenomena where a machine learning algorithm can neither fit or generalise on the provided data.
Volatile Memory	-	A form of memory that requires a constant supply of power in order to retain data.

Nomenclature

Greek Characters

α - Predetermined Constant

β - Friction Hyper-Parameter/Layer Shifting Parameter

γ - Layer Scaling Parameter

δ - Kronecker Unit Vector

ϵ - Smoothing Term

ε - Turbulence Dissipation Rate

θ - Activation Function

η - Learning Rate

μ - Dynamic Viscosity

μ_B - Empirical Batch Mean

μ_t - Coefficient of Eddy Viscosity

ν - Gradient Square

ρ - Volumetric Density

σ - Turbulence Prandtl Number

σ_B - Empirical Batch Standard Deviation

$\bar{\bar{\tau}}$ - Viscus Stress Tensor

Roman Characters

b - Bias term

c_p - Constant Pressure Specific Heat

c_v - Constant Volume Specific Heat

$\mathcal{C}$ - CART Cost Function

E_{ij} - Rate of Deformation

f - Neurone Magnitude

$\vec{g}$ - Gravitational Field Strength

$\mathcal{J}$ - Mean Absolute Error

k - Thermal Conductivity

K - Turbulence Kinetic Energy

$\mathcal{L}$ - Mean Squared Error

m - momentum term

M - Molecular Mass

n - Molar quantity

N - Sample Size

P - Fluid Pressure

R - Gas Constant

Re_t - Turbulence Reynolds Number

t - Time

$\vec{u}$ - Velocity Vector

V - Volume

w - Weight Scalar

$\bar{W}$ - Weight Vector

x - Neurone Instance

$\hat{x}$ - Standardised Neurone Instance

$\bar{X}$ - Layer Instance Vector

y - Exact Output Value

$\hat{y}$ - Hypothesised Output Value

$\hat{z}$ - Rectified Neurone Input

List of Abbreviations

Adam	-	Adaptive Moment Estimation
ANSYS	-	Analytical Systems
Bagging	-	Bootstrap aggregation
CART	-	Classification And Regression Tree
CFD	-	Computational Fluid Dynamics
CPU	-	Central Processing Unit
GPU	-	Graphics Processing Unit
LLE	-	Local Linear Imbedding
MAE	-	Mean Absolute Error
ML	-	Machine Learning
MLP	-	Multilayer Perceptron
MSE	-	Mean Squared Error
NN	-	Neural Network
PCA	-	Principal Component Analysis
RAM	-	Random Access Memory
RANS	-	Reynolds-Averaged Navier Stokes
ReLu	-	Rectified Linear Unit
RMSE	-	Root Mean Squared Error
RMSProp	-	Root Mean Square Propagation
RSS	-	Resident Set Size
SST	-	Menter's Shear Stress Transport

Introduction

The use of computational physics simulations are an essential tooling method that allow engineers to accurately predict solutions for relatively complex and undefined systems. These simulations are advantageous as the data can be visualised and manipulated to provide a more detailed description of the underlying state of the analysed system. However, often these physics simulations are computationally intensive and take significantly long to resolve (particularly in systems that have high degrees of freedom) [1]. An adverse result of these physical limitations is that often these simulations require the processing capabilities of very powerful computers capable of calculating and storing the associated information [1]. In addition to the financial cost implications, these computing systems introduce, the extensive simulation time requirements limit the application potential of such systems (as they are unable to be used in situations that require expeditious results). This means that it would be infeasible to implement these physics simulations into industrial settings because they cannot be reliant on producing real-time solutions. Furthermore, the associated temporal limitations prevent some such applications like design space exploration which can be used to enhance the design and operations of the relating system (e.g. the identification of design parameters and potential locations for ideal sensor placement). Other restrictions that arise due to the limitations of physics simulations are; the inability to efficiently detect system anomalies, conduct *'what-if'* analysis on unknown or unexplored parametric conditions, and to instantaneously perform model-predictive control (which requires almost immediate control optimisation) [2].

An example of such physics simulation is the application of Computational Fluid Dynamics (CFD) to resolve the unpredictable nature of fluid systems. CFD is an essential tool for developing very accurate visual and informative detail regarding the state of a dynamic fluid system. Computational fluid dynamics has very useful applications including calculating the motion and mixing behaviour of turbulent fluid flow, understanding the dynamic interactions between control surfaces and fluid bodies, as well as being able to visualise heat-transfer effects within fluid systems. This computational tool works by applying mathematical models to a specified system in order to simulate the dynamics and fluid flow behaviour in a realistic sense [3]. If the initial setup of the CFD simulation is correct, these iterative processes can produce very accurate results which are representative of real fluid system interactions. CFD simulations are a vitally essential modern engineering tool, as it allows mechanical systems that interact with fluids to be better understood and optimised. Although, because this form of analysis is computationally expensive, a trade-off is made between the accuracy of the data obtained, and the time taken to perform a simulation. Consequential to this trade-off, is the advent of the problems associated with limitations inherent in many physical simulations. This results in the application of CFD for the analysis of systems with varying flow parameters being inefficient and relatively impractical to implement in near real-time [4]. Although the high computing times and operating inefficiencies limit the applicability of CFD, these drawbacks can potentially be overcome with the aid of machine learning.

Machine learning (ML) refers to any algorithm that is capable of learning/improving its prediction results when exposed to a sample of information [5]. These programs are particularly useful in complex applications that do not have a definite algorithmic solution. A variety of machine learning models have been developed, each with unique characteristics that enable the algorithm to perform optimally for a given scenario. These functions can be categorised as either: supervised, unsupervised, or semi-supervised learning models [2]. Predominantly machine learning is used to solve problems requiring the classification of sample groups, or the numerical regression analysis of data [5]. ML algorithms learn by correlating trends in large samples of data. These data trends can often have highly polynomic degrees of dimensionality, which may be too complicated to program/compute using conventional methods. Given a large enough training set ML algorithms are capable of accurately predicting, interpolating, and (to a certain extent) extrapolating these seemingly complex data correlations [6]. An advantage to using a well-trained machine learning model is that the resulting functions are capable of producing comparatively accurate and precise results in a relatively short time period. Although machine learning functions are capable of predicting the outcomes of seemingly complex and non-algorithmic problems, these computing programs operate by learning the trends and correlation of the data rather than taking a brute force approach. The advantage to this learning method, allows for these very complex data trends to be represented with relatively simple algorithmic structures. Subsequently enabling ML algorithms to make extremely accurate predictions in almost imperceivable timeframes. The nature of these algorithms also enable them to be implemented and executed with a relatively low computational expense, as the static and operational memory footprint these simplistic algorithms engender can be very small.

By training a machine learning algorithm on data that has been sourced from a CFD simulation, the ML algorithm may potentially be able to reconstruct the original complex fluid flow data. An advantage to using these prediction models is that the resulting algorithms can be used for near real-time performance monitoring [7]. The small computational footprints associated with ML can allow very intricate processes, like CFD simulations, to be implemented on simplistic/rudimentary computers. This enables engineers to visualise and monitor the operation of complex networks almost immediately after the associated variables are adjusted. The ability to promptly analyse the performance of a system can provide a better understanding for previously unknown *'what-if'* cases regarding how the system in question may function. Along with real-time visualisation, this computational tool can be used as a means of anomaly detection; to further understand how the ML algorithm deviates with respect to the simulation from which the training data was sourced. The ability to monitor such systems to this extent enables ML to be utilised as a method for improving the understanding, safety, and control of very complex fluid systems. Although machine learning algorithms may potentially be applicable in solving CFD systems, these algorithms require extensive amounts of original training data in order to learn the trends of a specific system. This necessitates hundreds of simulations with varying parametric conditions to be simulated prior to the training of such ML models. Thus for ML algorithms to be implemented as a method of in-situ, real-time fluid motion analysis, these algorithms are to be developed in parallel with these CFD models as a compressed representation of the original fluid system.

The application of machine learning in engineering fields is a relatively unexplored subject with a potentially vast range of application [8]. Machine learning is a powerful computational, as well as engineering tool that can be used as an aid to optimise design processes, and better enhance methods for safety monitoring and assurance. These ML models can also be implemented with a relatively low computational and financial impact, as the facilities/hardware required to host such algorithms do not demand very complex and cost-intensive processing capabilities. This lucidity further increases the potential use of these learning models, as the associated efficiency and portability allow ML systems to be implemented and adapted in parallel with already existing engineering projects.

This report details the data acquisition, development, and experimentation of a machine learning model capable of accurately predicting the flow regime of a simple 2D mixing domain in near real-time. To overcome this problem, several different ML models will be constructed as a means of deducing whether machine learning is a viable solution to enabling near real-time CFD analysis, as well as to whether it is viable for these models to be portably implemented in existing engineering systems. By developing and testing different ML algorithms (specifically a Neural Network, Random Forest, and Autoencoder); the viability of these models in particular for this specific fluid dynamics application can be determined. These models are to be compared based on the total error and precision each algorithm produces when provided the initial training and testing CFD data, as well as each models ability to predict extrapolated fluid data. Along with the accuracy and precision measurements the algorithms will be examined on their memory and temporal impact given the provided data. The selection of such algorithm can be used as a proof that ML has the potential to be used, in conjunction with physical simulations, as a potential solution to solving complex iterative tasks for the applications of anomaly detection, visualisation of unknown instances, and real-time flow analysis. Along with these proofs, this report will also particularise the potential implementation methods of these algorithms. As this is necessary for the application of such ML models in current existing fluid based engineering systems.

Problem Statement

Scope

In order to investigate whether ML algorithms can be used as a form of real-time visualisation, performance monitoring, and anomaly detection for CFD simulations; the ML algorithm must be trained on a sufficient dataset. This data will be acquired by analysing a simple T-piece mixing setup using ANSYS Fluent simulation software [9]. The simulation will highlight the mixing behaviour of incompressible air flow. Figure 1 shows the design profile for the T-piece mixing domain which is to be analysed:

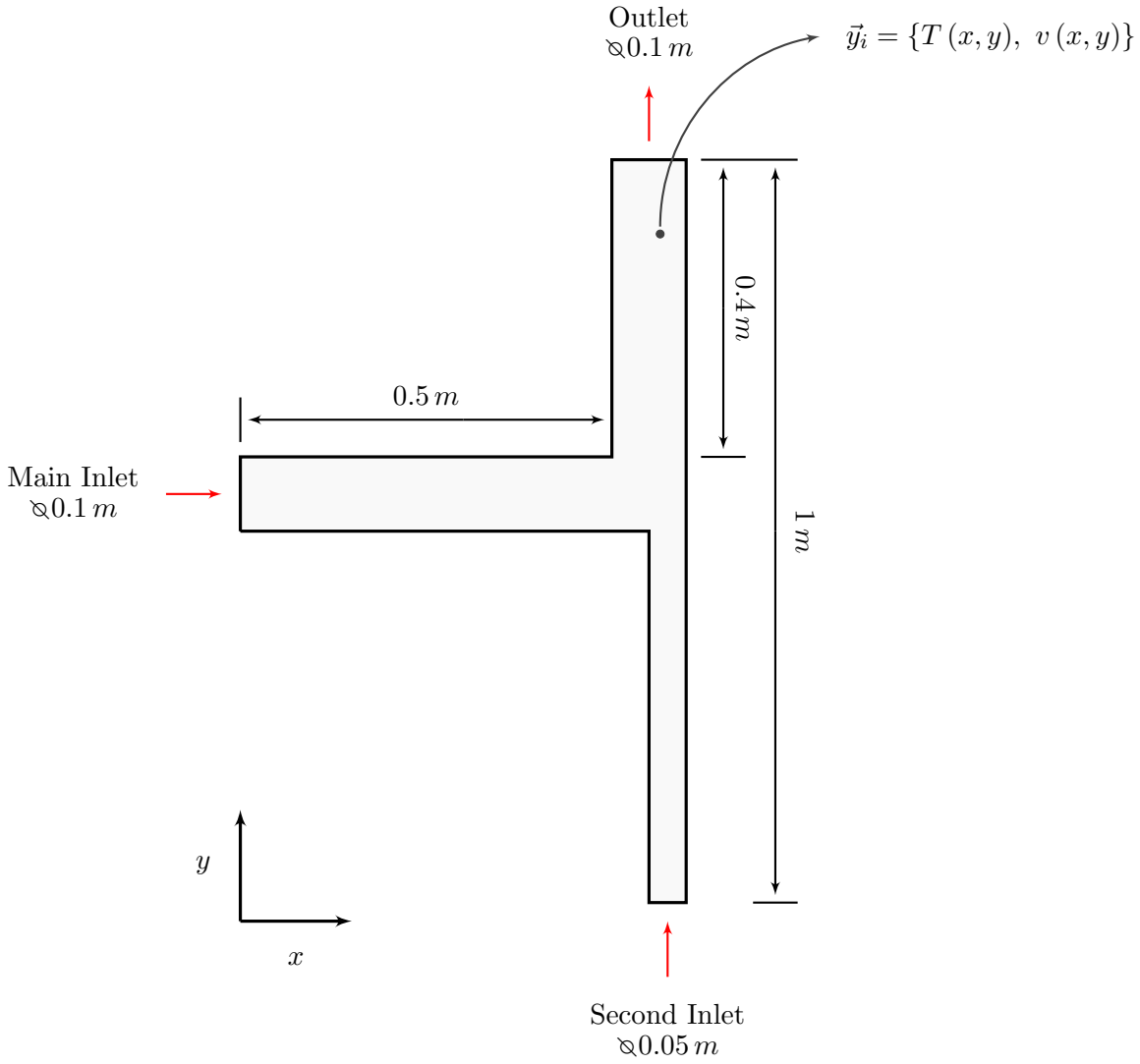


Figure 1: Scale diagram of dimensioned T-piece to be analysed

As shown in the two dimensional diagram above[1], by varying the input velocity and temperatures for the main and secondary inlet, an output dataset for the resulting steady state temperatures and velocities at each mesh cell can be obtained. By automating the simulation, this dataset can be expanded to provide solutions for a variety of input conditions.

The dataset obtained from the computer simulation will be separated (into training, and validation samples), from which an analysis of the data characteristics can be performed. By assessing the CFD data, a detailed approach as to which machine learning algorithms are best suited for this particular regression problem can be formulated. This information is to be wrangled into a data frame that is compatible with Python built machine learning frameworks [2] (Keras; TensorFlow; Sci-Kit Learn; etc...).

Three main regression algorithms will be evaluated (namely: an Artificial Neural Network, a Random Forest regressor, and an Autoencoder). These selected algorithms will be trained and validated using the acquired CFD data.

The selected models will be compared by the following performance criteria:

- The prediction error and precision of each model is to be documented and compared. Each algorithm is to be assessed on its ability to accurately predict the solutions to the original CFD dataset (from which each algorithm was trained).

- The overall runtime and operating memory usage for varying data quantities will be assessed.

- The model performance on outlying data, for both above and below the original training data range will be examined, as this determines each algorithm's ability to generalise on unseen outlying data instances.

The resulting CFD contours, as well as the overall statistical data for each algorithm is to be used as the medium for the performance comparison. This is necessary to determine what particular CFD application each algorithm is specifically suited for. In the analysis of each algorithm the generalisability, as well as the algorithms potential to be remotely implemented for real-time analysis is to be examined[3].

Using the results of the final algorithm analysis; implementation strategies/methods will be discussed. These methods will detail the possible ways for executing the trained models on external micro-controlling devices. This is essential if similarly trained algorithms are to be implementation in fluid based mixing systems, as these ML models can potentially benefit current in-place engineering systems.

Based on the observations made by the assessment; conclusions will be developed as to which ML algorithm is best suited for the particular application of predicting CFD simulations, as well as to whether these algorithms show potential to be implemented as a viable method for near real-time data visualisation and anomaly detection.

[1] Analysing the CFD simulation in two dimensions allows for a more simplistic flow system. Due to time constraints this simplified setup allows for more data to be acquired over a shorter period, and at a lower computational expense.

[2] By using built-in libraries sufficient project time can be saved, in lieu of building the algorithms from scratch which is more exhaustive.

[3] The structure of this project methodology can be further illustrated through the use of process flow diagrams (*see Appendix I*).

Limitations

Due to the vast application of computational fluid dynamics, this project will only focus on a simple two dimensional mixing scenario. This is to ensure more time and data can be allocated to the research and development of the appropriate machine learning models. The dimensions of the T-piece along with the physical properties of the mixing fluid will remain constant. This is because this simulation will only emphasise the effects of varying the velocity and temperature parameters, as these are the necessary parameters required for effective mixing. It will also be assumed that the fluids in question are incompressible (*see Literature Review*), this is due to the fact that the governing equations for incompressible fluid motion is generally simpler than that of compressible fluids [3]. Furthermore, for computational simplicity, only the steady state flow condition will be assessed when modelling the mixing of the T-piece system. As this results in faster CFD solution times, which allows for more simulation data to be obtained for the training of the ML model.

Although machine learning algorithms are capable of performing complex predictions in relatively short periods of time, these algorithms need to be trained on a sufficiently large dataset. This means that the accuracy of the ML model is limited by the size and complexity of the training samples. As a result the algorithm best suited for this particular two dimensional application may not necessarily scale as well as other prediction models for higher dimensional problems. This project will also only compare the results of the machine learning predictions to the CFD simulation from which the training data is sourced. Therefore this report will not detail the accuracy of these learning algorithms when compared to realistic experimental observations.

It must be noted that the three algorithms which are to be developed and evaluated (Neural Network, Random Forest, and Autoencoder), are not an exhaustive representation of all the potential ML models that can be applied to this particular CFD application. These algorithms were chosen to be tested based on information gathered from literature, the nature of the engineering problem in question (i.e. this fluid system is inherently a multidimensional regression problem), and by intuition. Furthermore during the development of each model, various machine learning methods and practices are to be applied prior to the actual implementation and analysis. This is to ensure the algorithms are performing near optimally for the given training set, before they are to be critiqued (i.e. the models are not overfitting, and the finalised algorithms produce acceptably low errors). However due to the extensive range of different algorithm development techniques, the selected models from each ML class may not be uniquely accurate, as various different methods can be applied to produce functions of similar results.

Literature Review

Conservation Equations and CFD Setup

In order to properly simulate a realistic fluid flow environment, the dynamics of the fluid flow must be understood on a finite scale. This can be achieved by analysing the Eulerian based conservation laws of the fluid. The Navier-Stokes equations are principal formulas used to resolve these conservation laws, as they characterise the physical parameters that influence the behaviour of the fluid [10].

For steady-state incompressible flow regimes the mass of the fluid entering the control volume is equivalent to the outlet mass. Assuming the volumetric density of the fluid is constant throughout the control volume, the general mass conservation equation (for a differential 3D flow element) can be described by the following equation [11]:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}\,) = 0 \tag{1}$$

Where:
∇ = Gradient operator $[\frac{\partial}{\partial x_i}\hat{i} + \frac{\partial}{\partial x_j}\hat{j} + \frac{\partial}{\partial x_k}\hat{k}]$
ρ = Fluid density $[kg/m^3]$
t = Time $[s]$
$\vec{u}$ = Fluid velocity vector $[m/s^3]$

This formula accounts for the rate at which mass is entering and exiting the control volume. It is evident from the equation that the change in mass of the system is equivalent to zero. This implies that the total inlet and outlet flow rates must be equal.

Similarly to the mass conservation of the control volume, the total momentum of the system must be conserved in order to comply with Newton's second law of motion. The conservation of momentum, for an incompressible Newtonian fluid, can be described by the formula:

$$\rho \left(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} \right) = \rho \vec{g} - \nabla P + \mu \nabla^2 \vec{u} \tag{2}$$

Where:
$\vec{g}$ = Gravitational field strength $[m/s^2]$
P = Fluid pressure $[Pa]$
μ = Dynamic viscosity $[N\,s/m^2]$

This formula relates the product of the mass flow rate and velocity with the external forces (gravitational, differential pressure, and viscus shear) acting on the fluid element. When analysing a flow system the internal finite elements of the control volume must obey the above conservation equation. This is necessary to correctly determine the major, and minor loss factors that characterise the flow dynamics [3].

The third conservation equation, required to replicate natural fluid flow, is the law of conservation of energy. Assuming the finite control volumes being analysed are incompressible (i.e. the element volume is constant) and that the conductive heat transfer obeys Fourier's law [12], the equation for the conservation of energy can be represented as follows:

$$\rho c_v \left(\frac{\partial T}{\partial t} + \vec{u} \cdot \nabla T \right) = -P \nabla \vec{u} + k \nabla^2 T + \bar{\bar{\tau}} \, \nabla \vec{u} \tag{3}$$

Where:
c_v = Constant volume specific heat capacity $[J/kg\,K]$
T = Fluid temperature $[K]$
k = Thermal heat conductivity $[W/m\,K]$
$\bar{\bar{\tau}}$ = Viscus stress tensor $[Pa]$

This equation relates the total thermal energy of the fluid (the sum of the enthalpy and internal energy) to externally introduced energy terms (i.e. the boundary work due to the pressure, thermal conductivity, and shear stress). The individual metrics of the viscus stress tensor can further be represented in the following relationship:

$$\tau_{ij} = \mu \left(\frac{\partial \vec{u}_i}{\partial x_j} + \frac{\partial \vec{u}_j}{\partial x_i} - \frac{2}{3} \frac{\partial \vec{u_k}}{\partial x_k} \delta_{ij} \right)$$

Where:
δ = Kronecker unit vector $\left[\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \right]$

The Kronecker unit vector is a term that arises from Stokes' hypothesis [12]. This includes an additional shear term along the main diagonal of the viscus stress tensor.

These conservation equations can be used to accurately model naturally observed fluid flow scenarios, and are generally the governing equations used when modelling finite CFD systems. However, in realistic fluid flow the formation of turbulent eddy currents cause the physical properties of the fluid (defined by equation (2)) to fluctuate. In order to appropriately analyse the steady state fluid motion as a result of these non-periodic oscillations, the time average of these fluctuations are calculated. The mean magnitudes of these stochastic flow parameters gives rise to the Reynolds-Averaged Naiver Stokes (RANS) models [13]. From these RANS equations many of the standard CFD based turbulence and wall treatment models are derived. The two most common partially analytical turbulence models are the K-Omega, and K-Epsilon methods [14].

Predominantly the K-Omega model is best suited for predicting turbulent flow fields near the walls and boundaries of control volumes, whereas the K-Epsilon model produces more accurate predictions far from boundaries (i.e. within the fluid body) [13]. These turbulent models can be combined using the Menter's Shear Stress Transport (SST) model. However this equation requires adjustments, which are time consuming and beyond the scope of the project. As a result the K-Epsilon model is to be used in the simulation environment (as this model provides a more accurate description of the flow characteristics at the mixing interface, and is typically used to model pipe flow systems) [13]. The K-Epsilon turbulence model can be expressed by the two dependent energy equations described as follows:

$$\text{K-Epsilon} \quad = \quad \begin{cases} \frac{\partial(\rho K)}{\partial t} + \frac{\partial(\rho K \vec{u}_i)}{\partial x_i} = \frac{\partial}{\partial x_j}\left[\frac{\mu_t}{\sigma_K}\frac{\partial K}{\partial x_j}\right] + 2\mu_t E_{ij}E_{ij} - \rho\varepsilon \\ \frac{\partial(\rho \varepsilon)}{\partial t} + \frac{\partial(\rho \varepsilon \vec{u}_i)}{\partial x_i} = \frac{\partial}{\partial x_j}\left[\frac{\mu_t}{\sigma_\varepsilon}\frac{\partial \varepsilon}{\partial x_j}\right] + 2\mu_t \alpha \frac{\varepsilon}{K} E_{ij}E_{ij} - \rho\beta\frac{\varepsilon^2}{K} \end{cases} \tag{4}$$

K = Turbulence kinetic energy $[J/kg]$
ε = Turbulence dissipation rate $[m^2/s^3]$
α, β = Adjustable constants [4]
σ = Turbulence Prandtl numbers
μ_t = Coefficient of eddy viscosity $[m^2/s]$
E_{ij} = Rate of deformation $[s^{-1}]$

When modelling a fluid at a finite level, the mass of the fluid can be described in terms of its volumetric density. This method is beneficial as the density can be described by the physical properties of the fluid. Knowing that the mixing fluid to be simulated is incompressible air, the properties and behaviour of the gas can be conveniently generalised by the Ideal-gas law as follows:

$$PV = nRT$$

Where:
V = Volume of gas $[m^3]$
n = Molar quantity of gas $[mol]$
R = Universal gas constant $[8.314\, J/\, Kmol]$

The density of the incompressible ideal gas can be represented by rearranging the above equation in terms of the substance's molar mass:

$$\rho = \frac{PM}{RT} \tag{5}$$

Where:
M = Molecular mass of substance $[kg/mol]$

This formula simplifies the computation of the entire control volume, as the result is independent of local relative pressures that act on the fluid [14]. This means that the density of the fluid element can be determined from the operating pressure of the system, rather than computing the pressures associated with each mesh cell. Thus the volumetric density determined from this ideal-gas law can be substituted into the conservation equations, so as to accurately predict the dynamic behaviour of the fluid.

To correctly predict the heat flow within the simulated gas, an accurate representation of the material's specific heat is required. For incompressible fluids the specific heat at constant volume can be approximated by the specific heat at constant pressure (i.e. $c_v \approx c_p - R$) [12]. This means that governing equations that are dependent on the principals of heat transfer such as equation (3) can be expressed in terms of the specific heat at constant pressure. By representing the specific heat as a temperature dependent piecewise-polynomial function, a more accurate expression for this heat capacity variable can be determined. The general form for this piecewise-equation can be described overpage:

[4]Note: these constants have been derived through experimental observation, and iterative computational processes relating to the dynamics of air [14].

$$c_p(T) = \begin{cases} A + BT + CT^3 + DT^4 + ... & a \leq T < b \\ E + FT + GT^3 + HT^4 + ... & b \leq T < c \end{cases} \tag{6}$$

Where:
$c_p(T)$ = Temperature dependent constant pressure specific heat $[J/kg\,K]$
A, B, C, D, E, F, G, H = Predetermined coefficients[5]
a, b, c = Specified temperature boundaries $[K]$

From the parametric equation above, it can be seen that for varying temperature ranges, a distinct c_p value is initialised for the system. Thus, this constant specific heat is to then be equated to the volumetric specific heat where it is to be subsequently referenced in the governing energy equations, as depicted in equation (3).

For turbulent flow, the interface of the wall boundary and fully developed fluid stream can be subdivided into two domains (namely a viscous sublayer and a turbulent region). Noting that the K-Epsilon model highlights the mixing behaviour within the fluid stream, it is imperative that the input parameters to this turbulence model do not allocate meshing resources to resolve the viscous sublayer at the wall boundary [14]. The diagram below shows the near wall flow behaviour for a turbulent stream:

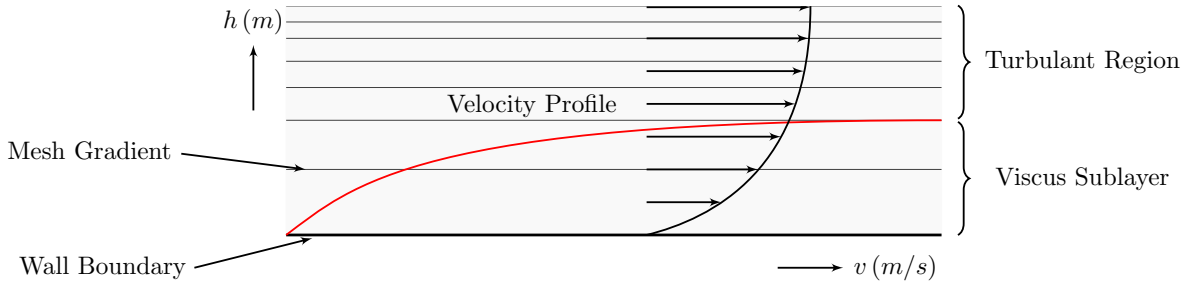


Figure 2: Standard wall function meshing for near wall treatment

Thus it can be seen, that by using the standard wall function as a near wall treatment, the logarithmic behaviour associated with the viscous sublayer can be approximated using fewer mesh cells. This inevitably reduces computational resources allocated to resolving the fluid state at the walls of the domain (consequently emphasising the dynamic characteristics within the fluid body) [14].

[5]Similarly to the parameters of the turbulence modelling equations, the temperature range, constants, and coefficients of the piecewise-polynomials have been empirically determined; in order for the computational solution to match realistic observations [14].

Machine Learning

Machine learning algorithms are programs that are capable of learning from experience. This experience is acquired by measuring the algorithm's performance with respect to a particular task. Thus machine learning is defined as a program that is capable of improving its prediction performance when exposed to a given task [5]. Machine learning models can broadly be categorised as either supervised, semi-supervised, and unsupervised; depending on the nature of the problem and the amount of supervision the algorithms require [6]. The type of model best suited for a particular task can be determined by the availability of training data and type of solution the final algorithm is required to output. These categorised algorithms can therefore be further subdivided as a result of the learning information available [2]. Figure 3 shows the types of subdivisions applicable to different machine learning models.

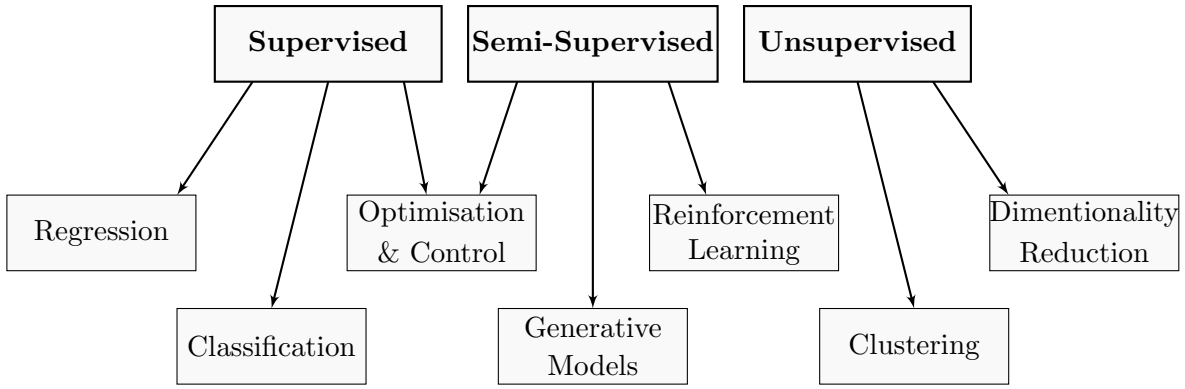


Figure 3: Machine learning categories and subclasses

In the context of the mixing T-piece, the ML algorithm is expected to predict the flow regime of the system from the given input characteristics (i.e. CFD boundary equations). The performance of this estimated output is to be compared to the results of the CFD simulation with the same specified input conditions. It is apparent from the structure of this particular nodus that the required ML program is a supervised learning task. This is evident as there is an available unique solution set (known as labels) from which the algorithmic approximation can be validated [6]. More specifically this problem is an example of a regression task as the program is expected to predict a numerical target value [6]. These predictions are in the form of temperature and velocity magnitudes measured at the nodes of each cell in the CFD mesh (*see Appendix II*).

Machine learning algorithms work by computing a series of given training instances (these features are typically denoted by the variable X). The model then predicts the corresponding output for each given input instance (these values are represented by the symbol Y)[6]. In a supervised machine learning scenario, the error of this hypothesised prediction is compared to the original output data [6]. The most prominent way of computing the prediction error of a machine learning algorithm is the mean squared error (MSE) cost function [15]. This error/loss function can be represented by equation (7) as such:

[6]For models that are required to solve multi-dimensional problems these input and output parameters are represented in the form of matrices $\bar{X}$ and $\bar{Y}$ respectively.

$$\mathcal{L}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y} - y)^2 \tag{7}$$

Where:
$\mathcal{L}(\hat{y}, y)$ = Mean squared error function
N = Number of samples in output set
$\hat{y}$ = Hypothesised output value
y = Exact output value

This cost function was derived from the formula for the standard deviation/root mean square error (RMSE) of a dataset. The MSE is typically used over the RMSE, as it is less computationally intensive to minimise whilst still maintaining analytical correctness [6]. Although the MSE is the most common error analytical tool, other methods such as the mean absolute error (MAE) may be more appropriate when solving data problems with many outliers (as the RMSE and subsequent MSE are more sensitive to outlying data) [6]. The MAE can be defined by summing the absolute error margins for all the data samples within the set, as shown below:

$$\mathcal{J}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}|\hat{y} - y| \tag{8}$$

Where:
$\mathcal{J}(\hat{y}, y)$ = Mean absolute error function

It is evident from equation (8) that by measuring the absolute difference between the hypothesised and exact outputs, rather than the square of the variance, a cumulative error metric that is less affected by large differences can be produced. This attribute can be more useful when computing very diverse datasets with large variances and outliers, as accurate measurements are not neglected by the larger imprecise data points [6].

Depending on the type of algorithm used; the measured prediction error is then evaluated and the machine learning function modified so as to minimise the error. By continuously correcting and minimising the prediction error, the algorithm develops an approximate function that best fits the training data. This correlated function can then be passed new unseen input variables and be able to accurately map these instances to the corresponding output variables [5]. These algorithms are particularly useful as they can be used to correlate complex linear, polynomic, and logarithmic data profiles. Typically the training of such algorithms can be categorised as either online or batched learning.

Online learning is useful in environments that demand regular adjustments due to the constant fluctuation of data, and when the available dataset is too large to be stored in the memory of the host virtual machine (known as out-of-core training) [6]. This learning system is beneficial in applications that do not have a readily available supply of training data (such as reinforcement learning) [6]. These processes work by training off of small groups of data known as mini-batches.

Batched learning systems are explicitly trained and validated offline. For relatively small datasets (less than 500 000 entries) it is conventional practice to separate the training and validation data in accordance with Pareto's' principle (i.e. a training-testing ratio of 80 % : 20 % [6]. Batched learning is generally time consuming, and requires a sufficient source of training data. However once these algorithms are trained, they can be implemented easily and with relatively low computational intensity [6]. Thus for the application of real-time flow analysis this form of training is most applicable, due to the available source of CFD data and the numerical efficiency of the resulting function [4].

Given the correct ML model and a sufficiently sampled training set, these algorithms are capable of predicting the trends of seemingly complex data distributions. However, due to the natural noise and variance that occurs in data sampling, oversimplified and unregularised ML models are not realistically useful as functional algorithms. This is because they are prone to underfitting and overfitting [5]. As a result these algorithms are incapable of generalising on new instance data as illustrated in figure 4 below:

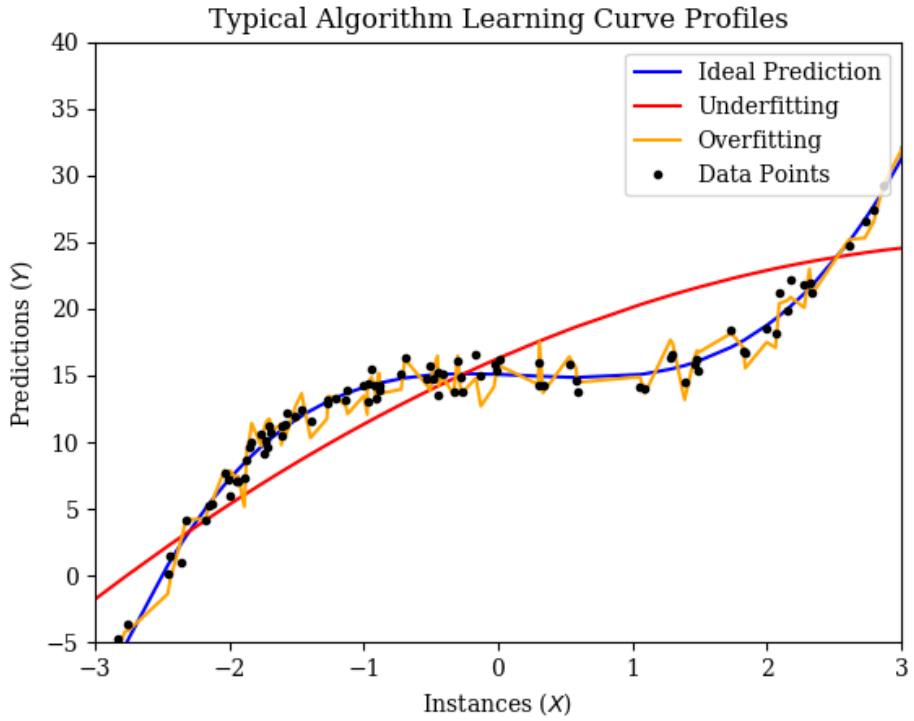


Figure 4: Machine learning training profiles for varying model structures

It is evident from the graph above that if the algorithm is too simplistic the function has a tendency to underfit the training data. Typically this occurs when the machine learning model is highly regularised (*see Appendix II*). This regularisation can occur when factors such as the model hyper-parameters[7] are overly restricted, or when there is an insufficient number of features from which the ML function is comprised [5]. It can be seen in figure 4 that the underfitting curve is incapable of matching the higher degree distribution line which the data points correlate to. Underfitting can be overcome by using a complex ML model with more parameters that can be modified, reducing the effect of regularisation hyper-parameters, and by training the algorithm using input features that have better correlations to the data profile [6].

[7]Hyper-parameters are parameters that can be modified to change the learning outcome of the ML algorithm [6].

Conversely to phenomena of underfitting; overfitting occurs when the approximation function generated by the ML algorithm over-compliments the training data distribution. This means that the model generalises particularly well on the given training set, but is incapable of accurately approximating new instances [5]. This is a result of the ML program mapping the output data to a higher degree than the original dataset. Overfitting can be observed when the prediction error of the training set converges to a relatively low value, whilst the validation error increases (indicating the model is struggling to predict the output values of the testing dataset). The effects of overfitting can be reduced by limiting the complexity of the ML model, adding regularisation parameters to inhibit the algorithm's sensitivity to outliers and variable data, reduce noise during the sampling of data, use larger datasets, and reduce the dimensionality of the data that the algorithm must predict [6].

In order to produce a very accurate model that is capable of matching complex data profiles whilst still generalising well for new instances, a trade-off between the model's complexity and regularisation must be made. This is known as the bias/variance trade-off [6]. The bias refers to the incorrect assumption of the functions complexity. This implies that the complexity if the algorithm is underestimated (producing a simpler and more restricted model). Variance refers to the function's tendency to overfit the data, as it results in a greater sensitivity to outlying data [5]. By balancing the extent of this trade-off ratio, an ideal structure for the model's complexity can be produced.

Machine Learning in Fluid Mechanics

Due to the computational efficiency of machine learning, these ML programs can be particularly useful when integrated in parallel with CFD systems. Current research into using machine learning in conjunction with CFD has proven to be very advantageous for engineering systems. Some such progressions in the field include; using these models as input and feedback mechanisms in the control of inherently unstable flow systems, as the relatively fast and accurate data processing associated with trained ML models, allows for very responsive and precise feedback [2]. Other ML characteristics (specifically associated with non-linear regression algorithms) have been exploited for the development of better simulation models. These properties allow for the production of more encompassing turbulence models, and as a means of optimising stochastic flow [16, 17]. Moreover, the high dimensional predictability of ML algorithms enable machine learning to be utilised as a means of dimensionality reduction; allowing these models to be used for applications such as fluid system architectural design optimisation, and the minimisation of surface drag acting on objects within regions of high velocity fluid flow [17, 18]. By utilising the high simulation accuracy of CFD and the relatively low runtimes of ML algorithms, machine learning has shown to be very useful for the predicting and correction of local errors in coarse grid CFD (improving the reliability and applicability of coarse element CFD simulations) [15]. Thus from these applications, it is evident that the use of machine learning in CFD systems is a powerful tool for anomaly detection, system visualisation, and real-time analysis of fluid dynamics.

More specifically to this project, research entailing the parallel utilisation of machine learning and CFD has been used in pipe junction mixing systems. This research involved using the high prediction speeds of ML regression algorithms to model the mixing phenomena in T-piece pipe networks of variant physical structure [4]. It was found that by varying the physical pipe flow parameters, the evaluated ML models (Support Vector Machine and Artificial Neural Network) were capable of developing relatively fast and accurate prediction results [4]. This research showed that the modelling of fluid flow in T-piece mixing systems is relatively complex due to the numerous variables that can contribute to the result of the final solution. Factors such as the length of the mixing system, the difference of flow between the separate inlets, and variations in pipe diameters can all affect the observed mixing behaviour [4]. This means many of these factors need to be controlled when conducting both a CFD and numerical analysis. Thus it is essential to predefine the physical structure of the mixing system (highlighted by this project), in order to ensure only the mixing results due to the variation of inlet conditions are assessed.

Machine learning has shown to be very applicable in CFD systems, particularly when modelling the high dimensionality, highly turbulent flow properties associated with fluid mixing [19]. Based on the available research into CFD and machine learning systems, it was noted that the most common/preferred non-linear regression models to be tested were: Support Vector Machines, Decision Tree Regressors, Random Forests, and Artificial Neural Networks. These algorithms are of particular interest as they are capable of mapping input instances to relatively high dimensional degrees, and have been known to successfully predict turbulent fluid motion [15].

From several performance comparisons made, it was found that although Support Vector Machines and Decision Trees were particularly good at generalising noisy data, and mapping inputs to relatively high dimensions; they did not perform as well as other regression algorithms [19] . It was found that the most accurate overall networks for predicting turbulent flow were Random Forests, and Artificial Neural Networks [4, 19]. Thus for the purposes of this project these two algorithms will be analysed, along with an Autoencoder[8].

[8]An Autoencoder is a specialised variation of a Neural Network, with the added advantage of non-linear dimensionality reduction (*see Materials and Methods*).

Materials and Methods

CFD and Data Preparation

In order to obtain the CFD data required to train and validate the machine learning algorithms, several factors regarding the CFD setup need to be considered. These include the design of the T-piece, surface meshing and boundary definition, the setting up and configuration of the simulation, and the automation of the CFD simulation and data capture. All these simulation criteria are to be done using ANSYS Fluent simulation software [9]; as this software package offers the design, meshing, simulation, and automation features required for appropriate CFD simulation and data acquisition.

Using the ANSYS new design modeller geometry feature [9], the T-piece diagram depicted in figure 1 is to be reproduced. It is evident from this diagram that the secondary inlet diameter is half that of the main inlet. These values were chosen as they are relatively similar in dimension to realistic standardised pipe diameters. Along with these inlet dimensions, the rest of the T-piece proportions are to be constrained to the specified set dimensions. This is to ensure the dimensional parameters that may affect the mixing behaviour, of the fluid, are kept constant. Once these bounding dimensions of the T-piece are defined, the surface of the design is to be generated. This process is essential as it initialises the area on which the CFD mesh and simulations are to be defined. It is important that this surface is designed and generated in a single plane as the simulations conducted thereafter, are to be done in a two dimensional domain. Thus in accordance with the image depicted in figure 1, the T-piece will be designed relative to the X-Y Cartesian coordinate plane (defined by the ANSYS new design modeller geometry canvas).

Once the surface geometry of the T-piece has been defined and initialised, this geometry is to be linked to the ANSYS meshing feature [9]. This feature allows a mesh to be imposed on the defined surface (*see Appendix II*). The mesh feature allows the entirety of the surface to be discretised (which allows the computer to resolve the fluid dynamics at each discrete step). To achieve an accurate simulation the mesh dimensions are to be set to a relatively fine size and uniform arrangement. As a result a linear mesh type of size $5\,mm$ is to be defined for the whole surface. This can ensure an accurate simulation result, whilst still maintaining a relatively low simulation runtime. Once the mesh has generated, the defining boundary conditions are to be initialised. This includes labelling the walls, outlet, main inlet, and secondary inlet. By defining the boundaries of the surface, the associated boundary parameters can be modified during the simulation component of the CFD setup. Following these procedures, the complete CFD mesh is then to be exported to the ANSYS Fluent application [9] where the CFD simulation is to be conducted.

To set up the simulation parameters, all the factors associated with the fluid dynamics theory are to be accounted for (*see Literature Review*). When setting up the simulation model and appropriate fluid variables, it must be noted that the energy equation is to be applied. This energy equation allows for internal heat transfer to take place between mesh cells. These equations acknowledge the various heat transfer processes within the fluid (such as conductive, diffusive, and viscous dissipative heat transfer processes) [14]. Other model parameters such as the turbulence model (K-Epsilon) are to be applied to the fluid. The variables associated with the initialisation of the K-Epsilon turbulence model will remain at the default setting, as these values have already been experimentally determined for computational accuracy. These model parameters are to be applied to the fluid in question (air). Furthermore the fluid will also be defined as an incompressible ideal-gas, and have a constant pressure specific heat define by the experimentally derived piecewise-polynomial function. These steps ensure the simulation produces a relatively accurate result, whilst reducing the overall temporal impact of the costly simulation process.

With respect to the T-piece section depicted in figure 1, it is evident that both the interior surface as well as the wall boundaries of the pipe system are clearly identifiable. These boundaries will provide the surface from which the fluid simulations are to be conducted. Furthermore it can also be noted that a uniform thermal constraint (a temperature of $800\,K$) is to be defined across the wall boundary of the design. By assigning a constant temperature at the walls of this CFD model, the temperature associated with the boundary fluid layer can be maintained. This temperature allows for the establishment of a region from which the thermal gradient of the mixing fluid can be generated[9].

To determine the reliability and validity of the CFD solution, an initial CFD simulation was setup and tested. The boundary conditions for this initial test were defined as follows:

Table 1: Initial design point boundary parameters

Boundary Parameter	**Velocity [m/s]**	**Temperature [K]**
Main Inlet	20	373
Secondary Inlet	50	298
Wall Boundary	N/A	800

Using these values the initial design point was set up for 1000 iterations on a duel core Intel I5 central processing unit (CPU). From this simulation the following features were noted:

- The meshing and surface generation took approximately 30 seconds.
- The 1000 simulation iterations took roughly one minute to completely solve.
- The simulation converged at approximately 185 iterations.
- A total of 5000 data points were computed and exported.

[9]These temperatures are to be determined using the conservation formula detailed in equation (3).

Based on the relatively few iterations required for successful convergence, it was noted that for later design points the simulation would not have to be configured for 1000 steps, as approximately $200 - 210$ iterations would suffice.

The exported data file consisted of the corresponding X and Y coordinates (as defined in the T-piece geometry modeller) for each mesh cell in the CFD solution. Along with the coordinate information, each data point represented the corresponding temperature and velocity magnitudes determined from the CFD simulation. This data allowed for the respective contour maps for both the temperature, and velocity regimes to be plotted. The figure below represents the respective contour graph for both the temperature, and velocity of the T-piece, in accordance with the CFD solution:

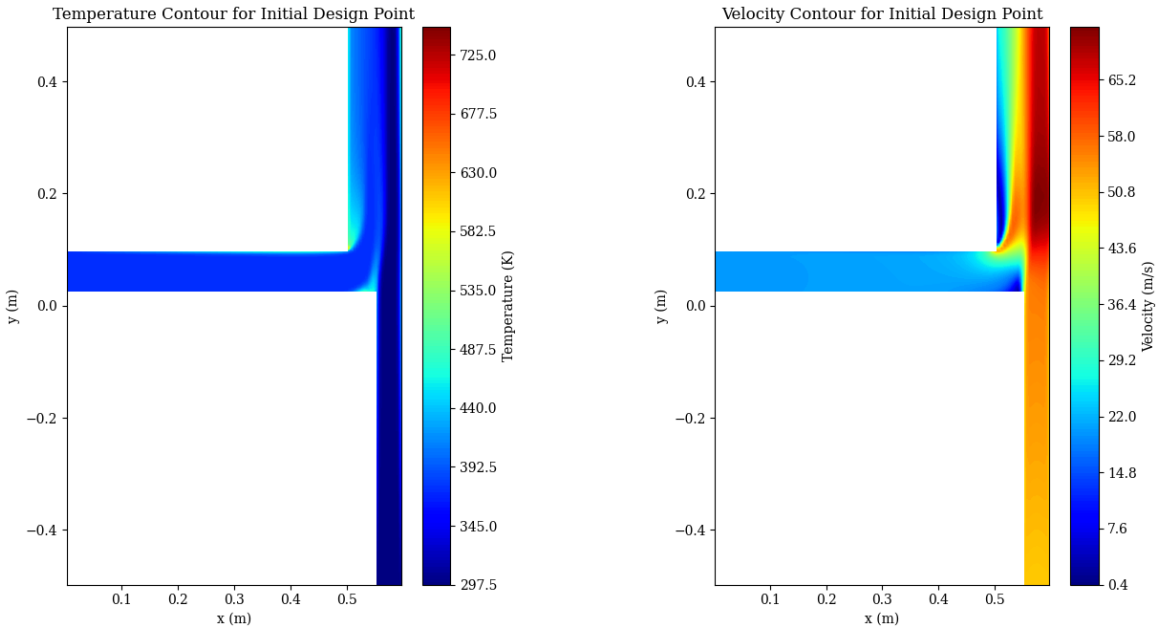


Figure 5: Initial CFD simulation contour solution

Thus from inspection of the above graphs, it is evident that the CFD results and corresponding setup conditions are acceptable for further development and implementation in machine learning.

Sampling the variable temperature and velocity for a wide variety of inlet flow parameters, could produce a vast datasets necessary for the training of the ML algorithms. The velocity and temperature values chosen were to be within a realistic and reasonable range. Table 2 shows the variable bounds which are to be sampled from the CFD simulation:

Table 2: Inlet boundary ranges used in mixing T-piece

Boundary	**Temperature [K]**	**Velocity [m/s]**
Main Inlet	298 - 600	5 - 20
Secondary Inlet	298 - 600	20 - 70

These inlet conditions were selected to ensure the data sample range spans the transitional and turbulent phase of the analysed fluid (air). This data trend diversity can allow the ML models to be trained and validated on a vast range of flow conditions, which can resultantly produce generalised algorithms capable of predicting radically different flow characteristics.

To acquire a sufficient amount of data for the ML models, the data extraction process was to be automated. This was achieved using the Design of Experiments feature available on the ANSYS simulation software [9]. The input conditions of the T-piece were then to be parameterised in accordance with the variable ranges indicated in table 2. To ensure a variant dataset with fairly good data dispersion, the parameters were distributed using a Latin-hypercube arrangement. This arrangement ensures all the parameters bounded by the instance limits were organised in a way such that; the space relative to all four input parameter ranges were randomly distributed. The stochasticity of this data arrangement ensures the machine learning algorithms do not over associate/train on a specific set of inputs, as this would result in a biased algorithm that will subsequently be incapable of generalising on the entire training set. Using this input arrangement, 1000 simulations could be conducted and automated using the Design of Experiments feature. Each simulation was configured for 210 iterations, and the resulting solution design point data files were saved to a local repository.

Before the CFD data can be applied to the respective machine learning algorithms the data first needs to be loaded and preprocessed in Python [20]. This preprocessing includes the normalisation and splitting of the training and testing data. The most efficient method for this data manipulation is by implementing the various processing steps in a data pipeline (*see Appendix II*). This method involves executing each required process in a sequential fashion. An advantage to using such pipeline is that the debugging and correction of large datasets can be easily performed in an efficient and structured manner. This data preparation can be further simplified with the aid of Scikit-Learns's built in features [21]. These libraries can be utilised for processes such as normalisation, and dataset splitting. The following algorithm shows the implementation steps required to prepare the CFD data files for ML training:

Algorithm 1: Data Extraction and Preparation

Data: 1000 separate CFD data files and associated input data arrays

`Data Split Function(`*Input Data, Output Data*`):`
⇒ shuffle and split function parameters into testing and training dataset
⇒ save each dataset as individual dataframe

for *Data File* **in** *Folder:*
    rename data file according to corresponding design point
    extract CFD data from data file, and append to common dataframe
    transfer displacement coordinate data into separate array
end

save coordinate data array
save temperature and velocity dataframe as output data

with *Input Data & Output Data:*
    create and export arrays of minimum data
    create and export arrays of maximum data
    fit and transform data using Scikit-Learn MinMax scalar function
end

split training and testing dataset as 80 : 20 ratio `(1)`↩
convert data frames to NumPy arrays

Result: Export normalised training and testing NumPy arrays

The algorithm above clearly indicates the various steps and processes necessary for the preparation and normalisation of the CFD data. Due to this normalisation technique it can be seen that during the pipeline process a NumPy [22] array of the minimum, and maximum data points for both the input and output datasets are to be exported. This is necessary for the evaluation process, as the output predictions of the machine learning algorithms are required to be rescaled. Thus these minimum and maximum arrays are used to map/correlate the predicted trends to their representative values (from which the CFD contours and prediction errors can be graphed and evaluated respectively). It can also be noted that the data set was of sufficient size to be separated into an 80 % : 20 % ratio. This is to ensure the sample size of the validation dataset is large enough to provide a reliable validation error. Thus following the procedures listed above, the data can be processed from the simulation material, to the necessary data structures required for the application in machine learning.

Neural Network

Neural Networks (NN) are a class of artificial learning algorithms, with a neural architecture based loosely on the biological structure of the human brain [23]. Each network is comprised of interconnected layers of numeric quantities known as neurones. These structures are designed to replicate the communication hardware between the synapses[10] and neurones present in the brain [24]. The diagram below shows the fundamental operation of an artificial neurone:

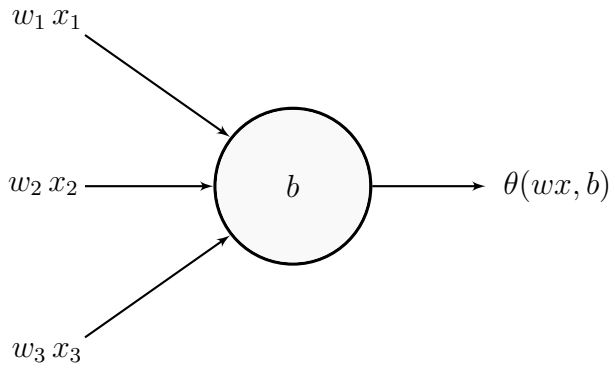


Figure 6: Functional operation of an artificial neurone

It is evident from the above figure that the neurone receives a series of values corresponding to a given variable x_i scaled by a constant value w_i (these scalars are conventionally referred to as weights). Every input of the neurone is computed, and a unique bias is added to this particular solution, depending on the relevance it possesses within the overall network. Given an n column vector matrix $\bar{X}$ of input instances, and a corresponding scaler matrix $\bar{W}$; the resulting neurone magnitude/value can be represented as a linear combination of the scaled inputs and bias terms:

$$f(wx,\, b) = \bar{W}^T \bar{X} + b = (w_1 x_1 + w_2 x_2 + \ldots + w_n x_n) + b \tag{9}$$

Where:
$f(wx,\, b)$ = Neurone magnitude[11]
b = Bias term

In a Neural Network these neurones are organised into multiple layers, each of which are interconnected to every other neurone in the adjacent layers. The advantage of this structure is that deeply nested networks are capable of computing high dimensionality/non-linear functions [1]. The ability to map inputs to higher dimensionalities is what makes NN particularly desirable for solving the inherently polynomic behaviour of fluid motion [1]. Along with the dimensionality characteristics associated with Neural Networks, these machine learning algorithms are also capable of processing noisy data, with reasonably high computational efficiency [24].

[10] Synapses are more appropriately known as synaptic terminals, and are the structures within biological neurones that enable inter-neural communication [6].

[11] Note: for illustrative purposes, the parameters of the neurone magnitude function are henceforth to be voided.

Neural Networks function by computing the values of each neurone in the first layer (given the corresponding input values). The calculated magnitudes of each neurone is then appropriately transformed using an activation function. The purposes of these activation functions are to scale the data in order to improve the training efficiency and prediction accuracy of the network [6]. The most common activation function used in regression problems is the Rectified Linear Unit function (ReLu). This function bounds the neurone value between 0 and ∞, which makes it particularly useful when solving for a range of numerical quantities. By applying the Rectified Linear Unit function to the equation of the neurone value, the resulting neurone output can be represented as such:

$$\theta(f) = \max\{0,\ f\} \tag{10}$$

Where:
$\theta(f)$ = Activation function

The advantage of using ReLu in regression problems is that; the magnitude of all positive values remain unchanged. Thus applying the ReLu activation function to the magnitude of the neurone has no effect on information loss (granted the input parameter is positive). It must be noted that this activation function is also differentiable, allowing the weights of each neurone to be adjusted during backpropagation[12] [5].

In most conventional NN the outputs of these activation functions are then used as the individual inputs to the neurones in the following network layer. The process is repeated until the values of the final layer (output layer) are computed. This sequential dynamic connection is known as a feedforward network [25]. The layered Neural Network structure can be shown in the annotated diagram below:

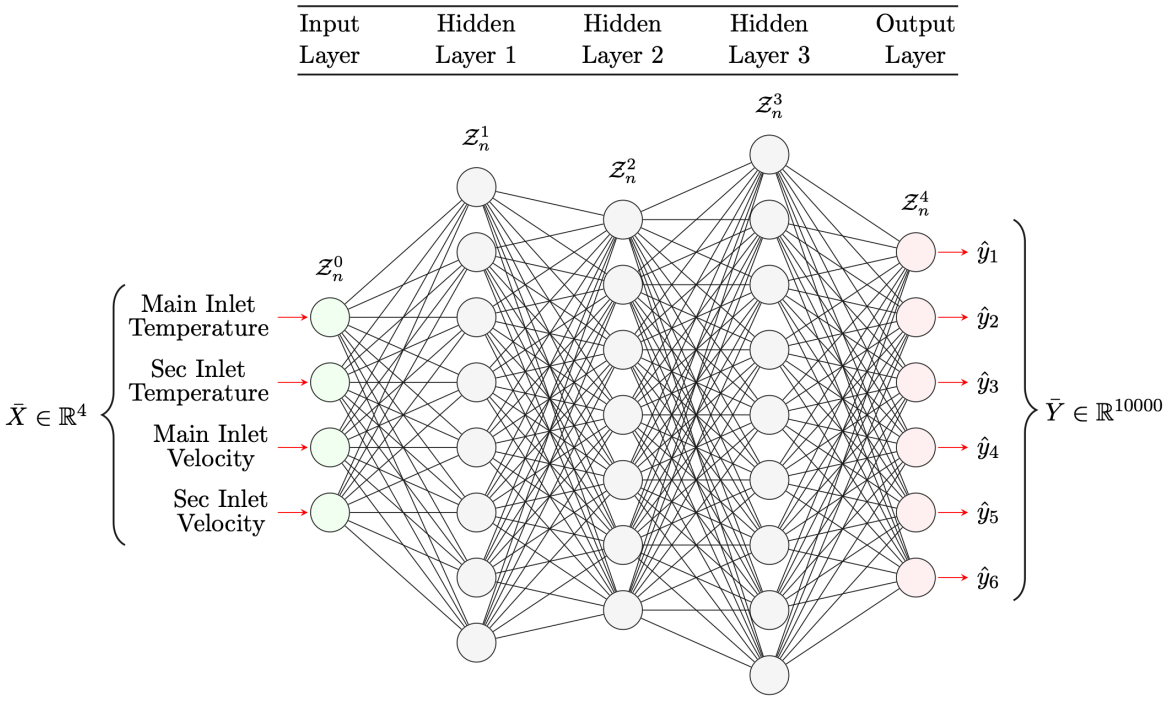


Figure 7: Standard Model of a Neural Network

[12]Backpropagation is the process of modifying the relative weights of a Neural Network in order to optimise the output accuracy.

From figure 7 above, it can be seen how the inputs to the particular T-piece CFD system are to be fed into the network. It is also evident how each neurone layer ($\mathcal{Z}^i$) forms the inputs to the ensuing layer ($\mathcal{Z}^{i+1}$). By iterating this feedforward process through multiple network layers the resulting function at the output can be represented as a nested/compounded function of the input instances. This is what resultantly gives the NN its multi-dimensionality characteristics.

However although the network is capable of mapping inputs to varying dimensional states, the algorithm must be capable of modifying the output prediction to match the trend of the labelled data. This process is achieved through backpropagation, and is what inevitably allows the network to learn prediction trends relating to the provided data. To do this the error of the network prediction must be computed using the mean squared error cost function described by equation (7) (*see Literature Review*). In order to maximise the error reduction of the output prediction, the gradient vector is applied to the MSE [26]. Subtracting this maximum gradient from every weight in the network, results in the overall prediction error reduction and subsequent learning behaviour of the algorithm. By representing equation (7) in terms of the Neural Network weight parameters, the following formula can be developed:

$$w = w - \eta \nabla_w \mathcal{L}\left(\theta(f),\, y\right) \tag{11}$$

Where:
$\eta =$ Learning rate

This form of optimisation is referred to as stochastic gradient descent , due to the stochastic/random nature of the gradient optimisation pattern [6]. It can be seen from equation (11) that the gradient optimisation term is scaled by a learning rate constant (η). This scalar acts as a form of regularisation to restrict the size of the reduction step made during each training iteration (epoch). The magnitude of this learning rate hyper-parameter can greatly affect the training behaviour of the algorithm, as the random nature of the gradient function my cause the prediction error to converge on a local minima (i.e. overfitting). This convergence matter can be illustrated in figure 8 below:

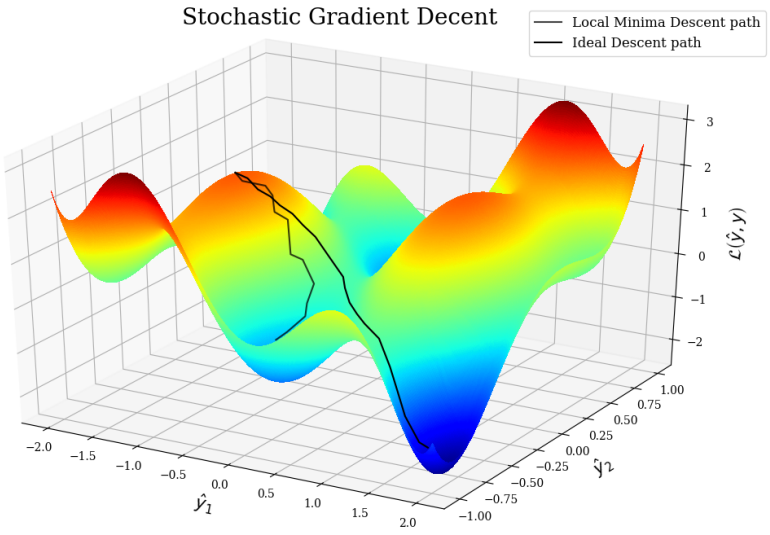


Figure 8: Gradient descent path of three dimensional surface

It is apparent from equation (11) and the above diagram that the application of a relatively large learning rate can create a very staggered descent path that results in the prediction error converging on a local minima. This is why tuning the learning rate hyper-parameter has a regularising effect on the algorithm's learning ability, and its sensitivity to noisy data [6]. To counter this issue and assist the learning process, data normalisation techniques are used as this creates a *'surface'*[13] contour that aids the convergence rate of the algorithm.

However, although a smaller learning rate can allow for an ideal descent path; this hyper-parameter also increases the overall training time of the algorithm. This is due to the smaller convergence steps which require more epochs to reach the global minima [6]. The added number of training iterations also increases the number of error function calculations that are performed. This further escalates the training time of the algorithm, as the updating of the individual network weights is a costly computational procedure [25].

To overcome this time consuming descent pattern several optimisation methods have been developed. Among these are the Momentum and RMSProp optimisers, both of these optimisation solutions make use of an added hyper-parameter. Momentum optimisation aims to maintain the momentum of the descent, this is achieved by adding a momentum term to each gradient [6]. The addition of the momentum hyper-parameter increases the magnitude of the learning rate step as the number of training iterations progress. Although one drawback to using the momentum optimiser is that the momentum term tends to cause overshooting and oscillations about the solution minima [6]. An alternative to the momentum optimiser is the application of Root Mean Square Propagation (RMSProp). This optimiser acts to normalise the gradients, in order to achieve a more direct convergence path [6]. The normalisation effect is achieved by dividing the stochastic learning rate term by a fraction of the most recent gradient vector. For very complex problems this direct path can significantly reduce the training time as opposed to the rudimentary stochastic gradient descent method [6].

In the optimisation of Neural Networks, the most widely used method for backpropagation is the Adaptive Moment Estimation (Adam) optimiser. The Adam optimiser uses a combination of both the momentum and RMSProp methods, to produce a faster learning model [5]. This optimiser can be described by the five equations below:

$$\text{Adam} = \begin{cases} m &= \beta m + (1-\beta)\nabla_w \mathcal{L}\left(\theta(f),\, y\right) \\ \nu &= \gamma\nu + (1-\gamma)\nabla_w \mathcal{L}\left(\theta(f),\, y\right) \otimes \nabla_w \mathcal{L}\left(\theta(f),\, y\right) \\ m &= \frac{m}{1-\beta^T} \\ \nu &= \frac{\nu}{1-\gamma^T} \\ w &= w - \eta m \oslash \sqrt{\nu + \epsilon} \end{cases} \tag{12}$$

Where:
m = Momentum term
ν = Gradient square
β = Friction hyper-parameter
γ = Decay rate hyper-parameter
ϵ = Smoothing term[14]

[13]Note: the surface depicted in figure 8 is a three dimensional representation of the multidimensional surfaces that are often associated with deep Neural Network prediction spaces.

[14]The smoothing term is a very small number that is added to the divisor in order to prevent an occurrence of a zero division error [6].

From the above equations, it is apparent that the modified weight term is a combination of the momentum term, and the gradient scaled learning rate (just like the momentum and RMSProp optimisers it was derived from). It can be seen that the momentum vector is a combination of the current momentum vector and the decaying average sum of the gradient vector [5]. This vector will consequently ensure the deterioration of the gradient magnitude does not slow down dramatically as the prediction solution converges. Similarly to the momentum vector, the square of the gradient is roughly defined by the same solution (with the addition of an element wise squared gradient vector). This element wise square ensures the overall gradient square vector remains positive, as a negative value may result in an increase of the modified weight the error. It is evident in the fifth defining equation of the Adam optimiser that; the combination of the momentum and gradient square terms produces a more direct descent path with characteristically larger reductions per epoch [5]. As a result of the vector initialisation, the third and fourth equations act to boost the momentum and squared gradient terms. This prevents these vectors from obtaining a biased close to zero [6]. Due to these gradient enhancement alterations, the Adam optimiser is typically the most common and sought after method used in the backpropagation of Neural Networks.

In terms of the using NN to predict the acquired CFD data, the Adam optimiser and MSE cost function were chosen as the ideal method for adjusting the network weights. This is due to the convergence efficiency this descent method offers. Along with this, each neurone in the network was to have the ReLu activation function applied to the output. This particular activation function is beneficial for application in regression algorithms as it is differentiable (and thus the gradients of the weights can be computed), as well as providing relatively noticeable training time reductions. The network depicted in figure 7 shows the general layout and implementation of the CFD related Neural Network. It can be seen that the four dimensional input array is passed into the feedforward network. This network then predicts the 10 000 dimension output contour array.

Using the optimisation methods discussed above, a simple feedforward Neural Network could be developed to predict the T-piece CFD contour given the specified input variables. From the detailed model selection process (*see Appendix III*) it was found that a 5 layered MLP (with 128 neurones in each hidden layer) was the best NN structure for predicting the specific CFD data. As a result, the performance of this selected model structure was to be further evaluated and compared in order to determine whether Neural Networks are a viable method for real-time anomaly detection, and in-situ CFD analysis.

Random Forest

Random Forests are a class of supervised learning algorithms which are notoriously fast to train. These programs are particularly simple to implement and can result in very accurate predictions. Random Forests work by grouping several, more simplistic; learning algorithms known as Decision Trees. The set that is formed by the grouping of such algorithms is referred to as an ensemble, and typically produces results that are more accurate than that of the algorithms which constitute the ensemble [6].

Decision Trees (the root components of Random Forests) are very adaptable learning algorithms that can be used in both regression, and classification tasks [6]. These algorithms are particularly desired as they are easy to implement and make predictions relatively fast (compared to other algorithm types) [27]. Another advantage associated with Decision Trees, is that the learning and prediction process is easy to visualise and understand [28]. Decision Trees have a very simplistic structure, namely they are comprised of interconnected nodes. These nodes diverge into subclasses known as daughter nodes, similarly to the structure of an inverted tree (hence the name Decision Tree) [28]. Usually these parent nodes divide into two daughter nodes, to which the inherited decision is passed. This method known as the Classification and Regression Tree (CART) algorithm, and produces a tree of binary relations [6]. The prediction process of these trees work by classifying the instances at each node and splitting the data according to the binary result of the classification process. This tree structure and splitting process is illustrated below:

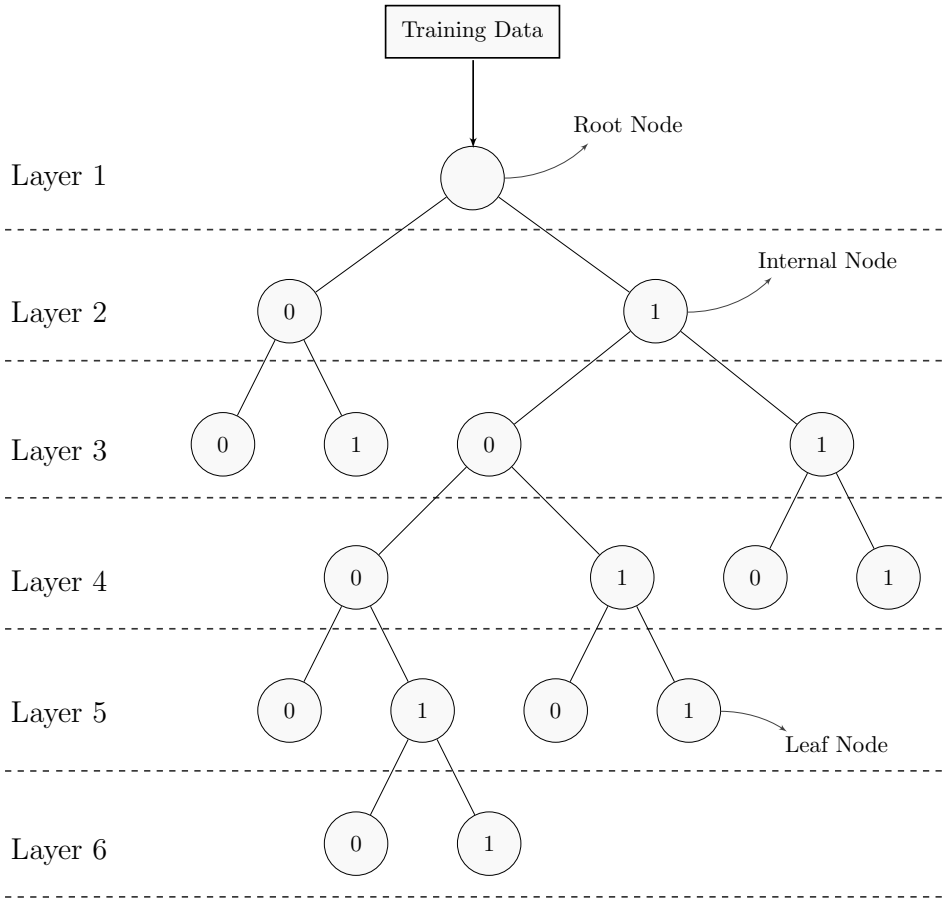


Figure 9: Annotated structure of a Decision Tree Regressor

It can be seen in the above diagram that the initial input instances are supplied to the root node, from which each decision splits. This subdivision of data is determined by the splitting criteria defined by the parent node [28]. For regression tasks this splitting criteria is developed using the CART cost function [6]. This cost function computes the total error by summing the MSE at each daughter node. This equation can be represented as such:

$$\mathcal{C} = \mathcal{L}(\hat{y}_{node}, \bar{y}_l)_{left} + \mathcal{L}(\hat{y}_{node}, \bar{y}_r)_{right} \tag{13}$$

Where:
$\mathcal{C}$ = CART cost function
$\bar{y}_l$ = Mean value for left node
$\bar{y}_r$ = Mean value for right node

It is evident that the error is computed by summing the MSE of the left and right nodes respectively. During the learning process, the splitting of the nodes are computed such that the result of the above equation is minimised. This ensures each node subsequently divides the instance data into the largest possible subsets [6]. This splitting process continues until the data is outputted at the terminal/leaf nodes at the ends of the tree, from which the corresponding result is used as the final prediction solution.

The advantages to this form of prediction, is that these algorithms are inherently capable of outputting very complex data structures (including very non-linear patterns such as the data associated with fluid mechanics). These algorithms also do not make any parametric assumptions on data, this allows the trees to develop very stochastic/informal structures [28]. Another benefit to using these learning structures is that little preprocessing is required (such as handling missing data values, and pre-scaling input data), as these factors do not drastically influence the algorithm's learning rate/prediction accuracy [6].

It can be noted from figure 9 that each decision is organised into sequential layers. These layers increment as the tree learns on the training data. One downside to this result is that if these trees are left unconstrained/unregularised, these layers will develop/grow to very large proportions [6]. The adverse effect of this growth is that the trees are heavily prone to overfitting. Although there are several ways of preventing this overfitting, such as restricting the depth and number of leaf outputs, the resulting predictions of these trees can often be affected. Due to these effects, and the predefined nature of these algorithms; Decision Trees are often very unstable [28] . This means that if the input data differs from that of the training data, then the outputs of the Decision Trees struggle to map to the correct solution. As a result of this instability, it is often infeasible to utilise Decision Trees for practical situations [6]. Thus to improve the accuracy and generalising capabilities of these algorithms; Decision trees are grouped into Random Forest ensembles to improve the system's overall generalising ability [28].

Random Forests typically consist of multiple Decision Trees, each with a unique and randomised structure [29] . In regression predictors, the outputs of each constituent Decision Tree is accumulated, and the mean value of this combined prediction is computed. This average prediction value is particularly significant as the resulting output is often more accurate than the individual Decision Trees that make up the Random Forest [6]. By computing the mean value of the random tree predictions, the invariant prediction characteristics of the individual Decision Trees can be eliminated [29]. This makes Random Forests very robust algorithms that are capable of generalising trends for very complex input data ranges, such as that of fluid mechanics.

Along with these complex data handling characteristics, Random Forests inherit many other advantageous traits associated with the Decision Trees (such as the relatively fast training times, and regularisation hyper-parameters) [28]. However this generalisability is dependent on the randomness of each Decision Tree, as each prediction is formed from the accumulated variance of each tree. Thus in order to produce a random ensemble, each tree in the regressor must be trained on a randomised subset of the original input data. This is achieved through the statistical sorting process of Bagging.

Bootstrap aggregation (Bagging) is a form of data replacement sampling [6]. During this process, data subsets are formed by sampling the input data required for the training of the learning algorithm. Each data subset is the same size as the original training input, however this bootstrapped sample may have reoccurring data points [28]. The advantage of this is that each bootstrapped sample is unique and random, but contain the same data as the original input set. By supplying every data subset to a separate Decision Tree, it is ensured that each regressor develops a unique prediction structure. This randomisation allows each tree to produce a unique output for a given input dataset (from which the mean output prediction is to be computed). The diagram below shows the intrinsic structure and data flow of a Random Forest Regressor:

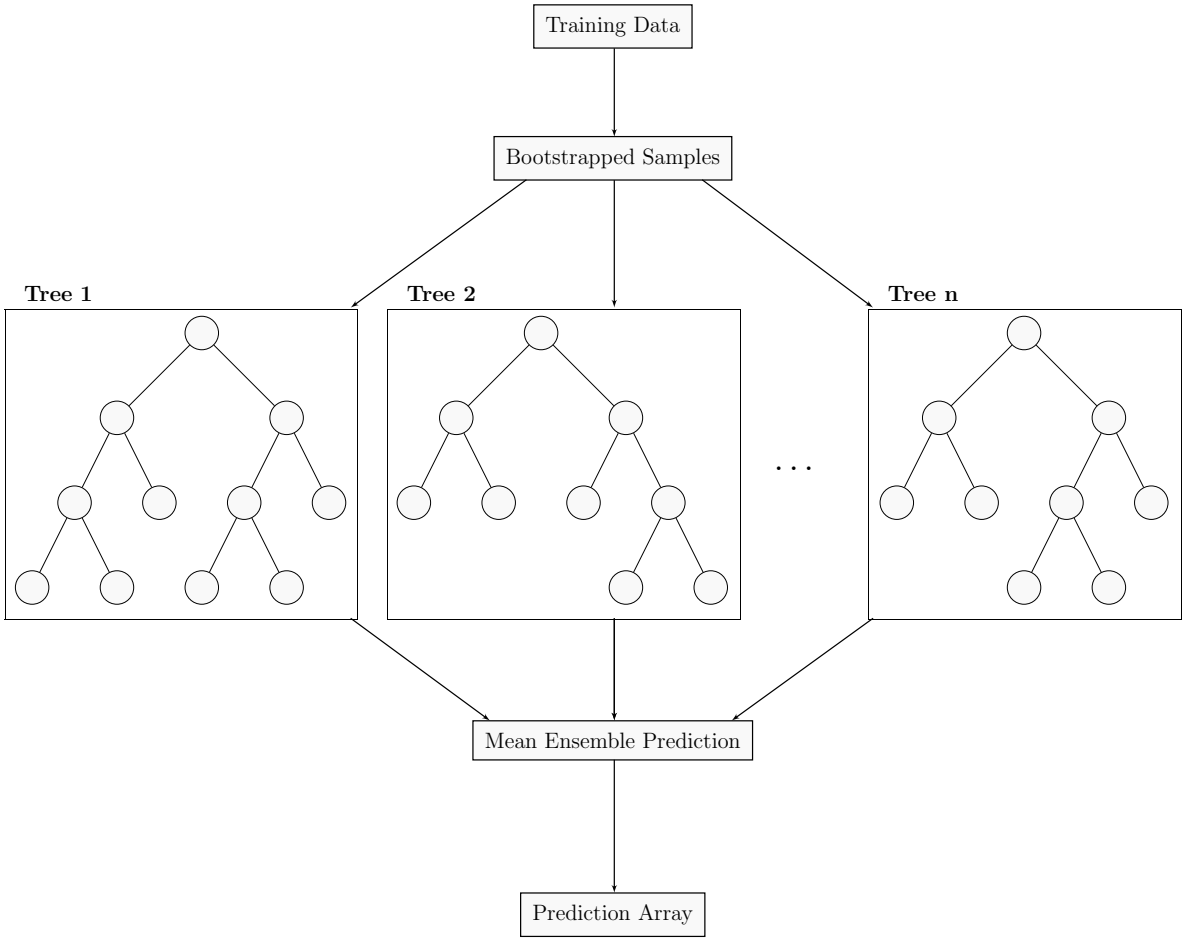


Figure 10: Flow diagram of Random Forest

It can be seen from the diagram above that each Decision Tree in the Random Forrest has a unique structure as a result of the bootstrapped sample data distribution. To increase the variance of each tree, the randomness of the bagging process can be further enhanced by adding a random state hyper-parameter. This hyper-parameter increases the robustness of the resulting algorithm, and improves the regressor's overall generalisability.

In order to ensure the Random Forest is capable of mapping each node in the CFD contour data a variation of different tree sample sizes were tested in conjunction with the appropriate regularisation parameters (*see Appendix III*). From the results of these tests; it was found that a Random Forrest of 150 Decision Trees, each with an applied random state of 42 was best suited to generalise and predict the provided T-piece CFD data range.

Autoencoder

Often data that is representative of real-world problems tend to have high associated dimensionalities (such as the data that defines fluid behaviour). One statistical issue that arises from this high dimensionality is the increase in the combination of distinct variables that are required to map the given solution, this phenomena is known as the curse of dimensionality [5]. In machine learning the mapping of such specific variables can often lead to overfitting, as the true trends and data correlations are not learned. However these data trends can often be represented in a lower dimensional impression, typically referred to as an intrinsic dimensionality space [30]. Rather than developing an algorithm to learn the true data, it can sometimes be more beneficial to train on the representative lower dimensional manifold space. There are several techniques that can be used in order to reduce high order data, common methods include PCA (Principal Component Analysis), Kernel PCA, LLE (Locally Linear Embedding), and Autoencoders [6].

Autoencoders are a form of unsupervised machine learning algorithm, moreover they are a specialised type of Neural Network [30]. These networks are specifically designed to learn representations of the input data. This makes Autoencoders particularly desirable for applications such as denoising, data reconstruction and generation, and non-linear dimensionality reduction [31]. As with many real-world problems, fluid mechanics typically follow inherently non-linear data trends. This makes the non-linear dimensionality reduction of Autoencoders a desirable method to solving these fluid related problems.

Like with simple feedforward Neural Networks, Autoencoders possess the same neurone and inter-neurone structures, as well as incorporating the same activation functions and backpropagation learning techniques. One defining characteristic of an Autoencoder, that differs from conventional Neural Networks, is the symmetrically stacked layer structure these networks possess [6]. Often Autoencoders map the instance data to a lower dimensional manifold referred to as an encoding space. This encoding space is then reconstructed in order to reproduce the input data at the output of the network. Conventionally the compression and reconstruction halves of the network are referred to as the encoding and decoding sections respectively [6]. These two network components are useful, as they can be used to generate and accurately decode the compressed data representation of the encoding space. The network diagram overleaf illustrates the fundamental structure of a standard Autoencoder:

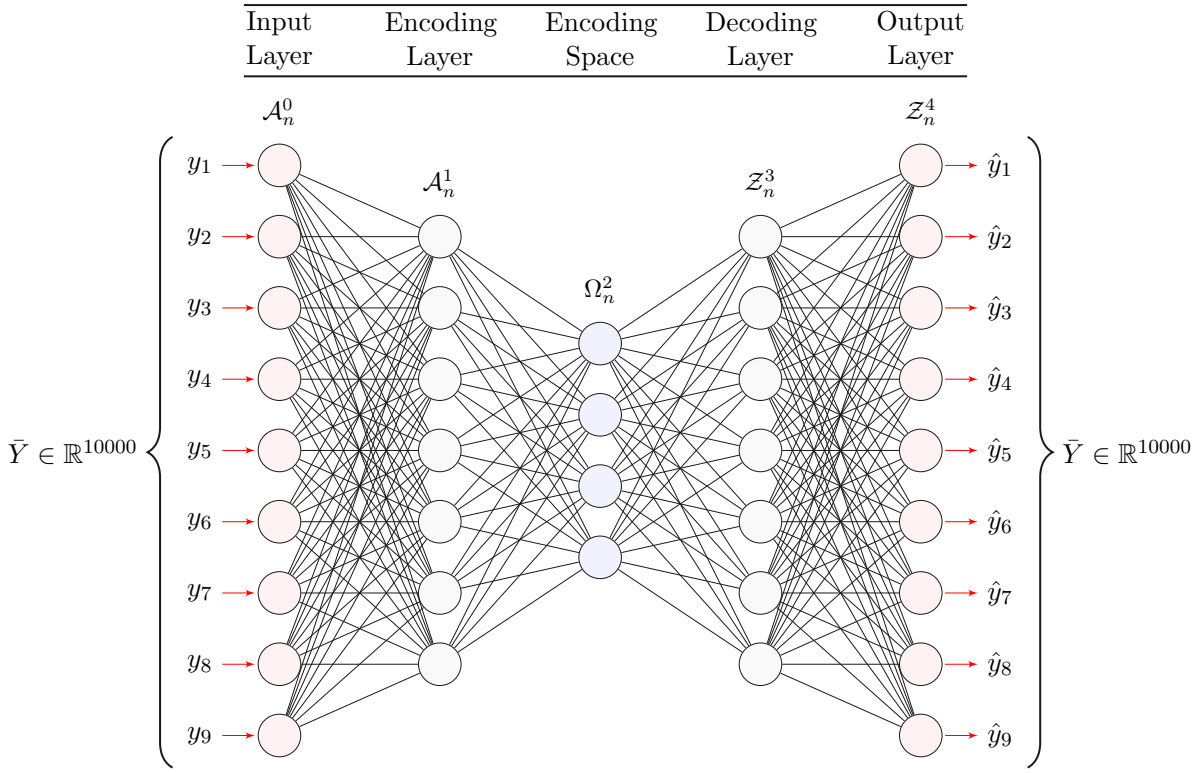


Figure 11: Standard Model of an Autoencoder

From figure 10 above, the symmetrical layered structure of the Autoencoder can clearly be noted. It can also be seen how the high dimensionality instance data (in this case the CFD contour data) is compressed by the encoder section, and then efficiently reconstructed by the decoder. In order to use the Autoencoder for efficient non-linear dimensionality reduction, the resulting intrinsic space at the centre of the network is to be used as a manifold representation of the output data. This lower dimensional impression will inevitably result in a more generalised ML model.

As mentioned before, because Autoencoders are specialised forms of Neural Networks, the same network features and characteristics are applicable. Likewise the same learning methods, such as the use of MSE[15] (to determine the prediction accuracy) as well as the Adam optimiser, can be used to train these Autoencoders.

Although Autoencoders are excellent methods for high fidelity reconstruction, these networks can be susceptible to overfitting (especially very deep encoder structures that possess high degrees of freedom). If an Autoencoder overfits the provided data, the integrity of the non-linear dimensionality reduction, as well as the reconstruction ability of the network may be compromised. Like with other ML algorithms, this can affect the model's overall performance. on unseen data instances. To overcome the overfitting issues associated with Autoencoders similar techniques to that of Neural Networks can be applied. Some such methods include reducing the overall size of the network (either by minimising the number of neurones or layers in the network), and by modifying existing hyper-parameters (like the optimiser's learning rate and momentum).

[15]The evaluation method used to determine the prediction error gradient in an Autoencoder is commonly referred to as the loss function. This is because autoencoders are mainly used as a form of data reconstruction, and the loss term refers to the overall reconstruction loss of the network [6].

One of the most common and robust methods of regularisation is the application of Dropout [6]. Dropout is a statistical regularisation method that can be applied to various ML models, the most popular of which are Artificial Neural Networks [5]. This method for model simplification acts as a form of bagging that effectively produces a very compact network ensemble [5]. During the Dropout process a probabilistic value, known as the Dropout rate, is assigned to the desired layers of the network[16]. This Dropout rate dictates the probability that each neurone in the layer is to be deactivated/dropped-out (hence the name Dropout) [6]. This neurone elimination is re-established for each iteration of the training process. An advantage to randomly dropping out neurones in the network is that the adjacent neurones and layers do not build dependencies on specific neurones [6]. This inevitably reduces the sparsity of the Dropout layers, as the network learns to utilise more neurones in the structure. Effectively acting as an ensemble of differing Neural Networks.

Another popular method for Neural Network regularisation is Batch Normalisation. Similar to the prescaling of the input data, Batch Normalisation works by applying data normalisation principals to each layer of the network [6]. This regularisation method works by sampling the neurone values for each network instance into batches of data. From this data, the mean and standard deviation of each neurone batch is computed. As with the normalisation that occurs during data preprocessing, these batched samples are transformed using the process of Gaussian normalisation/standardisation [6]. This process involves scaling and shifting the normal distribution of the sample (formed by the mean and standard deviation), such that the Gaussian curve is centred at zero with a unit standard deviation. A beneficial characteristic of this normalisation method is that highly variant outlining data points do not affect the overall distribution of the normalised data (unlike MinMax normalisation which does not scale well on outlying data) [6]. The equations that describe the Batch Normalisation process at each neurone can be represented in the equation array below:

$$\text{Batch Normalisation} \quad = \quad \begin{cases} \mu_B & = & \frac{1}{m_B} \sum_{i=1}^{m_B} x \\ {\sigma_B}^2 & = & \frac{1}{m_B} \sum_{i=1}^{m_B} (x - \mu_B)^2 \\ \hat{x} & = & \frac{x - \mu_B}{\sqrt{{\sigma_B}^2 + \epsilon}} \\ \hat{z} & = & \gamma\, \hat{x} + \beta \end{cases} \tag{14}$$

Where:
μ_B = Empirical batch mean
σ_B = Empirical batch standard deviation
$\hat{x}$ = Standardised input
$\hat{z}$ = Rectified input
γ = Layer scaling parameter
β = Layer shifting parameter

It can be seen from the equations above that the standardised result is then rescaled and shifted according to the layer parameters. These parameters act to modify the normalised batched samples so that they can be computed in the consequential layers.

[16] By varying the magnitude of the Dropout rate the extent of the model regularisation can be varied. Although, typically a Dropout rate of 50 % is preferred [6].

The advantage to using Batch Normalisation in Neural Networks, is that the algorithm can converge on a solution with very little training samples, as this regularisation method is equivalent to pre-scaling the input data at each specified layer of the network [6]. Resultantly the input variables to such batched networks do not need to be pre-scaled, as the required data scaling is performed within the network. This input scaling also gives Batch Normalisation its regularisation characteristics [6]. Furthermore, Batch Normalisation also allows for the gradients of the cost/loss function to be computed more efficiently; due to the normalised properties of the training instances. This efficiency makes training batched networks faster, although the added computation of the formulae described in equation (14) can extend the overall run time of the evaluated network (potentially compromising the real-time capabilities of the network) [6]. Given these advantages, Batch Normalisation can be a very effective method for the regularisation of Neural Networks and subsequent Autoencoders.

To utilise Autoencoders for dimensionality reduction, the reduced encoding space is to be used as the intrinsic output for an additional manifold ML algorithm. In order to be applied in the case of the CFD problem, the encoding space produced by the Autoencoder is to be used to train a standard Neural Network as depicted in the flow diagram below:

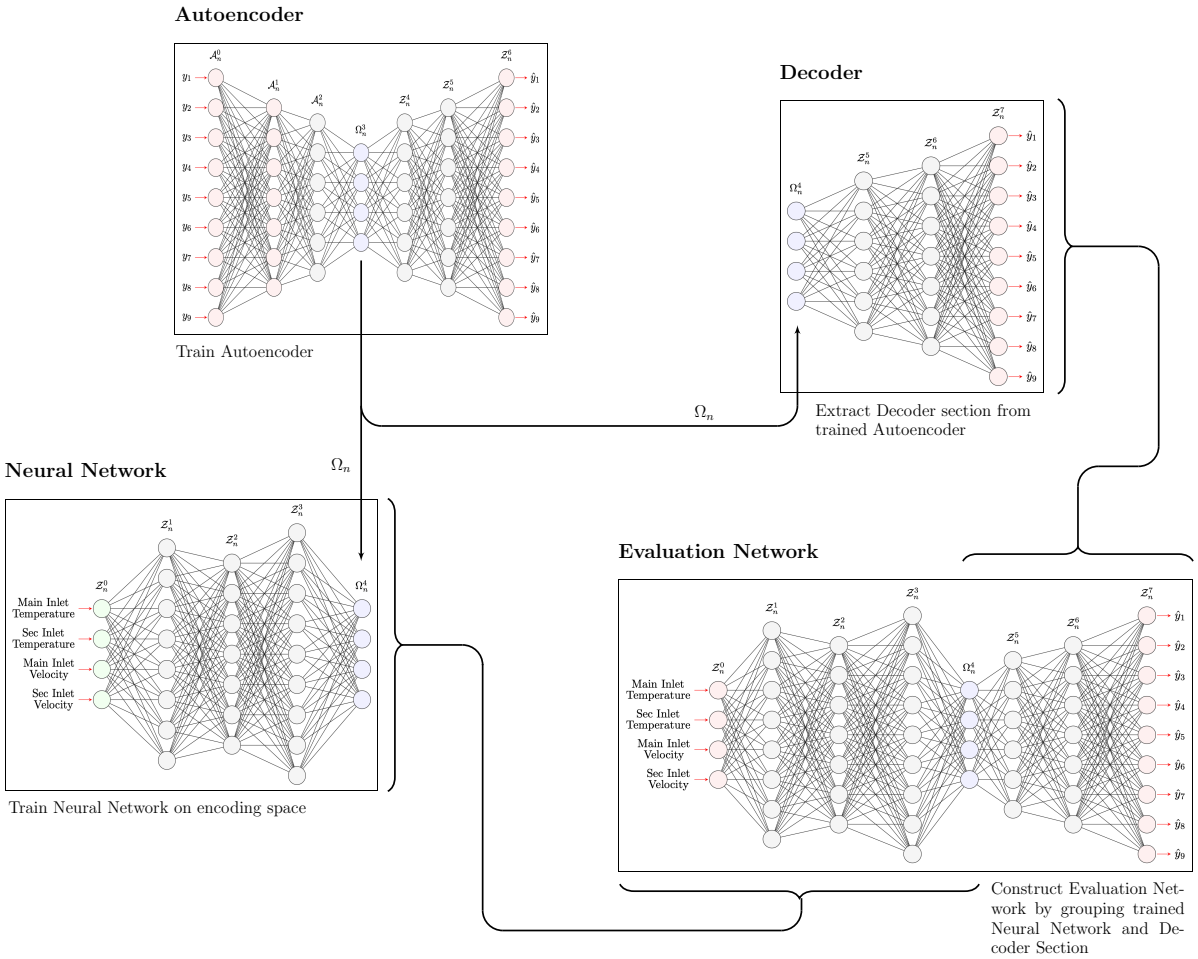


Figure 12: Flow diagram for Autoencoder training process

From figure 12 above, it can be seen that the resulting prediction of the Neural Network is to be reconstructed back to the original CFD representation by the decoder section of the Autoencoder.

Following the processes and stages highlighted in the figure above, the individually optimised Autoencoder and Neural Network structures could be selected (*see Appendix III*). From the results of this hyper-parameter tuning process it was found that the best model structure for predicting the CFD data comprised of an Autoencoder with a characteristic 32 dimensional encoding space, and a Neural Network with three hidden layers comprising of 512 neurones each. Thus the selected trained Neural Network, Random Forest, and Autoencoder model structures (specified above) are to be further compared and measured on their individual prediction performance given the T-piece CFD data.

Results and Discussion

From the algorithm hyper parameter search (*see Appendix III*), the following optimised model architectures were selected based on their generalisability and prediction accuracy:

- A three layered Neural Network with each hidden layer comprising of 128 neurones.
- A 150 tree, CART function, Random Forest with an in initial random state of 42.
- A deep Autoencoder network comprising of a three layered, 512 neurone per layer Neural Network and a deep decoder section with a batch-normalised encoding dimension if 32, and a dropout rate of 0.5.

In order to properly evaluate the capabilities of the individual algorithms; various performance factors were analysed such as the prediction error, model runtime, static and flash memory usage, and the ability to generalise on unseen outlying data. The metrics of which could be used as a means of comparison between the selected algorithms. To ensure the performance comparison was unbiased and controlled, all three models were tested using the same CPU and device. Thus each test and algorithm evaluation was performed using a quad-core Intel i5 processor.

Prediction Error and Comparison

To compare each model's ability to predict the turbulent fluid flow within the mixing T-piece, each of the 1000 recorded CFD data instances were supplied to the trained models. The cumulative average of these predictions was calculated in order to determine the mean prediction result. This average prediction was then compared with the true average contour data, generated by the CFD simulation, using the mean absolute error as a proportional comparison between the observed and actual contour results. Below is a table displaying these recorded mean and maximum error deviations for the temperature and velocity predictions made by each ML algorithm:

Table 3: Mean absolute errors for each algorithm prediction

Algorithm	**Temperature Error (%)**		**Velocity Error (%)**	
	Avg.	**Max.**	**Avg.**	**Max.**
Neural Network	3.333	3.835	3.764	68.776
Random Forest	2.206	2.787	3.626	42.967
Autoencoder	5.503	31.881	11.398	93.779

From analysis of the table above, it can be noted that the average prediction error was generally higher for the velocity contours than for the temperature data. Most-likely this predictability is a result of the random turbulent characteristics of the fluid, which are much harder to generalise and predict. This is further reflected in the maximum recorded errors, as the values of the maximum velocity error are significantly larger than that of the temperature measurements. Furthermore, from table 3 it is evident that the Random Forest produced the lowest average and maximum prediction error when compared to the other ML models.

In order to analyse the efficiency of the predicted contour, the resulting averaged model predictions are to be compared to the average flow contour generated by the CFD analysis (depicted in figure 5). By graphing the average data for each CFD cell prediction using Matplotlib [32], the quality and accuracy of the predicted CFD contour can be evaluated. This prediction contour can further be accompanied by the error contour, which can be used indicate the regions were the model prediction was unsuccessful. This graph can be produced by calculating the MAE for each cell in both the CFD data, and prediction set.

Thus, figure 13 below shows the mean temperature prediction and error contours for the selected Neural Network given the 1000 T-piece CFD data instances:

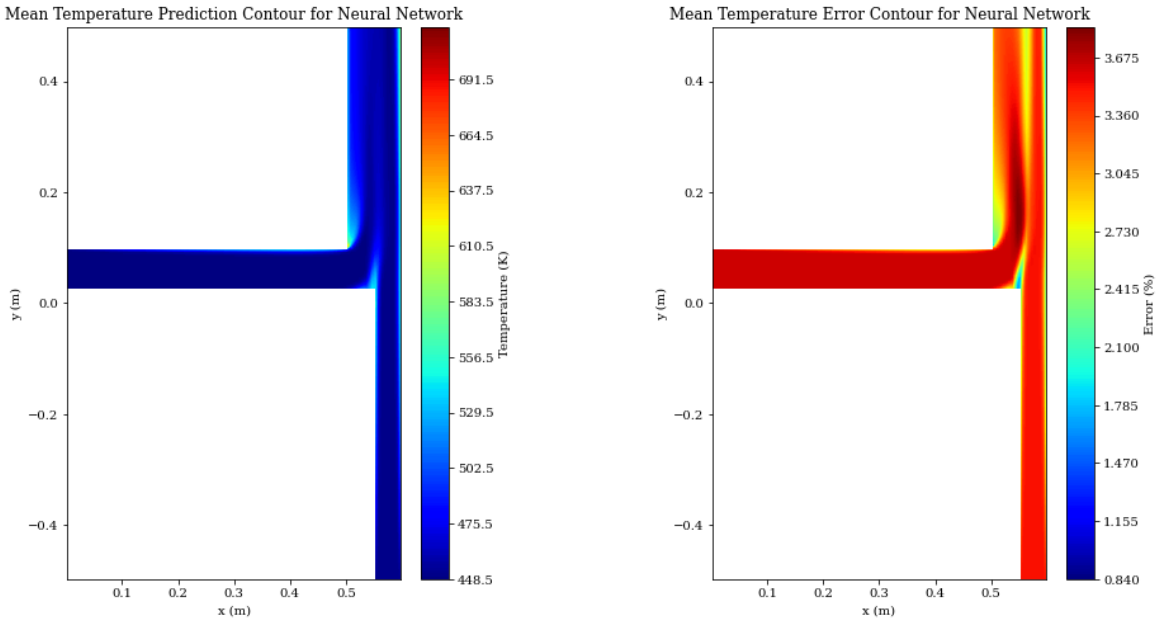


Figure 13: Mean temperature prediction and error contour for Neural Network

On inspection of the above contour graphs, it can be seen that the largest prediction error occurred at the mixing interface of the main and secondary inlet streams. The formation of this relatively high region of error is due to the large temperature gradients, thermal energy exchanges, and diffusive turbulent flow at this contact site. This is because the Neural Network cannot accurately learn/correlate the flow patterns for this specific portion of the fully developed fluid body. Based on the visual analysis of the above contour plot, it can be clearly seen that the temperature contour generated by the Neural Network is sufficient to be used as a relatively accurate indicator of the general thermal state within the T-piece.

Similarly to the temperature contour display, the velocity and error flow contour was developed from the Neural Network's prediction on the T-piece data. This Velocity graph can be seen overleaf:

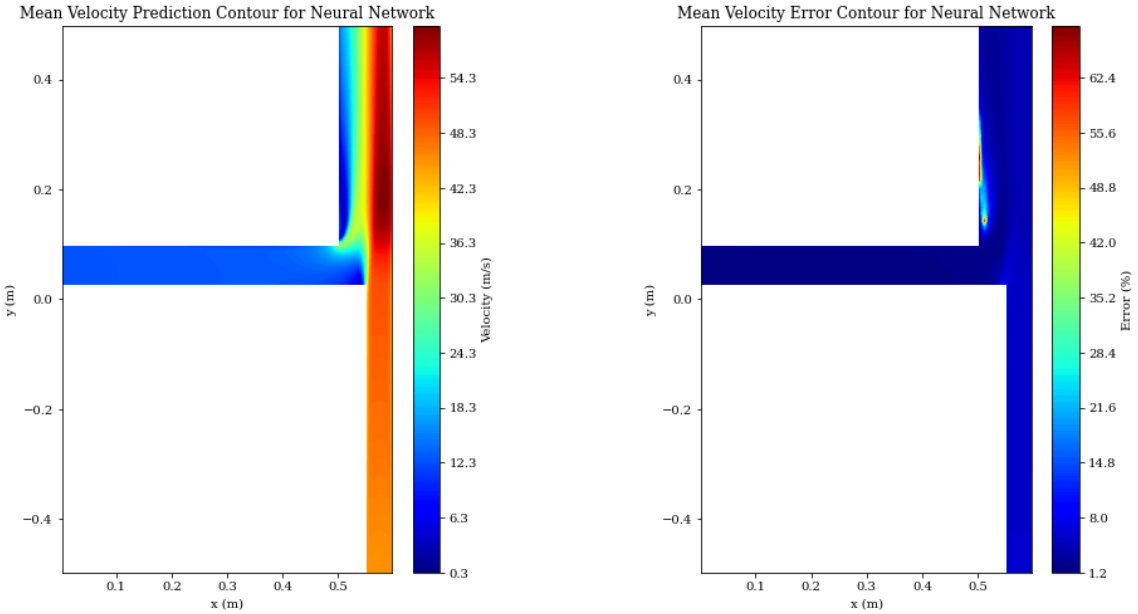


Figure 14: Mean velocity prediction and error contour for Neural Network

It is clear from the above flow diagram and accompanying error plot, that the magnitude and variation of the velocity prediction error is relatively low across the entirety of the T-piece surface. Except for a small portion of the flow regime indicated near the outlet pipe wall on the error contour. The identified region of high error is indicative of the high turbulence and flow separation that occurs at this location. This observation can be further supported by comparing and overlaying the turbulence fields recorded from the CFD analysis (*see Appendix IV*), as the regions of high turbulence align with the large errors depicted in the plot above.

The figure below shows the respective temperature prediction, and error plot produced by the selected Random Forest Regressor:

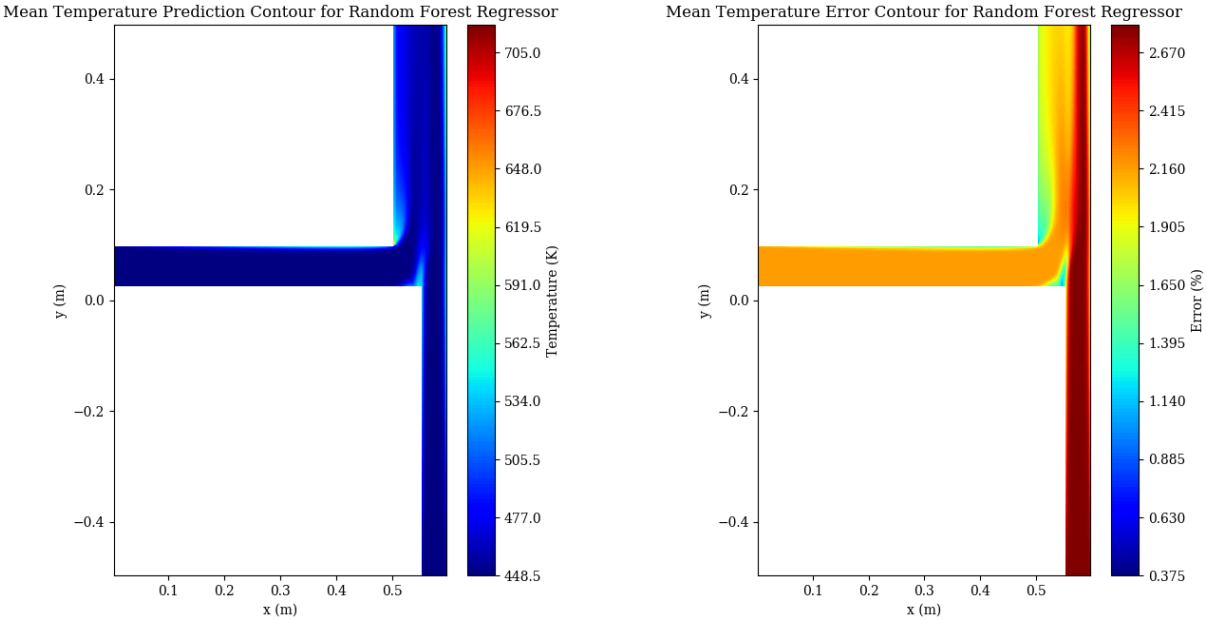


Figure 15: Mean temperature prediction and error contour for Random Forest

In contrast to the temperature prediction contours produced by the NN, it is apparent that the Random Forest's prediction error at the thermal interface of the two streams is significantly lower. Furthermore, it is clear that the overall quality of the prediction contour (for the predictions of data that falls within the range of the training set) is greater than that of the Neural Network. This is evident in the above figure, as the prediction contour shows significantly less error within the turbulent regions of the T-piece, as well as a lower overall error range (depicted by the contour bar/legend adjacent to the contour plot).

Repeating the same data visualisation procedures as above, the Random Forest's velocity prediction contours could be developed and displayed as follows:

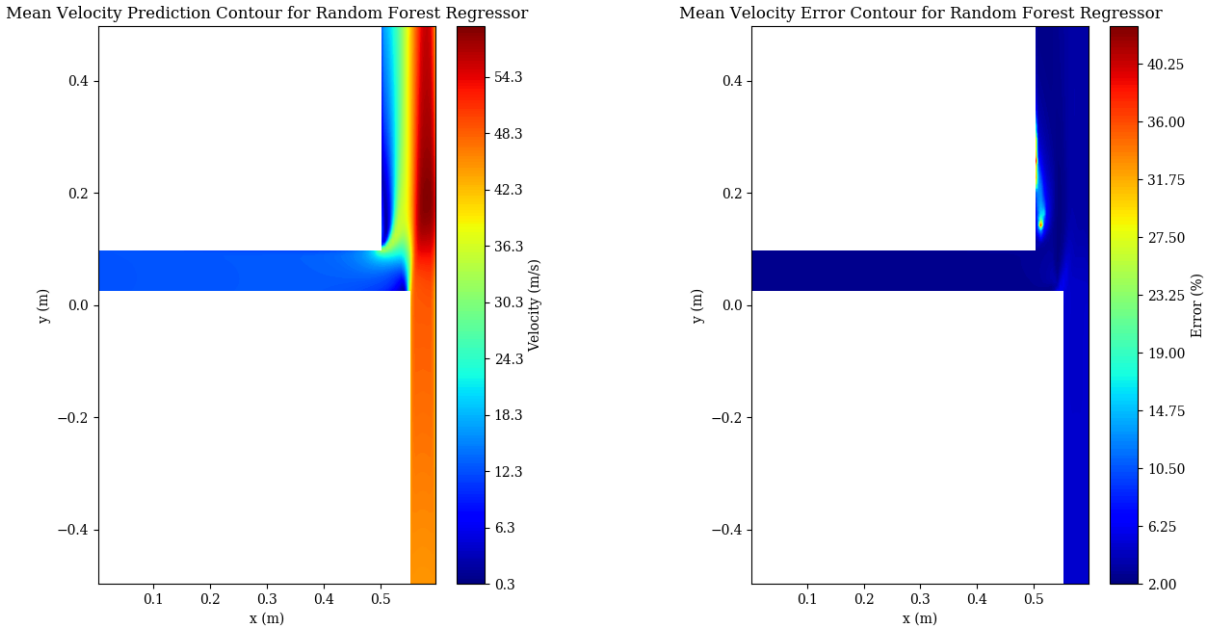


Figure 16: Mean velocity prediction and error contour for Random Forest

As with the velocity prediction contour of the Neural Network, the Random Forest's reconstruction of the CFD velocity profile seems to be relatively consistent with that of the documented turbulence characteristics of the mixing T-piece (*see Appendix IV*). Although the Random Forest produced the most precise and accurate data predictions, it must be noted that the minimum velocity error of the Random Forest's contour prediction is larger than the Neural Network's velocity reconstruction. This higher minimum velocity error seems to indicate that the Random Forest is not as capable of generalising on specific flow patterns as the Neural Network (i.e. the Neural Network produced more accurate predictions for some portions of the CFD system). From the Random Forest's temperature and velocity predictions, it can be seen that the error of each contour graph is less than that of the Neural Network. These observations coincide with the recorded values displayed in table 3.

From the Autoencoder's reconstruction of the CFD temperature data, the following results were graphed over-page:

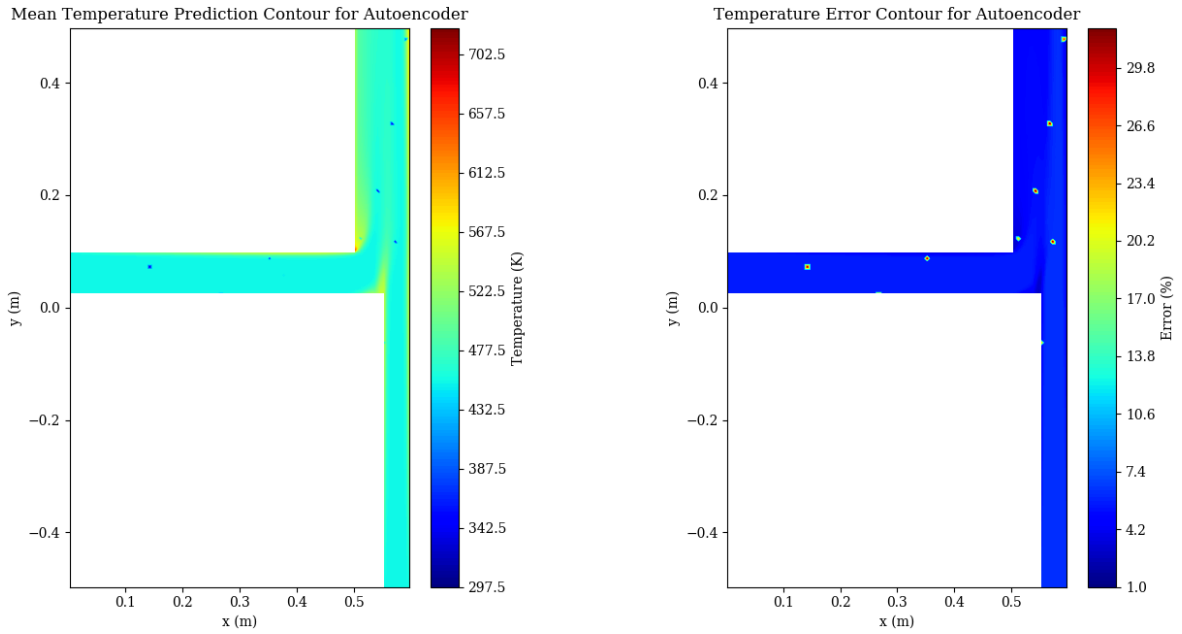


Figure 17: Mean temperature prediction and error contour for Autoencoder

From the diagram above it can be seen that the thermal contour plot was not consistent, as portions of the CFD map showed signs of relatively high prediction error. These depressions in the contour are most likely a result of the typical reconstruction losses associated with Autoencoders. This is because any errors that may originate at the output of the 32 dimensional NN are carried through during the decoding computations of the Autoencoder. As a result this ML model may be very sensitive to changes in particular regions of the CFD data.

Similarly to the observations specified above, the same features can be identified in the Velocity prediction contour below:

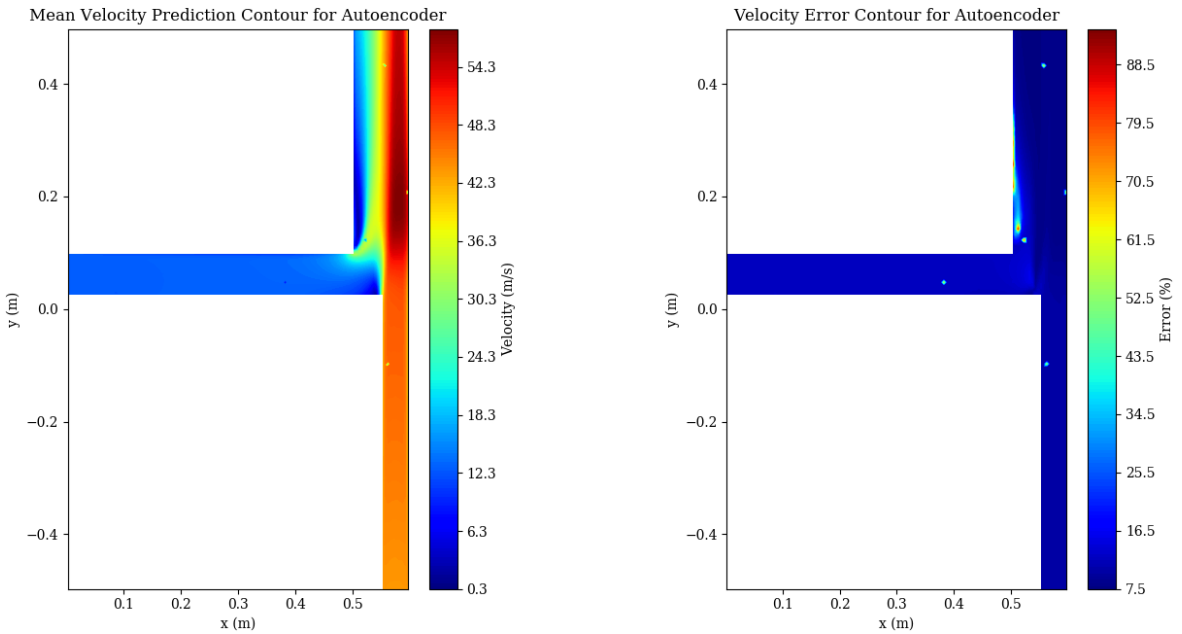


Figure 18: Mean velocity prediction and error contour for Autoencoder

Likewise these regions of high error produce very inconsistent and unreliable visual contour data. Due to the high variance of these localised depressions, the prediction errors across the rest of the CFD contour is obscured. However, by restricting the scaling of the colour gradient (*see Appendix IV*), it can be seen that the Autoencoder produced relatively consistent prediction results across the rest of the contour. Although, it must be noted that the error of the temperature and velocity predictions were still greater than that of the other two prediction models (which is representative of the errors recorded in table 3).

Nevertheless, based on the comparison and analysis above, the Random Forest is clearly capable of producing the best contour results for both temperature and velocity (within the data range specified in table 2). These observations tie-in with the recorded mean and maximum data represented in table 3, as well as the error frequency distributions of the model predictions (*see Appendix IV*).

Algorithm Runtime and Memory

Although the prediction error and comparison results seem to indicate that the Random Forest is the ideal ML regression model to use when predicting the CFD results of the two dimensional mixing T-piece; on further analysis of the algorithm memory and runtime performance metrics, it can be seen that the specified algorithm may not be optimal for use in other CFD applications.

In order to determine whether the algorithms are capable of producing prediction results in near real-time, the prediction times for varying input samples were recorded. This was achieved by supplying the algorithm with instance arrays of varying sizes (namely 1, 10, 100, 1000, 2000, and 5000 samples) and recording the time taken to produce the corresponding prediction array. Table 4 below shows the recorded prediction time of each algorithm on the varying sample sizes:

Table 4: Runtime measurements for varying instance sample sizes

Instance Batc Size	**Neural Network Runtime (s)**	**Random Forest Runtime (s)**	**Autoencoder Runtime (s)**
1 sample	0.067	0.343	0.065
10 samples	0.598	2.084	0.577
100 samples	5.879	15.186	5.621
1000 samples	80.135	132.576	80.210
2000 samples	206.487	281.503	211.071
5000 samples	844.163	972.936	831.439

By plotting the forecasted trends of these algorithm run times, the real-time applicability of these algorithms can be evaluated. The graph over-page shows these forecasted runtime efficiency trends, and how they scale with increasing sample sizes:

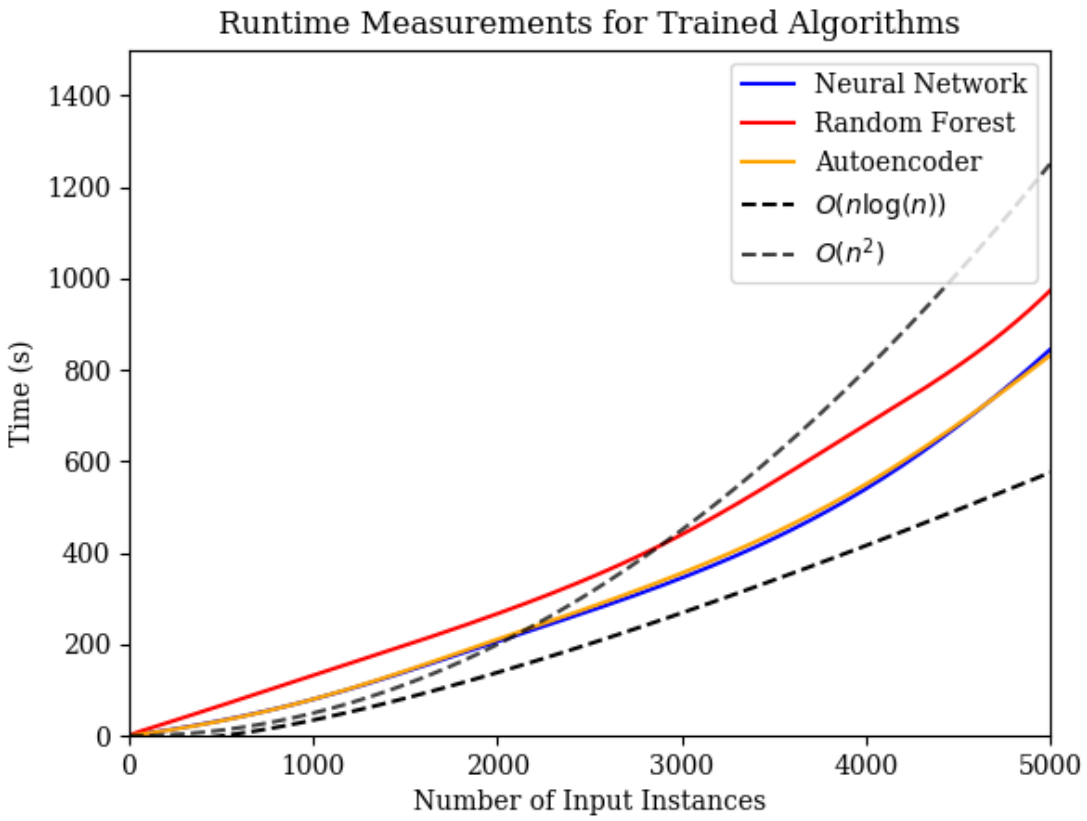


Figure 19: Interpolated algorithm runtime projections

From the trends in figure 20 and table 4 above, it can be seen that the Random Forest has the longest runtime of all the algorithms. Nevertheless, it is evident that all three ML models are capable of being implemented in near real-time. This is evident, as the time delay when computing a single data instance is relatively imperceivable. Moreover, on observation of the scaling forecasts, it can be seen that the runtime trends for all the ML algorithms follow a scaled logarithmic trend as seen by the standard $O(n\log(n))$ efficiency benchmark[17]. This means that the real-time prediction characteristics do not scale efficiently as the instance sample size increases.

In conjunction with the runtime measurements, the memory demands of the algorithms were compared. This includes the static/non-volatile memory (the available flash memory required to store the model structure and parameters), as well as the random access memory (RAM) required to run the program and computations. In order to properly evaluate the RAM usage, the metrics for the resident set size (RSS) memory were recorded. This is because the RSS memory encompasses all the sub-memory RAM processes that occur in main memory. The following table shows the recorded static and volatile memory sizes for each machine learning model:

[17]The big O notation is a standard metric from which the runtime performance of algorithms can be compared. This metric represents the expected scaling trend of the algorithm runtime as the size of the dataset is increased.

Table 5: Static and resident set size memory usage for varying instance sample sizes

Instance Batc Size	**Neural Network Memory (MB)**	**Random Forest Memory (MB)**	**Autoencoder Memory (MB)**
Static	5.230	3.385×10^4	42.757
1 sample	1.872	0.000	1.888
10 samples	3.060	0.000	2.363
100 samples	10.579	0.000	10.748
1000 samples	208.560	760.631	169.935
2000 samples	463.405	1629.327	426.365
5000 samples	1062.728	2873.876	1037.693

It is evident in the table above that the Random Forest has an outstandingly larger static and RAM memory demand than the other models[18]. Although, it is also apparent that the Random Forest does not require any significant RAM when predicting small instance samples. It must be noted that the recorded values for algorithm runtime and memory are not necessarily repeatable to the accuracy represented in the tables. As underlying computer processes may affect the accuracy of successive readings. However, these values are representative of the expected runtime and memory trends associated with these algorithms (i.e. they are consistent within a tolerable range).

The figure below displays the trends of the RSS memory recorded in table 5:

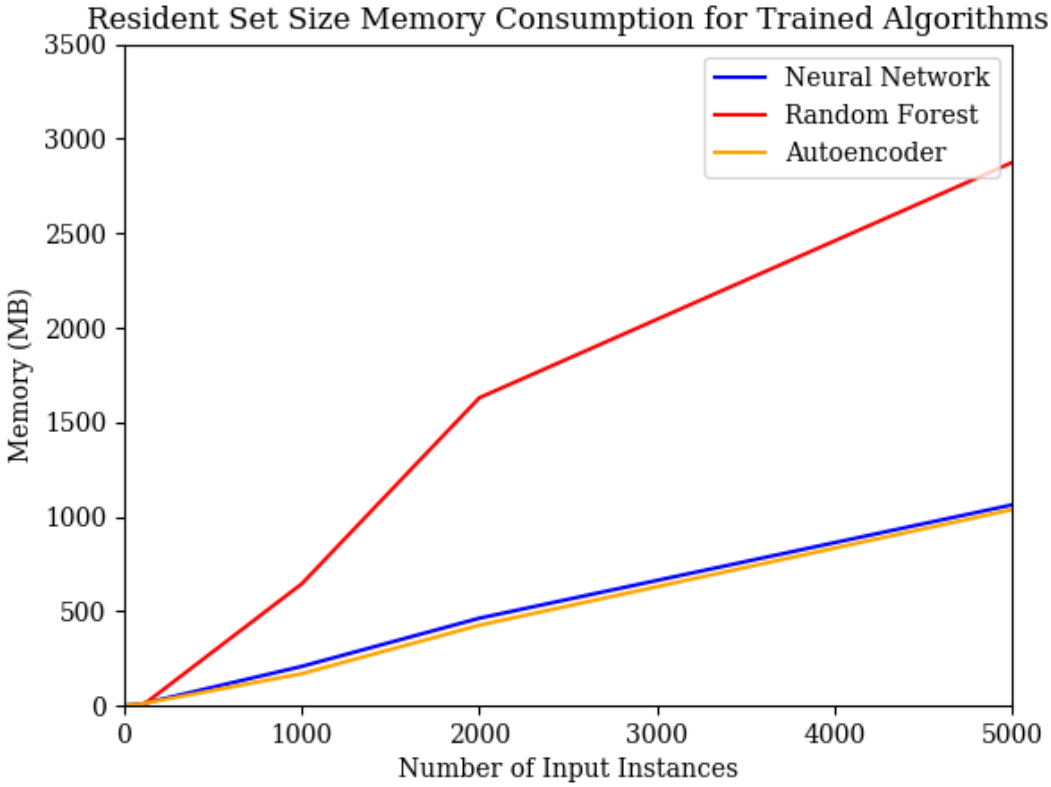


Figure 20: Random access memory usage for varying data sample sizes

[18]These high computational inefficiencies can hinder the Random Forest's ability to be integrated into current engineering systems (*see Appendix V*).

Thus from the runtime and algorithm analysis, it can be seen that all the models are capable of making a single CFD prediction in near real-time. Furthermore, based on the runtime and memory trends it is evident that the Random Forest had the least desirable performance. This observation counterbalances the high prediction accuracy identified in the section above.

Lower-Bound Error and Comparison

Similarly to the prediction error comparison, another method of validating whether an algorithm is effective at predicting the CFD data is by analysing the prediction error associated with data instances outside the original training dataset range. To achieve this a new data batch of 20 CFD samples were solved for using the ANSYS Fluent design of experiments feature (*see Materials and Methods*). This new dataset consisted of data that was sampled below the original training set. The table below shows the temperature and velocity instance ranges used to develop the lower-bound CFD data:

Table 6: Lower-bound inlet boundary ranges used in mixing T-piece

Boundary	**Temperature [K]**	**Velocity [m/s]**
Main Inlet	268 - 298	0 - 5
Secondary Inlet	268 - 298	5 - 20

It can be noted that these sample instances are lower than the original dataset instances depicted in table 2 (*see Materials and Methods*). From these 20 design points, the following lower-bound mean temperature and velocity CFD results could be produced:

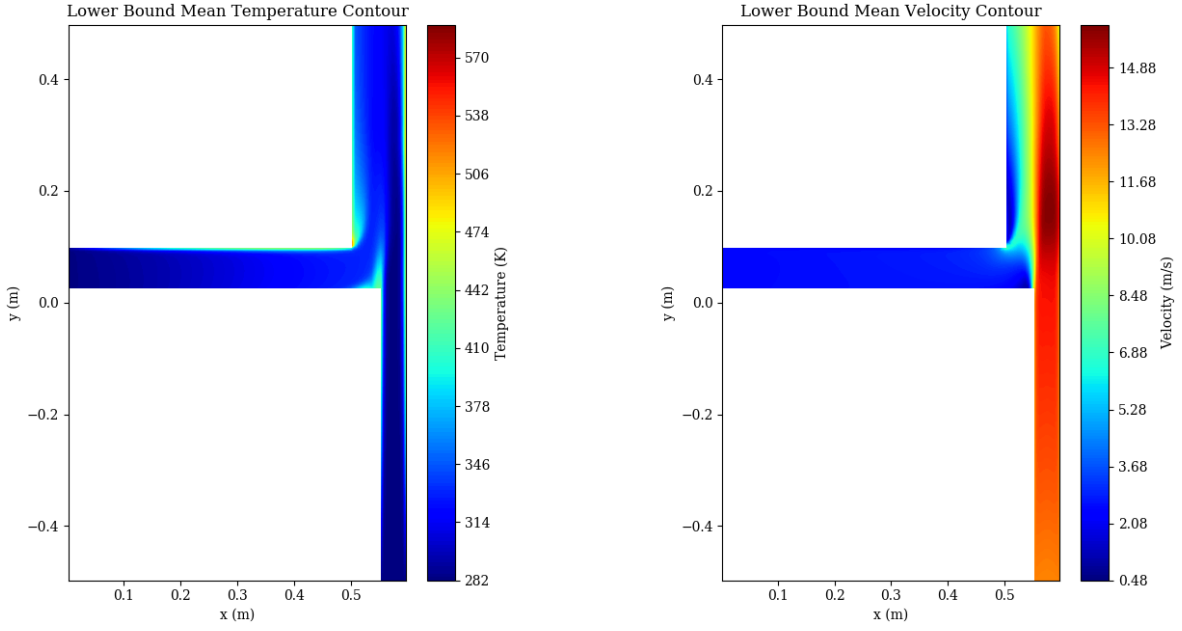


Figure 21: Mean temperature and velocity contours for lower-bound data range

On observation of these contour graphs along with the recorded turbulent Reynolds numbers (*see Appendix IV*) it can be noted that this data range possesses a characteristically more fully developed laminar flow regime as opposed to the predominantly turbulent characteristics of the original training data range.

By running the ML models on the instances listed in table 6, the following lower-bound prediction error results were obtained:

Table 7: Mean absolute prediction errors for lower-bound CFD data samples

Algorithm	**Temperature Error (%)**		**Velocity Error (%)**	
	Avg.	**Max.**	**Avg.**	**Max.**
Neural Network	11.306	27.525	10.951	488.712
Random Forest	11.768	29.434	19.773	511.889
Autoencoder	11.102	28.299	34.334	545.774

These recorded error results show vastly different orders to that of the training set error in table 3. This is evident as the Random Forest performed significantly worse on both the temperature and velocity data predictions, when compared to the other models.

By visualising the mean prediction contours for the lower-bound data instances, the error represented in the table above can be better evaluated. The contour below shows the Neural Network's temperature predictions for this lower-bound dataset:

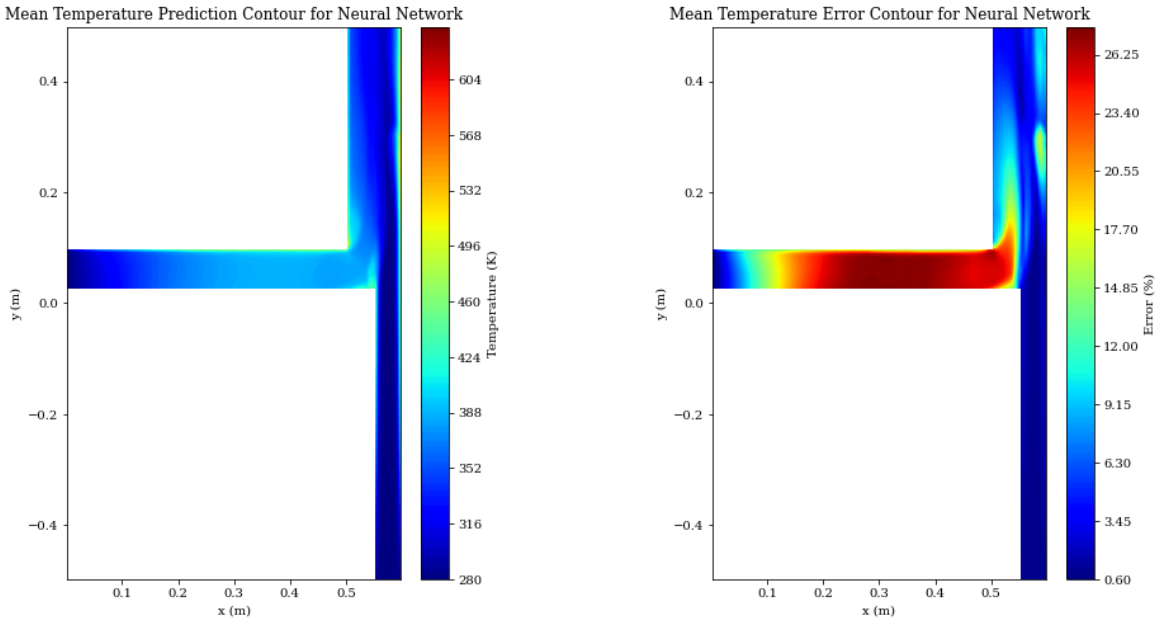


Figure 22: Lower-bound temperature prediction and error contour for Neural Network

It can be seen in the image above that the temperature prediction uncertainty in the main inlet is extremely high and inconsistent. This differs from the observations recorded in figure 13, which showed very low uniform error distribution throughout the majority of the pipe section. Likewise these significantly high errors can also be noted in the corresponding Neural Network velocity prediction contour overleaf:

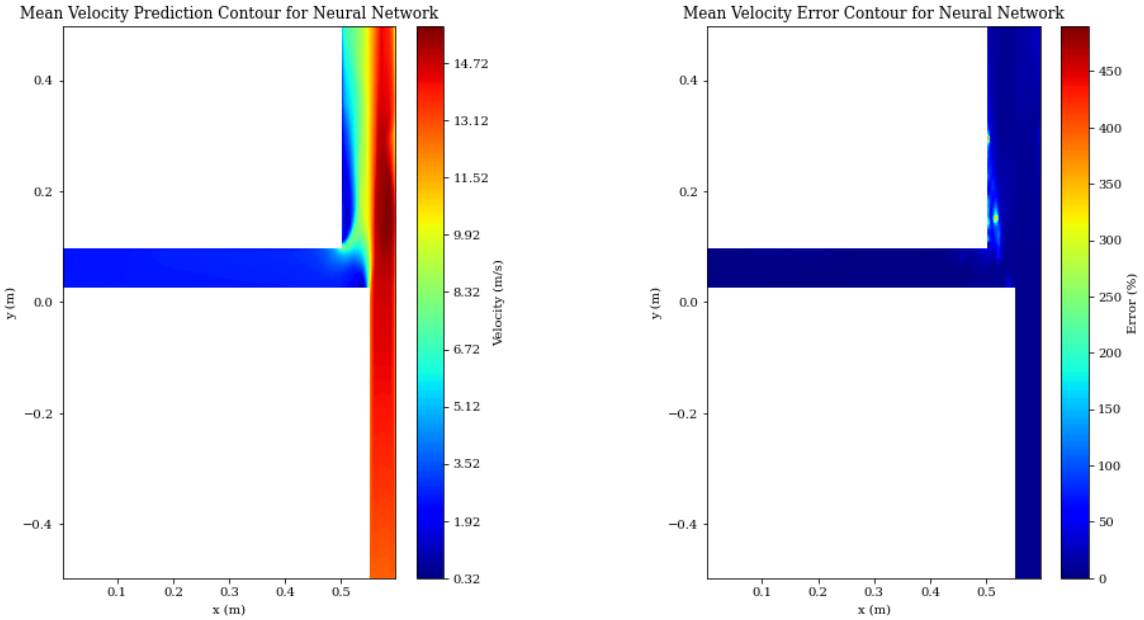


Figure 23: Lower-bound velocity prediction and error contour for Neural Network

Although the velocity profile and error is relatively consistent with that of the training and validation dataset predictions, it can be seen that the general error is significantly higher. This is unsurprising/expected, as the ML model struggles to generalise on the more laminar regime of the lower-bound dataset. Nevertheless, granted the application purpose is not dependent on accurate results, these outer-bound prediction contours can still produce reasonably correlated visual representations.

Repeating the graphing and analysis process for the Random Forest's temperature prediction produced the following results:

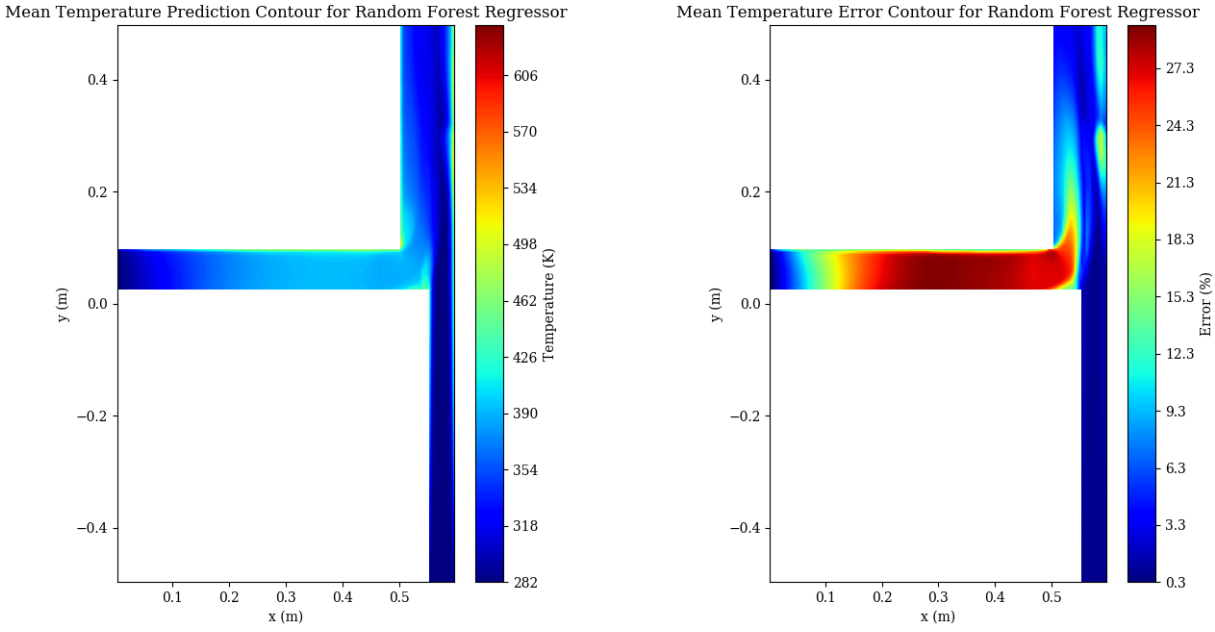


Figure 24: Lower-bound temperature prediction and error contour for Random Forest

Comparing the above temperature reconstruction with that of the Neural Network's lower-bound prediction, it is evident that the resulting regions of high error are almost identical. Furthermore it can be noted that the general error of the contour plot is greater than the Neural Network's reconstruction. This aligns with the overall errors represented in table 7 (which depicts the Random Forests having a higher overall error than both the NN and Autoencoder).

This less accurate contour map can be further illustrated in the velocity prediction graphs of the Random Forest:

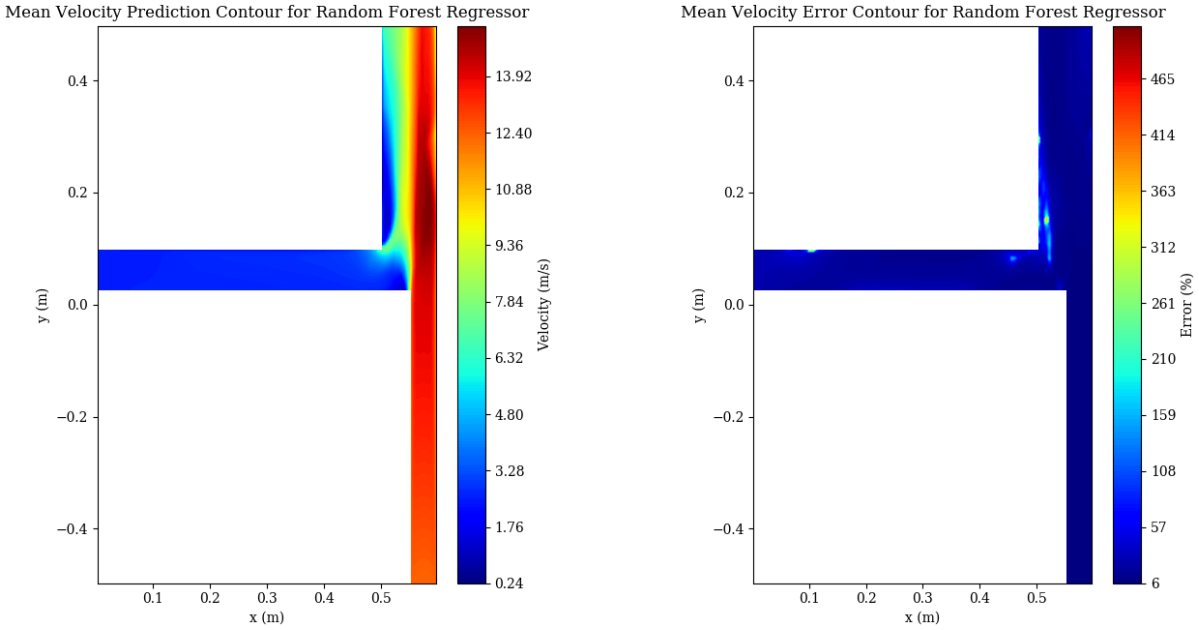


Figure 25: Lower-bound velocity prediction and error contour for Random Forest

It is clear that the Random Forest is unable to reconstruct outlier data as well as the Neural Network. This is evident, as several loci of high uncertainty can be identified on the above contour graph. These results contradict the observations made on the original dataset performance analysis. As the Random Forest produced the best contours, as well as the lowest error. This observation implies that the Random Forest is not suited for all contour reconstruction applications, as it fails to generalise on the lower-bound data as well as Neural Networks. Most-likely this behaviour is a result of the Random Forest's internal structure. This is because the Decision Trees that constitute the ensemble are relatively inflexible. Although the grouping of these trees produce more flexible predictions, the data correlation difference between turbulent and laminar flow restricts/prevents these trees from accurately matching the highly variant data trends. This characteristic overfitting explains why the Random Forest produces contours with higher minimum errors than that of the Neural Network (as this algorithm is not capable of generalising on the diverse flow patterns as well as the NN model).

The following graph illustrates the temperature prediction and associated error produced by the Autoencoder on the lower-bound instance data:

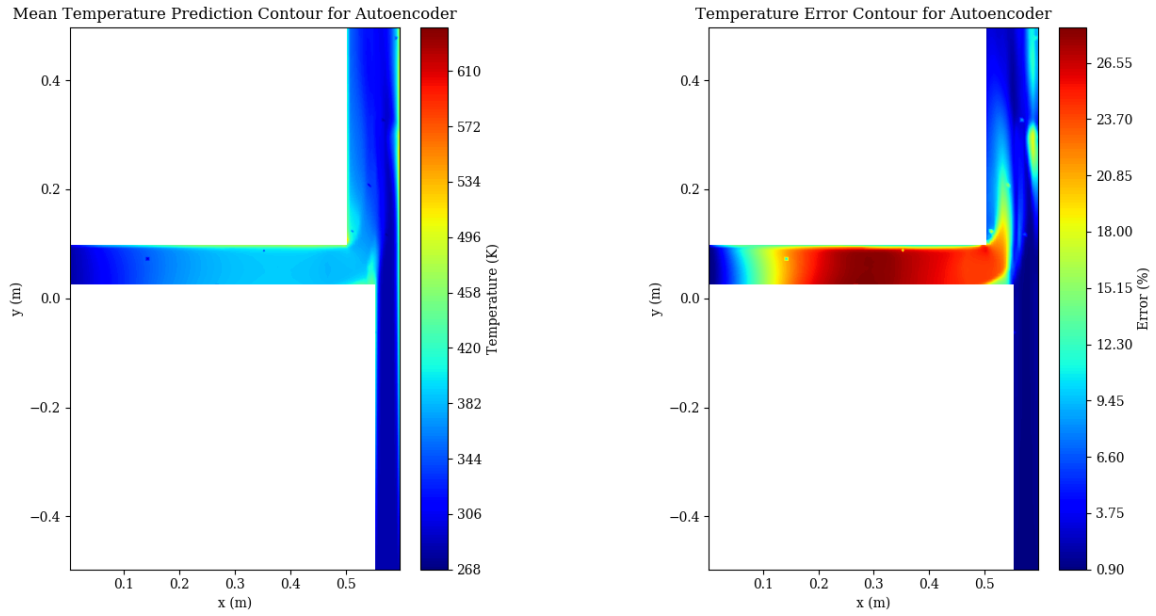


Figure 26: Lower-bound temperature prediction and error contour for Autoencoder

Similarly to the other two models, the temperature contour produced by the Autoencoder shows the same regions of high uncertainty and inconsistency. However, it is notable from the mean temperature contour plot that the Autoencoder produced a more accurate temperature reconstructions than the Random Forest, albeit the contour showed less consistency due to the visible impressions of high error.

Repeating the graphing and analysis for the Autoencoder velocity contour produced the following results:

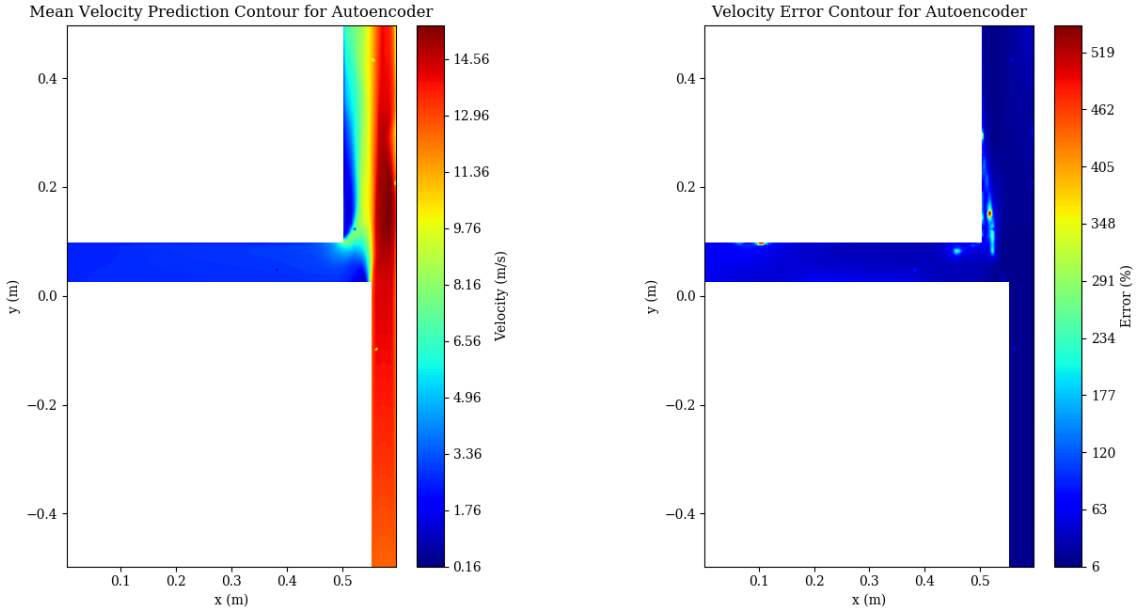


Figure 27: Lower-bound velocity prediction and error contour for Autoencoder

On inspection of both Autoencoder plots above it can be seen that the same regions of high error identified in the original training data range analysis are present in both velocity and temperature contours. As mentioned before, these inconsistencies are a result of the Autoencoder's design characteristics.

From evaluation of the lower-bound model prediction performance, it can be noted that although Random Forests are best suited for training data contour generation, they are not particularly apt for predicting the lower-bound outlier data. Furthermore, it is also apparent that the prediction flexibility of Neural Networks and Autoencoders better suit these algorithms for outlier data predictions. Although due to local flow reconstruction inconsistencies, Autoencoders are rather better suited for high precision data capture applications as opposed to high accuracy tasks which the CFD contour generation demands.

Upper-Bound Error and Comparison

Similarly to the lower-bound analysis, the algorithm performance on the upper-bound outlying data was also evaluated. The CFD instance ranges selected for this experiment are displayed in the table below:

Table 8: Upper-bound inlet boundary ranges used in mixing T-piece

Boundary	**Temperature [K]**	**Velocity [m/s]**
Main Inlet	600 - 630	20 - 25
Secondary Inlet	600 - 630	70 - 85

As with the lower-bound data 20 instances within the above data ranges were computed and solved for in ANSYS Fluent [9]. From the CFD analysis, the following mean contour data was recorded:

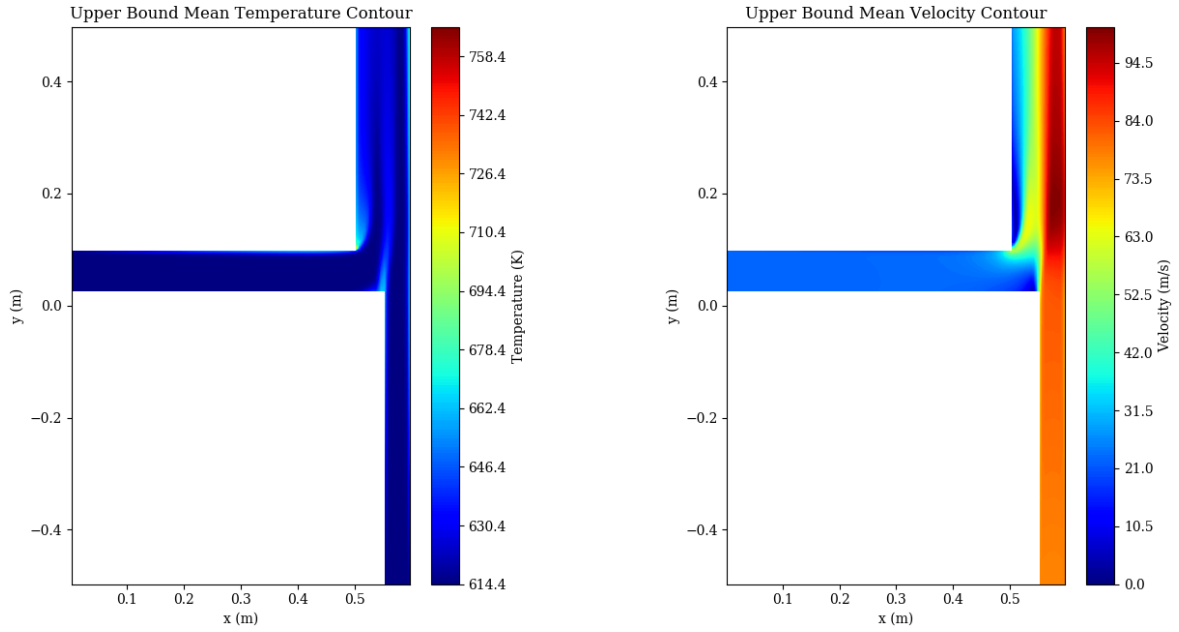


Figure 28: Mean temperature and velocity contours for upper-bound data range

It must be noted from the above diagram, that the larger temperatures and velocities imply that the observed mixing behaviour of each design point was almost purely turbulent (unlike the lower-bound samples). This means that the ML models are better suited to predicting the CFD trends, as these turbulent characteristics better fit the original training set. This is evident when evaluating the MAE for each predicted cell point of the dataset. These recorded errors can be seen in the table below:

Table 9: Mean absolute prediction errors for upper-bound CFD data samples

Algorithm	**Temperature Error (%)**		**Velocity Error (%)**	
	Avg.	**Max.**	**Avg.**	**Max.**
Neural Network	0.350	0.637	0.873	46.076
Random Forest	0.243	0.746	1.260	45.504
Autoencoder	0.438	2.375	2.155	42.965

From this table it can be seen that the error for this prediction set is significantly less than that of the lower-bound data represented in table 7. Furthermore it is evident from this table that both the Neural Network and Random Forest are ideal for predicting the highly turbulent upper-bound CFD dataset. These observations can be further highlighted by analysing the error frequency plots of each model (*see Appendix IV*).

To further evaluate the outlier performance of each model, the respective CFD data was recorded. Following the contour graphing procedure for the Neural Network, the ensuing temperature prediction plot could be reproduced:

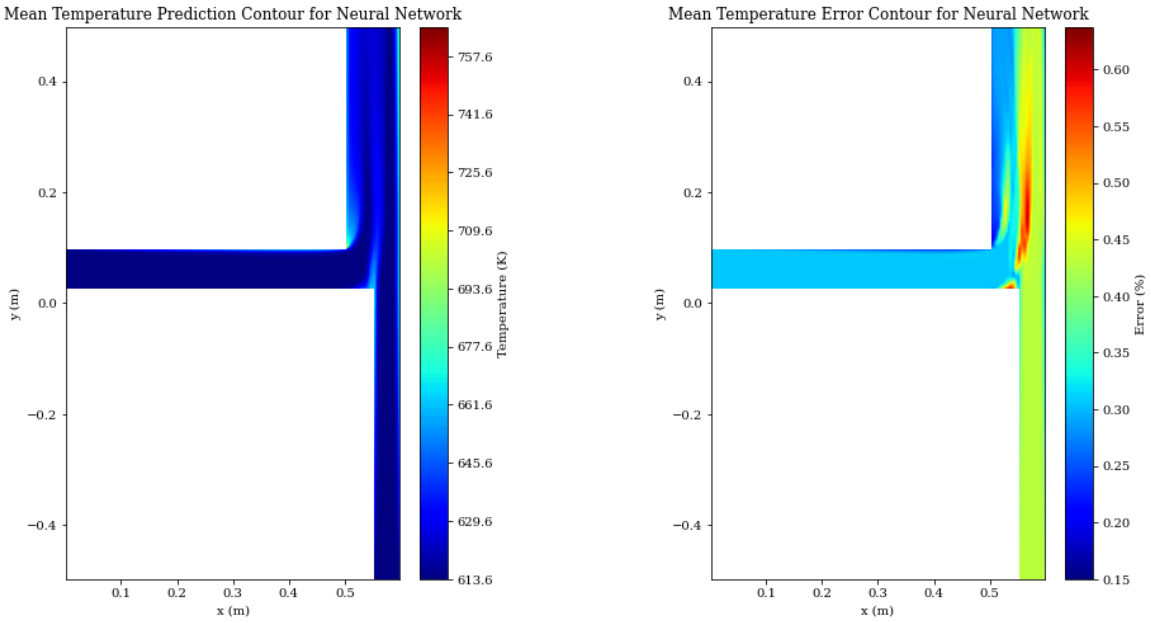


Figure 29: Upper-bound temperature prediction and error contour for Neural Network

From figure 29, it can be seen that the prediction error is significantly more accurate, and shows more uniformity than that of the lower-bound temperature prediction. It can also be noted that the highest prediction error occurred at the mixing interface (as with the training dataset error analysis).

Repeating the same process for the Neural Network's velocity prediction contour produced the following results:

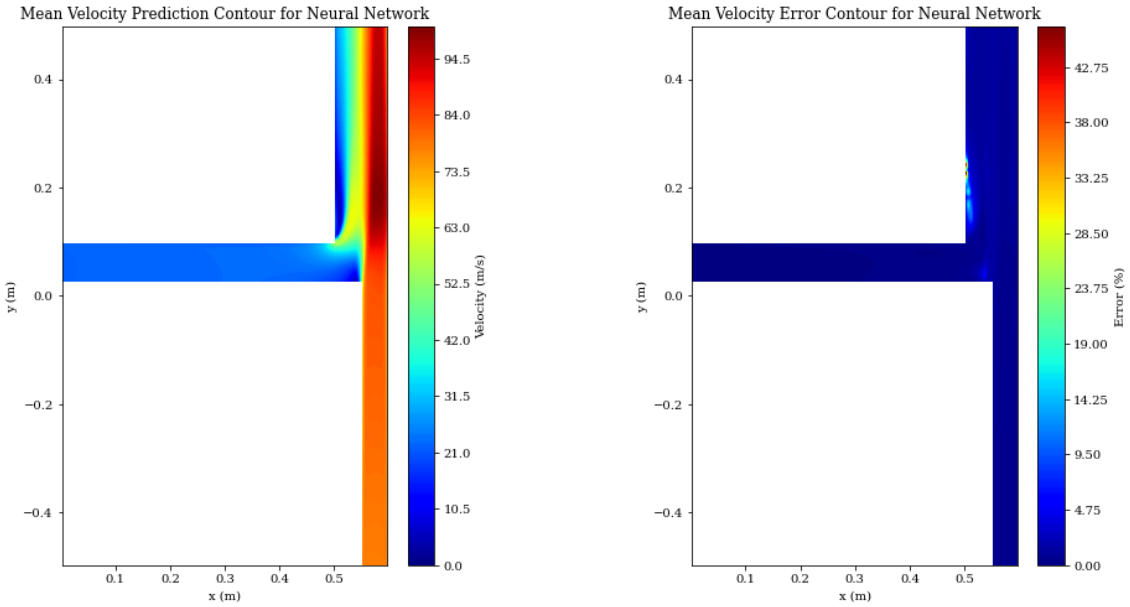


Figure 30: Upper-bound velocity prediction and error contour for Neural Network

As with the lower-bound temperature prediction contour, the NN velocity reconstruction showed less error than both the lower and initial data instance range. As mentioned before, this observation is most likely a result of the more turbulent characteristics of this specific dataset. Due to the nature of the original model training set, each algorithm was exposed to more turbulent flow trends as opposed to laminar regimes (resulting in the heightened performance on turbulent flow).

These reduced error characteristics of the upper-bound data are also present in the Random Forest predictions as seen in the temperature plot below:

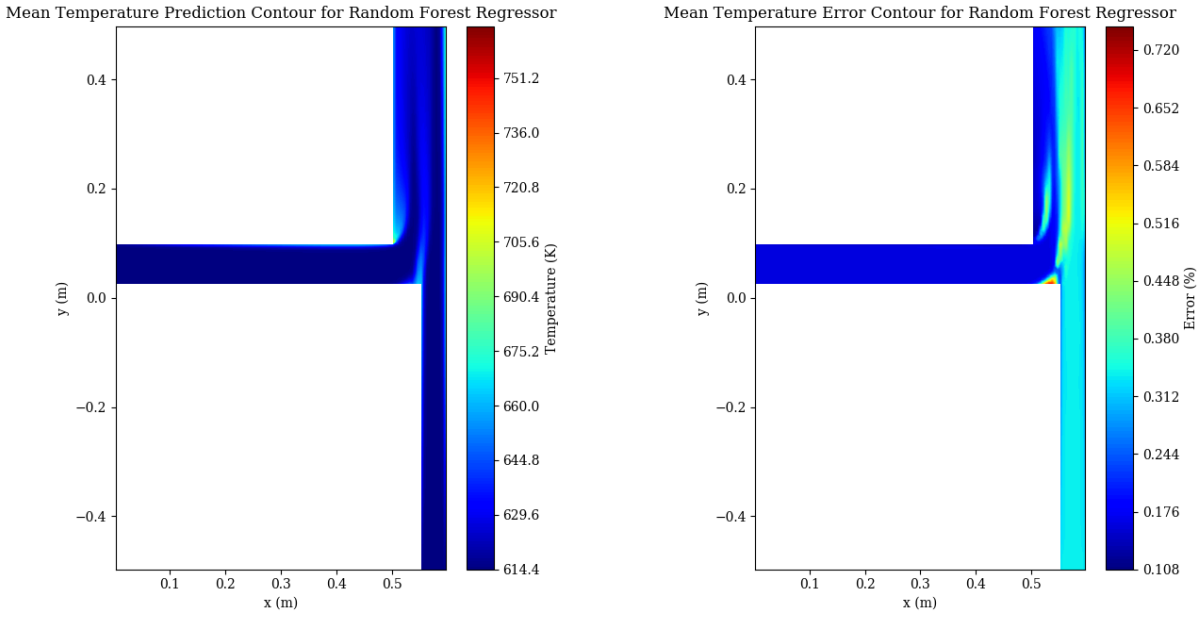


Figure 31: Upper-bound temperature prediction and error contour for Random Forest

From this temperature plot it can be seen that the Random Forest produced similar error results to that of the Neural Network. Although it can be noted that, even though the error range is greater, the prediction error across majority of the surface is significantly lower than the NN.

This low error observation can be further noted in the diagram below:

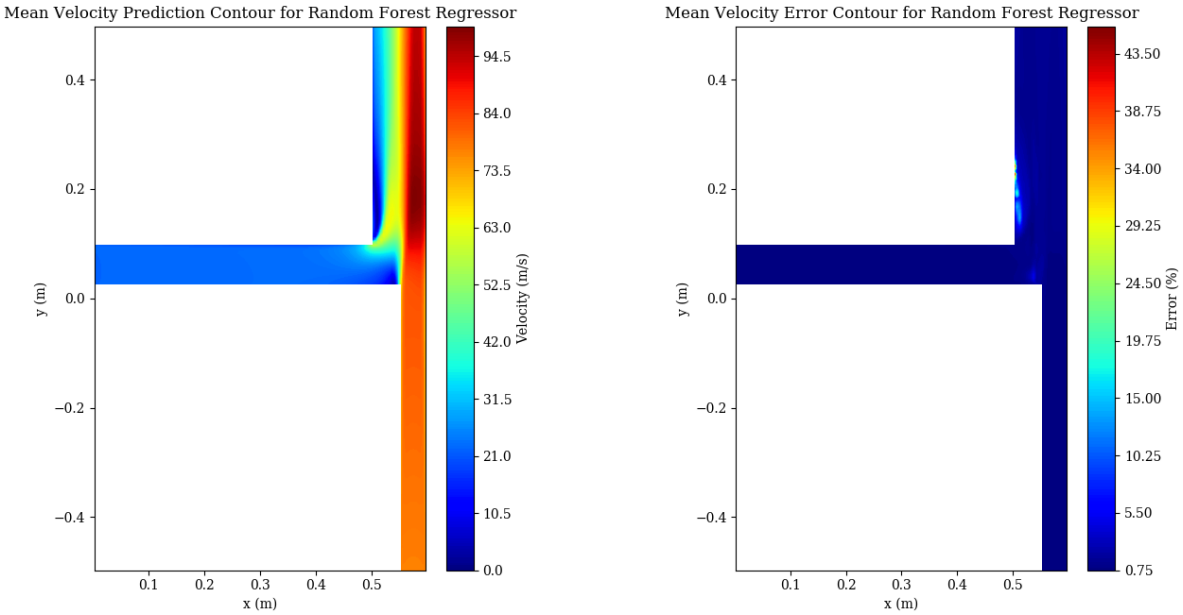


Figure 32: Upper-bound velocity prediction and error contour for Random Forest

It is evident that, unlike the lower-bound dataset, the Random Forest produced better overall flow contours than the Neural Network. This is because the Random Forest performs particularly well on the characteristically turbulent data.

The prediction and error contour below illustrates the Autoencoder's attempts to reconstruct the upper-bound temperature data:

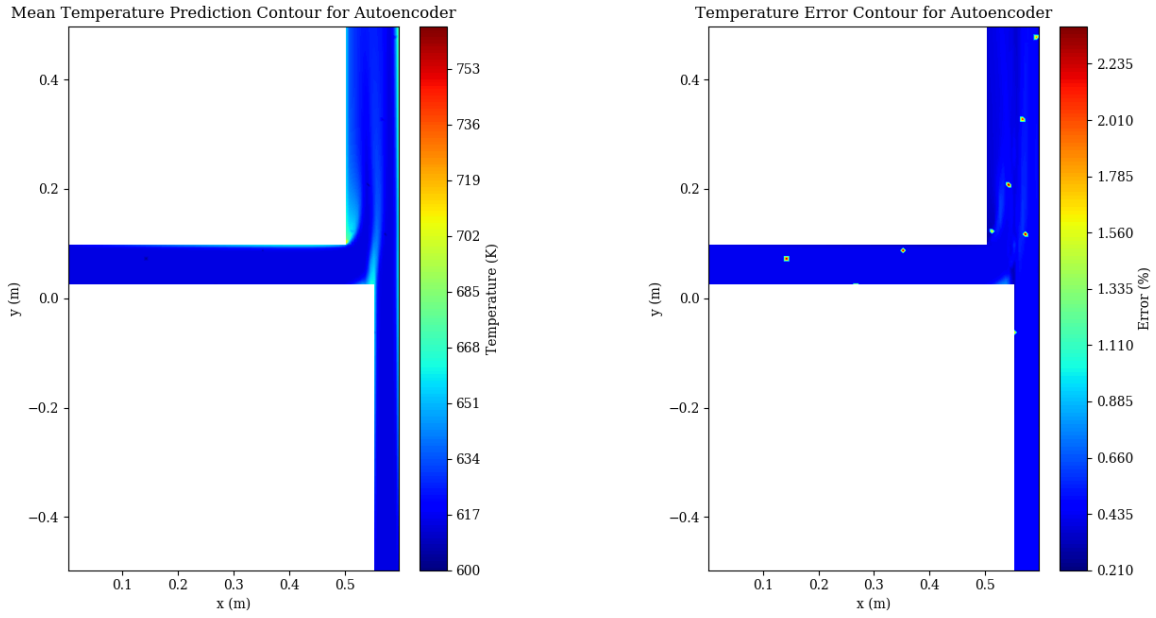


Figure 33: Upper-bound temperature prediction and error contour for Autoencoder

On observation of the Autoencoder's temperature prediction error it can be seen that the same points of high error are noticeable on the upper-bound contour. These inconsistencies (which are present in all the Autoencoder's predictions), ruin the visual validity of the contour reconstruction.

However, these loci of error are far less visible on the following upper-bound velocity contour:

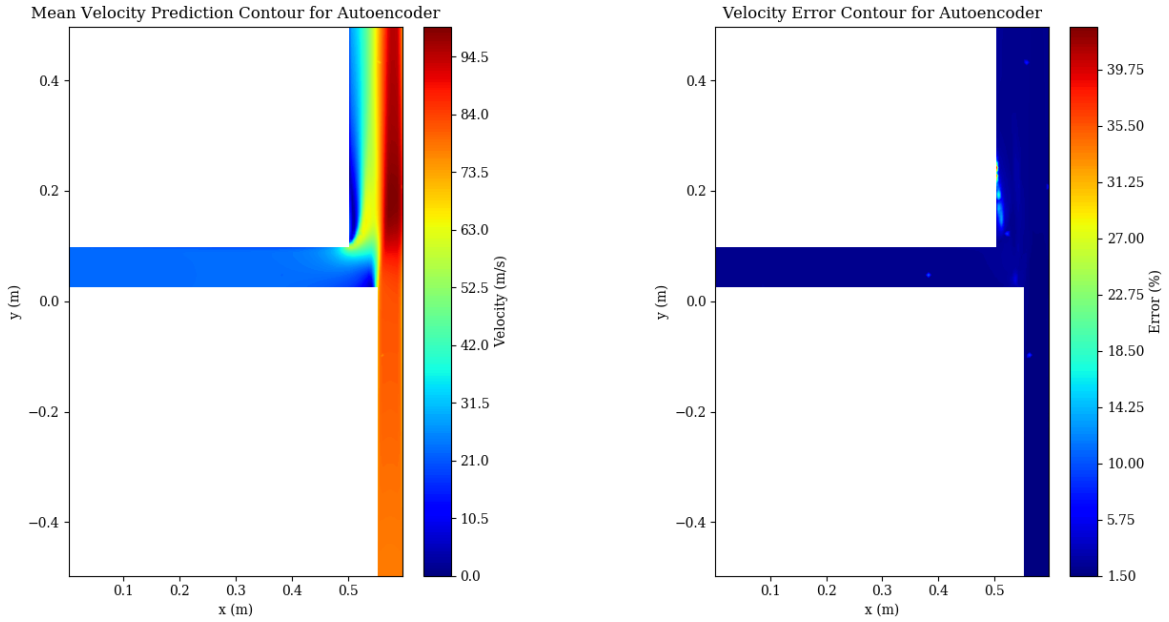


Figure 34: Upper-bound velocity prediction and error contour for Autoencoder

It can also be noted that, due to the reduced error of these uncertainty points, the Autoencoder produced the most precise overall upper-bound velocity prediction.

Discussion

By analysing the recorded MAE metrics for both the average prediction and contour generation, it was found that all three ML models produced reasonably accurate results when provided with instances that lay within the bounds of the training and validation dataset. Furthermore, it was also noted that these prediction results were still reliable when the models were expected to reconstruct data from provided instances that fell outside the original training data range (i.e. upper, and lower-bound instances). However, the shortcomings of these algorithms were also identified specifically on observation of the contour reconstructions. It was noted that temperature reconstructions struggled to fit the true CFD data trends at the mixing interface of the two inlet streams. This was indicative of the characteristically high temperature variations, and turbulent mixing that occurs within this region of the T-piece. Similar correlations were noted in all the reproduced velocity contours. As the prediction models were unable to correctly reproduce the flow velocity behaviour at the most turbulent regions. These observations were further highlighted when compared to the turbulence and Reynolds number contours computed during the CFD analysis (*see Appendix IV*).

From the overall algorithm operating performance analysis, it was noted that all three models were capable of making a CFD prediction, from a single instance, in near-real time. This was evident, as all three algorithms were capable of making a single prediction in a relatively imperceivable time frame (with the Random Forest producing the slowest runtime result of 0.343 seconds). On further analysis of the model prediction runtime, it was found that the CPU runtime for all three ML models scaled logarithmically ($n \log(n)$) with an increase in sample size. This means that algorithms may potentially lose their real-time prediction characteristics as the instance batch size increases. Similarly to classifying the runtime performance of each model, the memory utilisation metrics were recorded. These recorded flash and RSS values can be used to gauge the feasibility and practicality of implementing each model on an external CPU device (*see Appendix V*). From this analysis, it was found that Both the Autoencoder and Neural Network required relatively similar computational resources when operating. It was also surprisingly noted that although the Random Forest model demanded the greatest static memory storage, it used almost insignificant volatile memory when provided relatively small instance batch sizes.

Based on the detailed inspection of the Neural Network's recorded output error and error frequency (*see Appendix IV*), it was noted that the Neural Network produced the second best prediction results when provided with the original CFD dataset. Furthermore the Neural Network was the most adaptive model when generalising on outlier data. This accuracy enabled the Neural Network to produce some of the best contour reconstructions (and the best contour graphs for lower-bound data instances) of the two dimensional mixing T-piece. Moreover, this algorithm was capable of generating these contour maps with the lowest runtime and memory usage when compared to the performance of the other ML models. Thus these findings show that the Neural Network is best suited for applications that require relatively accurate, near real-time, contour representations of highly variant instance flow data (i.e. input flow that is both laminar, and highly turbulent), on devices which have characteristically low computational power (i.e. static and volatile memory resources).

From analysis of the Random Forest's operating performance, it was apparent that the regression model consistently produced the second most precise prediction error results, as well as the best overall CFD contour reconstructions for instances which lay in both the training set and upper-bound CFD ranges. However, it was noted that the Random Forest struggled to maintain this prediction accuracy for flow patterns that were inherently laminar (i.e. the lower-bound instance ranges). This was because the inflexible decision trees of the Random Forest ensemble were incapable of generalising on both the turbulent trends of the training set, as well as the vastly different laminar characteristics of the lower-bound flow regimes. This inherent overfitting, along with the relatively high runtime and static memory usages; limit the application potential of the selected Random Forest model. These limitations can be mitigated by regularising/restricting the depth and leaf node count of the pre-trained model, however this decision may be at the expense of the algorithm's highly accurate contour reconstruction ability. Therefore from these observations, it can be deduced that Random Forests are best suited for applications (with sufficient computational resources) that demand very accurate contour reconstructions when provided a finite and certain input data range (that preferably lies within the bounds of the original training set).

On observation of the Autoencoder's CFD prediction performance it was found that the Autoencoder produced the least accurate contour prediction results (on the original training and validation dataset). This was reflected in the high mean and maximum prediction errors, as well as the characteristically high range of recorded error frequencies (*see Appendix IV*). These high errors are mainly caused by the visual inconsistencies apparent on all the Autoencoder's CFD contours. This localised error phenomena is a result of the reconstruction losses that are inherent in these ML models. Nevertheless, by disregarding these high error values (*see Appendix IV*), it is evident that the rest of the contour reconstructions showed relatively consistent prediction results. Although this model was incapable of reconstructing the visual contour data as well as the other two algorithms, it can be noted that (like with the Neural Network) the Autoencoder's flexibility allowed it to adapt/generalise reasonably well on outlying data. Furthermore, another notable characteristic of the Autoencoder is that it displayed relatively similar runtime and memory results to that of the Neural Network. These observations are unsurprising, as the internal and operating structure of the Autoencoder is almost indistinguishable from that of the Neural Network. Due to these characteristic similarities, Autoencoders can be particularly useful for applications that require the remote implementation of these models on devices with low computational resources.

Conclusion

Proceeding the planning and initial project methodology detailed in the Interim Report (*see Appendix IV*), a comprehensive overview of the project could be developed. This allowed for the initial literature review, relating to the requirements of the CFD setup and ML models, to be conducted. From this literary investigation, the correct simulation parameters (in order to produce the most analytically representative simulation setup) for the mixing of air in the two dimensional T-piece design could be selected. Physical models such as the ideal gas law, and the K-Epsilon turbulence model were chosen for their computational efficiency and pipe flow simulation accuracy. Other parameters such as standard wall functions (as a means of near-wall treatment), and piecewise polynomial specific heats were also specified in order to produce a realistic flow scenario from which the machine learning data could be sourced. From the characteristics of the CFD data, and investigation into the various machine learning model types, it was concluded that supervised regression models were to be used. Furthermore, it was decided that the selected regression models were to be trained off a batched data set split with a training-testing ratio of 80 % : 20 %. Through the evaluation of past research, it was noted that the application of machine learning in fluid mechanics (and T-piece mixing in particular) was a relatively under-explored field. Although from the available research and experimental results, it was found that the best performing non-linear, supervised regression models were Neural Networks, and Random Forests. Thus these models, along with an Autoencoder algorithm, were selected to be trained and evaluated on their T-piece CFD prediction capabilities.

By running the CFD setup using ANSYS Fluent simulation software [9], a total of 1000 CFD data samples were acquired. These CFD results consisted of 10 000 mesh cell points representing the temperature and velocity magnitudes throughout the surface of the T-piece. The 1000 simulation samples were configured using a Latin Hypercube arrangement in ANSYS's Design of Experiments feature. This was to ensure each design point was uniquely recorded over a broad sample space, providing the ML algorithms with a more diverse dataset from which they were to be trained. From this training and selection process (*see Appendix III*), it was found that a simple feedforward three layered Neural Network with 128 neurones per layer performed best on the CFD data. Each layer was configured with the ReLu activation function as this function best suited the numeric characteristics of the data, as well as enabling the application of the Adam optimiser for back propagation. Likewise similar functions were applied to the Autoencoder, with the addition of Dropout and Batch Normalisation as an additional means of model regularisation. On further analysis of the Autoencoders performance, it was found that an Autoencoder consisting of a 32 dimensional intrinsic space decoder and a 512 neurone per layer Neural Network, produced the best contour reconstructions and error performance. From the selection process of the Random Forest, it was found that a randomised ensemble of 150 Decision Trees produced the most generalised results. These three selected models were then further evaluated on their CFD prediction performance.

Following the algorithm analysis and comparison, it was found that all three of the trained models were capable of making predictions in near-real time. Furthermore it was noted that this real-time prediction performance diminished logarithmically as the instance batch size was increased. Overall, each model produced reasonably accurate CFD prediction results; with average temperature errors ranging between $2.206\,\% - 5.503\,\%$, and mean velocity prediction errors falling between $3.626\,\% - 11.398\,\%$ (on the training and validation sample set data). Moreover these algorithms showed relatively good generalisability on outlying data, with noticeably higher error rates for prediction instances more correlated to laminar fluid flow. Overall it was found that Autoencoders produced the least desirable prediction results, but showed reasonable levels of prediction generalisability. It was also noted that the Random Forests was best suited for very turbulent flow contour reconstructions, as it produced the contour plots with the least errors and visual inconsistencies. However, although the Random Forest's predicted CFD contours were the most accurate, this model did not generalise well on the more laminar flow trends of the lower-bound datasets. Whereas, the Neural Network's prediction flexibility allowed it to better adapted on more variant flow parameters, this enabled the NN to produce more reliable reconstructions on all the provided data instances. This characteristic, along with the Neural Network's low operating memory requirements made this algorithm the preferred model for remote in-situ implementation/integration, on small scale microcontroller devices (*see Appendix V*), into current engineering systems.

Based on findings obtained from literary research, model selection, and algorithm experimentation; it can be concluded that machine learning regression models such as Neural Networks, Random Forests, and Autoencoders are capable of developing reasonably accurate CFD data predictions in near real-time. Furthermore, the operating efficiency of these algorithms make these models a particularly desirable engineering tool for in-situ solving of complex two dimensional CFD systems. This enables these models to be used as a virtual aid to current CFD dependent systems, improving the applicability of CFD and potentially other computational physics simulations. Due to the limitation of the project scope, various relating factors could not be explored. Thus for future projects it may be beneficial to examine whether these ML models maintain their prediction accuracy and runtime efficiencies for more realistic three dimensional CFD models. Other factors such as alternative forms of nonlinear dimensionality reduction can also be tested in-order to better improve the application potential of machine learning in fluid based systems.

References

[1] J. Morton, A. Jameson, M. J. Kochenderfer, and F. Witherden, "Deep dynamical modeling and control of unsteady fluid flows," in *Advances in Neural Information Processing Systems*, pp. 9258–9268, 2018.

[2] S. L. Brunton, B. R. Noack, and P. Koumoutsakos, "Machine learning for fluid mechanics," *Annual Review of Fluid Mechanics*, vol. 52, pp. 477–508, 2020.

[3] G. B. Nimadge and M. E. Student, "Cfd analysis of flow through t-junction of pipe," *IJSRD-International Journal for Scientific Research & Development|*, vol. 4, no. 2, pp. 906–911, 2017.

[4] L. Grbčić, L. Kranjčević, S. Družeta, and I. Lučin, "Efficient Double-Tee Junction Mixing Assessment by Machine Learning," *Water*, vol. 12, no. 1, p. 238, 2020.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[6] A. Géron, *Hands-On Machine Learning with Scikit-Learn*. Sebastopol: O'Reilly Media, Inc., first edit ed., 2017.

[7] R. S. Srinivasan and A. M. Malkawi, "Real-time simulations using learning algorithms for immersive data visualization in buildings," *International Journal of Architectural Computing*, vol. 3, no. 3, pp. 265–279, 2005.

[8] P. Cheema, G. Mathews, B. Thornber, and G. Vio, "Bayesian Inversion and Regression Trees for Mixing Length Model Development," 2016.

[9] ANSYS, "ANSYS Fluent - CFD Software | ANSYS," 2016.

[10] H. Ayhan and C. Sökmen, "CFD Modeling of Thermal Mixing In T-junction: Effect of Branch Pipe Diameter Ratio," 2013.

[11] W. Zuo, "Introduction of Computational Fluid Dynamics," *Joint Advanced Student School (JASS)*, 2005.

[12] C. Sert, "Finite Element Analysis in Thermofluids," 2018.

[13] A. Boatemaa, *Effects of Injection Pipe Orientation on Mixing Behavior in Contributing to Thermal Fatigue in a T-junction of a Pipe*. PhD thesis, UNIVERSITY OF GHANA, 2016.

[14] ANSYS Inc, *ANSYS FLUENT User's Guide*. No. 14.0, Canonsburg: ANSYS Inc, 2011.

[15] B. Hanna, N. Dinh, R. Youngblood, and I. Bolotnov, "Coarse-Grid Computational Fluid Dynamic (CG-CFD) Error Prediction using Machine Learning," 2017.

[16] Y. Zhao, H. D. Akolekar, J. Weatheritt, V. Michelassi, and R. D. Sandberg, "Rans turbulence model development using cfd-driven machine learning," *Journal of Computational Physics*, p. 109413, 2020.

[17] I. Sadrehaghighi, "Artificial intelligence (ai) and deep learning for cfd," 09 2020.

[18] M. Zaghloul, *Machine-Learning aided Architectural Design Synthesize Fast CFD by Machine-Learning*. PhD thesis, Alexandria University, Egypt, 2017.

[19] J. Ling and J. Templeton, "Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier Stokes uncertainty," *Physics of Fluids*, vol. 27, no. 8, p. 85103, 2015.

[20] G. Van Rossum and F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and Others, "Scikit-learn: Machine learning in Python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[22] T. E. Oliphant, *A guide to NumPy*, vol. 1. Trelgol Publishing USA, 2006.

[23] G. Kłosowski and T. Rymarczyk, "Using neural networks and deep learning algorithms in electrical impedance tomography," *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, vol. 7, 2017.

[24] B. Ulmann, "Theory:Foundations of Artificial Neural Networks," *Foundations*, vol. 1, 2003.

[25] J. Andersen, "Theory, Operation, and Application of Neural Networks," 2018.

[26] X. Zhou, "Understanding the convolutional neural networks with gradient descent and backpropagation," in *Journal of Physics: Conference Series*, vol. 1004, p. 12028, IOP Publishing, 2018.

[27] T. F. Cootes, M. C. Ionita, C. Lindner, and P. Sauer, "Robust and accurate shape model fitting using random forest regression voting," in *European Conference on Computer Vision*, pp. 278–291, Springer, 2012.

[28] A. Cutler, "Random Forests for Regression and Classification," 2010.

[29] M. G. Roberts, T. F. Cootes, and J. E. Adams, "Automatic Location of Vertebrae on DXA Images Using Random Forest Regression," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2012* (N. Ayache, H. Delingette, P. Golland, and K. Mori, eds.), (Berlin, Heidelberg), pp. 361–368, Springer Berlin Heidelberg, 2012.

[30] W. Wang, Y. Huang, Y. Wang, and L. Wang, "Generalized autoencoder: A neural network framework for dimensionality reduction," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 490–497, 2014.

[31] K. T. Carlberg, A. Jameson, M. J. Kochenderfer, J. Morton, L. Peng, and F. D. Witherden, "Recovering missing CFD data for high-order discretizations using deep neural networks and dynamics learning," *Journal of Computational Physics*, vol. 395, pp. 105–124, 2019.

[32] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[33] Andre Bakker, "Lecture 7 - Meshing," 2006.

[34] W. McKinney and Others, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, Austin, TX, 2010.

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[36] I. S.ERTESVÅG and B. F. MAGNUSSEN, "The Eddy Dissipation Turbulence Energy Cascade Model," *Combustion Science and Technology*, vol. 159, no. 1, pp. 213–235, 2000.

[37] A. Shiehnejadhesar, R. Mehrabian, R. Scharler, G. M. Goldin, and I. Obernberger, "Development of a gas phase combustion model suitable for low and high turbulence conditions," *Fuel*, vol. 126, no. 2014, pp. 177–187, 2014.

Appendix I - Project Structure

To appropriately illustrate the layout and operation of the projects components (data acquisition, algorithm development, and experimental analysis), a process flow diagram can be used. This visualisation is beneficial as it can better highlight the methodological approaches necessary to reconstruct/recreate the experimental procedures used to produce the observations and results depicted in the report. Due to the nature of the project layout, the major project components (i.e. data acquisition, algorithm training, model selection, and analysis and comparison) can be grouped into several sections. This categorisation enables the connections between project components to be better illustrated. The infographic displayed below shows this high level overview for the project structure and section grouping:

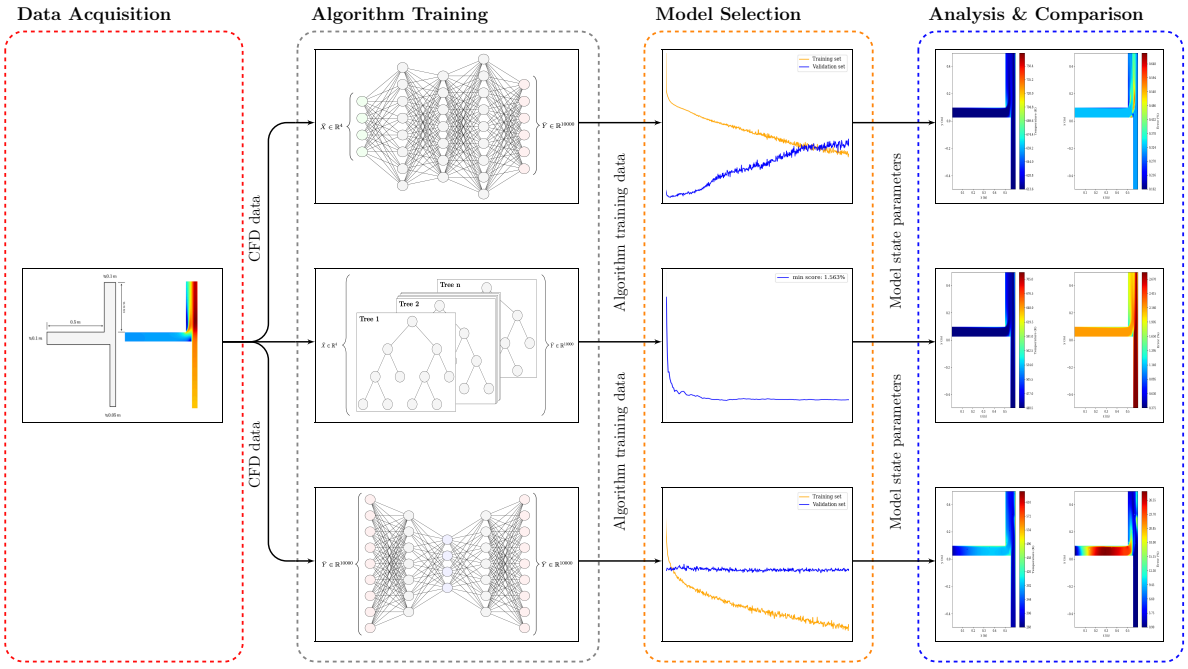


Figure 35: High level project structural layout and summary

It can be seen from the diagram above that the training, selection, and analysis of each algorithm is independent from the processes of the other algorithms. This means that the project can be modified/expanded by adding new ML models to the project scope (granted they are trained from the same CFD control data) without influencing the validity of the results obtained from the current prediction models.

The grouped sections shown in figure 35 can be further detailed by illustrating the sub-processes that occur within these grouped sections. Similarly to the high level infographic; the connections and relations between the sub-processes within each grouped section can be structured such that an accurate map of the project methodology can be developed. The process flow diagram below details the structure of the subprocesses that constitute the development and testing of the T-piece CFD prediction models:

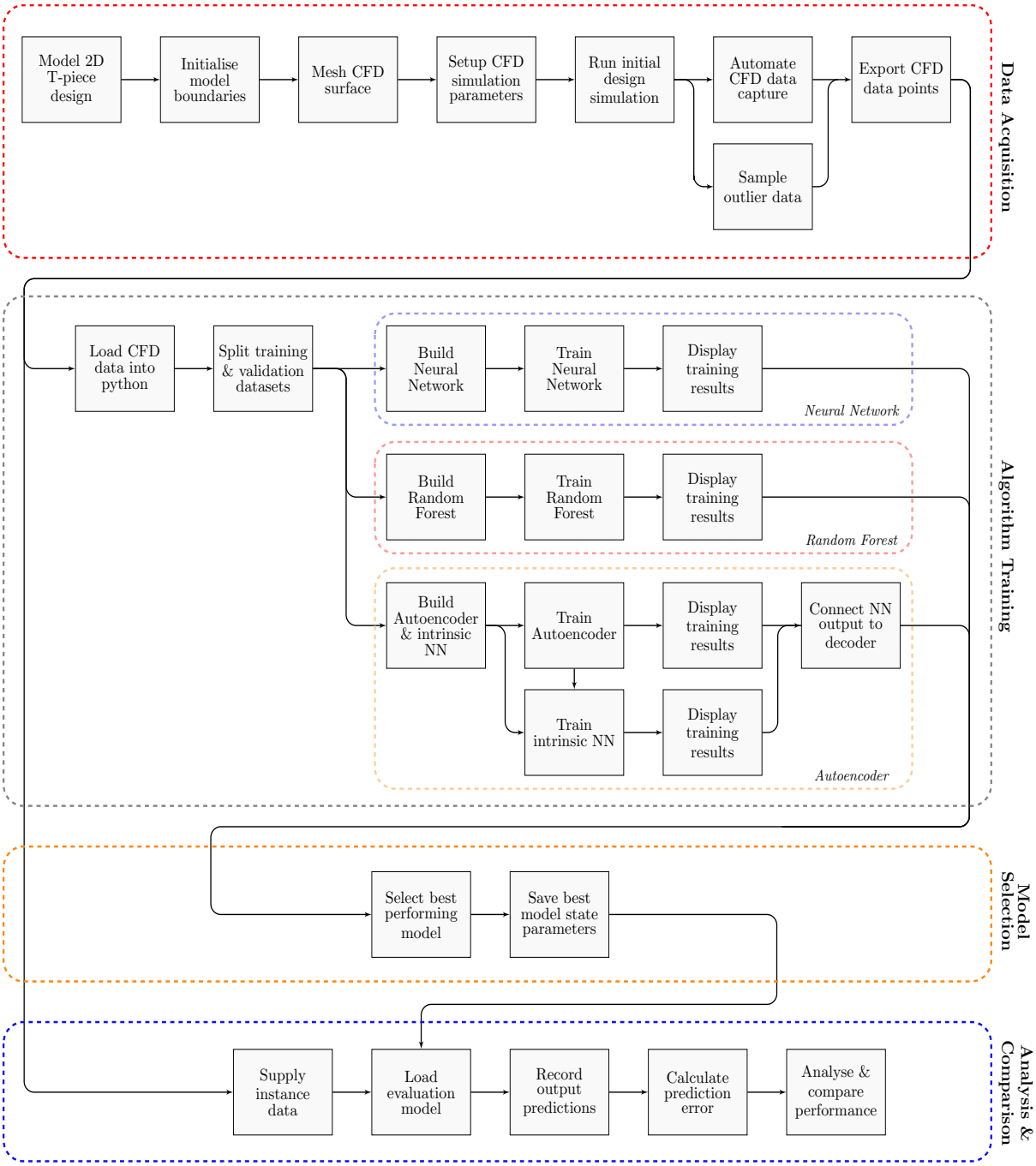


Figure 36: Process flow diagram for algorithm development and evaluation

From this diagram, the required sub-processes and section interfaces can clearly be identified. It is evident that this modular/pipeline project structure allows for each section to be separately modified without affecting the functionality of the adjacent subsystems (allowing errors/issues to be addressed without the need to modify the entire project).

Appendix II - Experimentation Theory

Mesh Modelling

The governing Navier-Stoke equations enable the resolution of fluid behaviour at the infinitesimal scale. These analytical formulae produce relatively realistic approximations for the dynamics of fluid control volumes. However, in order to utilise computational methods, to accurately determine the solution of the overall fluid system, the equations must be discretised. By approximating the respective fluid equations at a finite level, each fluid element can be computationally determined. This is achieved by dividing the work volume into relatively small gridded cells (also known as meshing). The most common methods for discretisation are the finite element, finite difference, and the finite volume method [11]. Typically the finite volume method is used in CFD as the conversion of mass, momentum, and energy are automatically satisfied for every cell (i.e. These parameters do not need to be manually adjusted for every control element) [11].

Correctly meshing a geometry is essential as these grids significantly impact the rate at which a simulation converges on a result, as well as the accuracy of the final solution [33]. For a good mesh quality, and subsequent solution, factors such as the grid density, adjacent cell size ratios, skewness of the cells, and how well the grid meshes with the boundary layer need to be taken into consideration [33]. Typically these factors can be optimised by selecting the correct mesh geometry (structured, unstructured, block structured cell geometries, etc...) [11]. For computational reasons block structured grids produce the most accurate simulation results and require the least computing time. However, these grids can only be used on very simple geometric shapes where the mesh is able to traverse the entire physical domain [33]. Due to the nature of the simple T-piece design, a block structured grid is best suited to analyse the CFD solution space. This is because more complex mesh structures may slow the rate at which the CFD simulation converges on a result. This grid structure also ensures a more accurate result across the whole mixing domain, as the size of each cell is approximately equivalent. The diagram overleaf illustrates the various technical components of the T-pice CFD mesh:

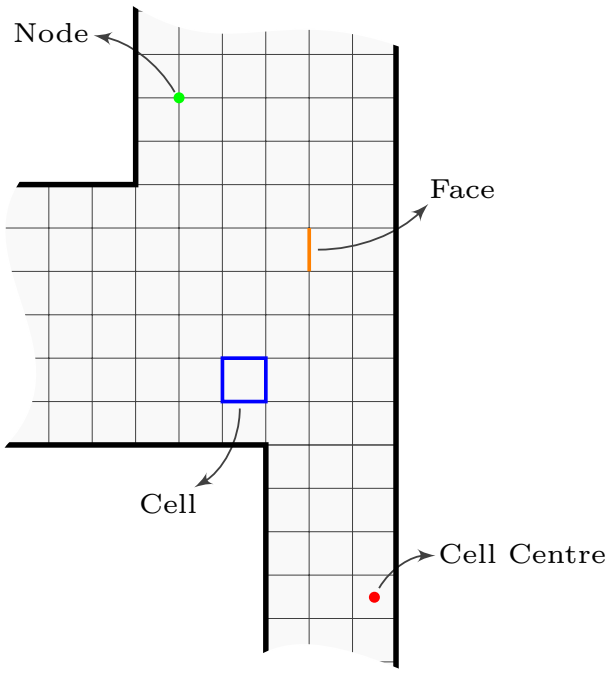


Figure 37: Grid mesh components and layout within the T-piece section

From figure 37 above it is evident that the simple surface design and two dimensional shape of the mixing domain allow for a more simplistic grid to be generated prior to the CFD simulation. It can also be noted that the majority of the cells have approximately the same area, this ensures the data sampled at these points is uniformly distributed across the surface of the fluid body. Thus by using this mesh structure, a relatively fine mesh size can be set in order to sample a sufficient number of data points, whilst still minimising the overall computational impact of the simulation process.

Data Processing

Before instance data can be passed to a machine learning algorithm for training, the data must be converted to a form that is legible by the ML program. This data processing is essential for selecting and extracting relevant input features, accounting for missing information, scaling data and creating the desired testing and training sets for the algorithm. Other than allowing the machine learning algorithm to operate, these data transformations can improve the models performance (by reducing factors such as the effects of overfitting and underfitting).

An effective method for processing large quantities of information is the use of data pipelines. These asynchronous processing structures work by sequentially concatenating the data, such that the outputs of each preceding component from the inputs of the next. Pipelines are a very common processing tools in machine learning, as each pipeline component performs as a self-contained system. This ensures that if a component is malfunctioning the rest of the pipeline remains intact [6]. Moreover, due to the simplistic structure of the data pipeline, these errors can be easily identified and corrected.

Using training data that has no relevant features relating to the output distribution can result in the ML model underfitting the data trend. Thus selecting the best features from the dataset to train the algorithm, is an essential consideration and necessary for the improvement of the models overall performance [6]. This feature selection can be done by plotting the distribution of the data. By looking for specific correlations between features, an understanding of what factors may affect the output of the function can be developed. Likewise, by using analytical methods these correlations may also be identified (i.e. noting proportionalities in the governing equations of the system from which the data was sourced) [8].

If the sourced data possesses irregularities (resulting in entries with missing/insufficient values), the machine learning model may incorrectly approximate data as these outliers could cause the sample data to deviate from its true mean. This data will inevitably produce an algorithm that fails to generalise when exposed to new instances. In order to prevent this phenomena from occurring: the entries can either be removed from the sample set, or the entries can be neutralised (i.e. set to zero, the mean, or the median of the sample data) [6].

Due to the large scaling variation of the input features, many machine learning algorithm struggle to approximate a solution [6]. This scaling variation typically results in the ML program overfitting the data, or in some cases failing to compute the input features. A solution to this problem is the application of feature scaling. Feature scaling reduces the variation of the input instances by proportionally scaling the input data range. The two ways of achieving this is by using min-max scaling or standardisation. Min-max scaling is often referred to as normalisation, typically this process involves scaling the result between 0 and 1 [6]. Min-max scaling often used in Neural Networks as the numerical operators within the neural layers require input features to be confined to a restricted range. Conversely to min-max scaling, standardisation does not bound the input features between a finite range. Standardisation works by setting the statistical mean of the input features to zero, and divides the distribution by the sample variance [6]. This method produces a scaled result with a unit variance, and is particularly useful when the dataset contains many outlying features.

The final step in the pipeline structure is to separate the processed data into separate testing and training sets. It is imperative that the datasets are separated randomly, as this ensures the sample mean and variance of each dataset is the same. This guarantees the data is evenly distributed among the respective testing and training sets (preventing the algorithm from overtraining on a biased sample of data which would result in overfitting). By validating the program's training progress an understanding of the algorithms performance can be developed and visualised.

Due to the large datasets that are associated with machine learning it can be relatively difficult to manage and manipulate the data during the pipeline process. As a result these large data samples are often organised in a structured data frame. This process is known as data wrangling[19] and it allows for large data sets to be represented in a manner that is simple to debug if an error were to occur during the data pipeline process [5].

[19]In the context of this project, the Pandas python library was used to *wrangle* the CFD data into its respective data frames [34].

Regularisation

Typically machine learning algorithms possessing internal structures capable of mapping data to a relatively high dimensionality, are prone to overfitting [6]. This overfitting can hinder the models ability to generalise new instance data that has been sampled beyond the scope of the original training set. As a result, the incapacity to accurately predict new instance data can drastically impact the algorithm's potential application and implementation in real-world situations [2]. Due to this issue, most algorithms require some form of regularisation to counteract the problems associated with overfitting. Furthermore, the regularisation of ML programs can enable the algorithm to accurately converge on a solution with significantly less data (as regularisation can act to reduce the complexity of the prediction space) [19].

Depending on the type of model being used different techniques for regularisation can be applied in order to produce an accurate algorithm. Some examples of regularisation methods include: normalising the preprocessed data to limit the prediction range the algorithm is expected to output, reducing the dimensionality of the overall dataset (especially when the prediction domain is significantly larger than that of the input dimension), limiting the weighting of the cost function, modifying the learning rate hyper-parameter, and by restricting the overall structure of the model [5]. These techniques prevent the algorithm from favouring the provided training instances, and limit the complexity of the prediction data. By using these methods, the algorithm is capable of learning the data trends without also acquiring the natural noise associated with realistic data [19].

It is evident from the nature of the fluid system in question that the difference in dimensionality between the input and output instances are significantly large. As a result it must be noted that the structure of the problem is inherently prone to overfitting. Thus techniques such as dimensionality reduction, and data normalisation are to be explored. The effects of these forms of regularisation will reduce the amount of training data required to produce an accurate final algorithm (subsequently reducing the total time dedicated to accumulating CFD data). moreover, due to this high risk of overfitting, several different structures for each algorithm are to be evaluated. This will allow for the best overall model design to be selected for further performance analysis.

Appendix III - Model Selection

Before the different machine learning algorithm types can be compared on their CFD data performance, the optimal/near-optimal models must be developed. In order to produce the best performing Neural Network, Random Forest, and Autoencoder for the particular T-piece contour data; varying algorithm characteristics and regularisation techniques need to be implemented. Using the baseline model structures and parameters detailed in the Materials and Methods section of the report (*see Materials and Methods*), the best performing algorithm for each model type can be developed. The following sections describe the procedural methods used to identify and select the best performing algorithms for each ML category.

Neural Network Selection Process

The structure of the NN was developed using built-in Pytorch modules [35]. An advantage to using Pytorch libraries is that each layer of the network can be uniquely customised to the desired specification. The Pytorch Neural Network module [35] allows activation functions, neurones, and regularisation methods to be specifically added to individual layers. Pytorch also conveniently allows the network to be run on a graphics processing unit (GPU), for faster parallelised training. It must be noted that in order to achieve this, the normalised data arrays must be converted to Torch tensor format. From which these tensors are to be loaded into the multilayer perceptron network (MLP)[20].

To initially evaluate the model, the general hyper-parameters associated with the learning process were kept constant to ensure the optimal network structure was selected. It was decided that the algorithm would be trained for 500 epochs, as this is typically a sufficient number of iterations required to evaluate the overall network convergence. For motives similar to these training cycles, a learning rate hyper-parameter of 0.001 was also applied to the network. This learning rate is a reasonably low value that would ensure the gradient descent was performed in a controlled manner. Beyond these factors, the friction and decay parameters (that are necessary inputs in the Adam optimiser) were left at default, as these values do not usually need to be modified. The pseudo-algorithm over-page shows the general implementation of the NN Python code:

[20] A multilayer perceptron network is another term used to refer to feedforward artificial Neural Networks.

Algorithm 2: Neural Network training process

Data: Normalised training and testing NumPy arrays

```
NN Class(Input dim, Hidden layer dim, Output dim):
        ⇒ create NN module
        ⇒ initialise network structure
        ⇒ define feedforward sequential model
Cost Function(Model prediction, Actual output):
        ⇒ define MSE cost function
convert data NumPy arrays to PyTorch tensors
create PyTorch Datasets and Dataloaders
define network structure, and send to GPU device (1)↩
initialise Adam optimiser and learning rate
for Epoch in Range(500):
    set network to training mode
    for Input & Output in Training Set:
        reset gradient optemiser
        make network prediction
        calculate total error (5)↩
        perform backpropagation on error
        recompute network weights
    end
    set network to evaluation mode
    validate prediction error from testing set error
    display epoch and corresponding error
end
plot training and validation results
save graphed results
```

Result: Export trained NN model state parameters

Following the process of the algorithm above, several different network combinations were implemented and validated. In conventional terms it is typical to use neurone counts in binary orders of magnitude. Thus the table below shows the six different network structures used[21]:

Table 10: Neurone quantities per layer for varying Neural Network structures

Network Name	$\mathcal{Z}^{\mathbf{0}}$	$\mathcal{Z}^{\mathbf{1}}$	$\mathcal{Z}^{\mathbf{2}}$	$\mathcal{Z}^{\mathbf{3}}$	$\mathcal{Z}^{\mathbf{4}}$
16 Neurones per Layer	4	16	16	16	10000
32 Neurones per Layer	4	32	32	32	10000
64 Neurones per Layer	4	64	64	64	10000
128 Neurones per Layer	4	128	128	128	10000
512 Neurones per Layer	4	512	512	512	10000
1024 Neurones per Layer	4	1024	1024	1024	10000

[21]Note: the naming convention of each layer is in accordance with that of figure 7.

From the table above it can be seen that, for simplistic reasons, each network structure was to have the same neurone count per hidden layer. It is also evident that each network is conveniently referenced by the number of characteristic neurones in each hidden layer.

The six different network structures were all run on a Nvidia GeForce 920MX laptop graphics unit. The approximate training time varied depending on the size of each network, although each NN structure was able to be successfully computed on the GPU in less than 24 hours. The mean squared error for both the training and validation prediction sets were used to gauge each network's performance. The figure below shows this prediction performance for all six Neural Networks on the normalised CFD data:

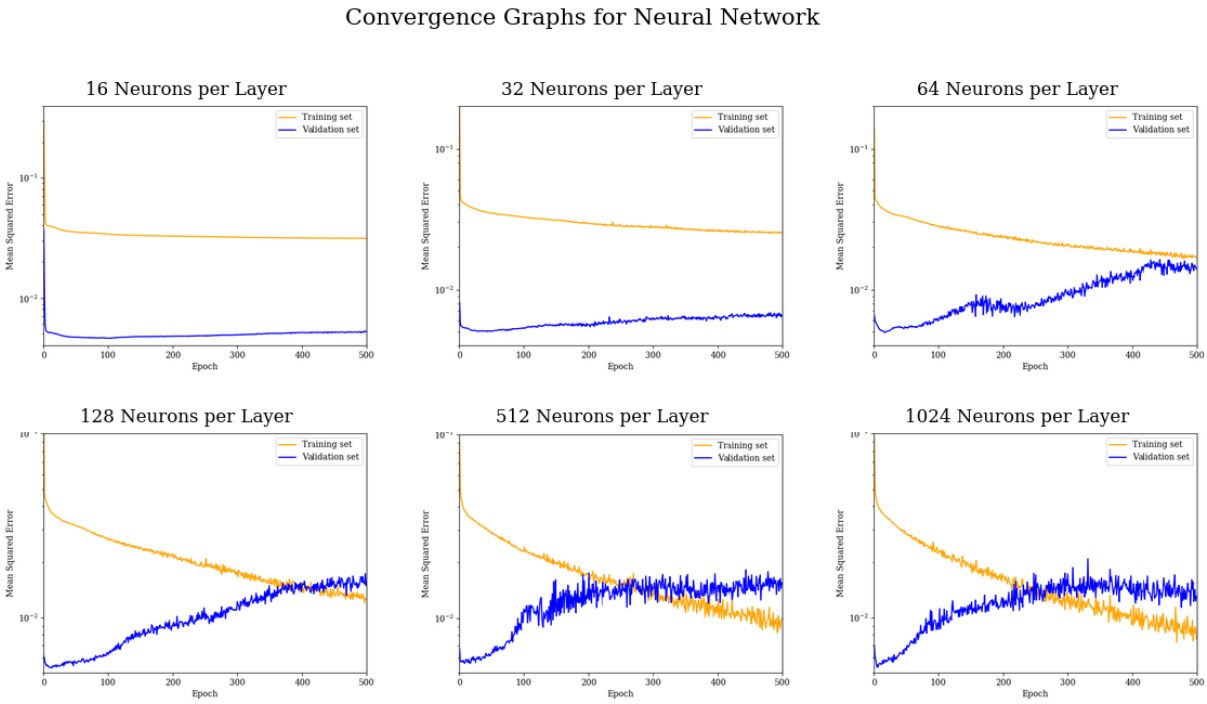


Figure 38: Neural Network training and validation convergence graphs

Based on the convergence graphs above it is evident that both the 16 and 32 neurone networks were incapable of properly converging. This is evident as both the training and validation curves are relatively flat. This seems to indicate that these network structures are unable to completely converge on the correct solution due to the limitation of their simplistic structure, which hinders the algorithms ability to map high dimensional CFD data.

Ideally the training and validation errors must be approximately the same (with the training error slightly lower than that of the testing set), as this indicates the algorithm is successfully able to generalise both the training and unseen datasets. From figure 38 it is apparent that the 64 neurone per layer network is unable to meet this ideal convergence criteria. It is evident that as the network size increases the rate of convergence increases. From this observation it is clear that the 128, 512, and 1024 neurone networks all fit the criteria for an ideal network. However, from closer evaluation and the display recorded on the Python console [20], it is apparent that the 128 neurone per layer NN produced the training and validation set with the lowest error at 500 epochs. Henceforth based on the performance graphs in figure 38, the 128 neurone per layer network structure is to be further analysed and compared on its ability to predict the CFD data.

Random Forrest Selection Process

Random Forests are particularly desirable for application in fluid related problems, due to the prediction accuracy and dimensionality characteristics associated with these ML models. Another advantage relating to these algorithms is that they are fast to train, and do not require any significant data preprocessing. Although, due to the high dimensionality and overfitting susceptibility of the T-piece CFD data, the predictors and solutions that are to be supplied to this particular Random Forest will be normalised. This is because normalising the data further reduces the algorithm's susceptibility to overfitting.

To accurately determine the best Random Forest structure, a range of models with varying accumulations of decision trees are to be trained and tested. From this training process, the model structure with the lowest validation error will be selected. The algorithm below shows how this Random Forest selection is to be implemented using the normalised CFD data:

Algorithm 3: Random Forest training process

Data: Normalised training and testing NumPy arrays

load normalised NumPy arrays

for *Tree Count* **in** *Range(200):*
    create Random Forest regressor
    assign tree count to initialised regression algorithm
    fit and train Random Forest on testing NumPy arrays
    run test predictions and compute `MSE`
    display `MSE` error to console
end

plot and display prediction `MSE` for varying tree counts
save model structure with lowest error value

Result: Export saved Random Forest model

It is clear to see from the algorithm above, that Random Forests are particularly simple to implement and train. It is also evident that a range of 200 different forest structures are to be trained and validated. To test which structure produced the lowest error without overfitting, no regularisation parameters were added to the initial structure. By validating the unaided algorithm's performance on the CFD data, the optimum model structure could be deduced.

This particular regressor was constructed using the built-in Random Forest libraries provided by Scikit-Learn. These particular libraries are significant, as they utilise the CART cost function as a means of tree partitioning [6]. A simple Random Forest structure was then developed and trained using the algorithm above. This training was performed using a quad-core Intel i5 processor. The training of all 200 algorithm structures took approximately 4 hours to fully train (the individual times of which varied depending on the size of each regressor). In accordance with the specified algorithm, the errors for each forest structure was recorded. The following graph shows the validation performance of all 200 tree counts on the CFD data for a simple, unregularised Random Forest:

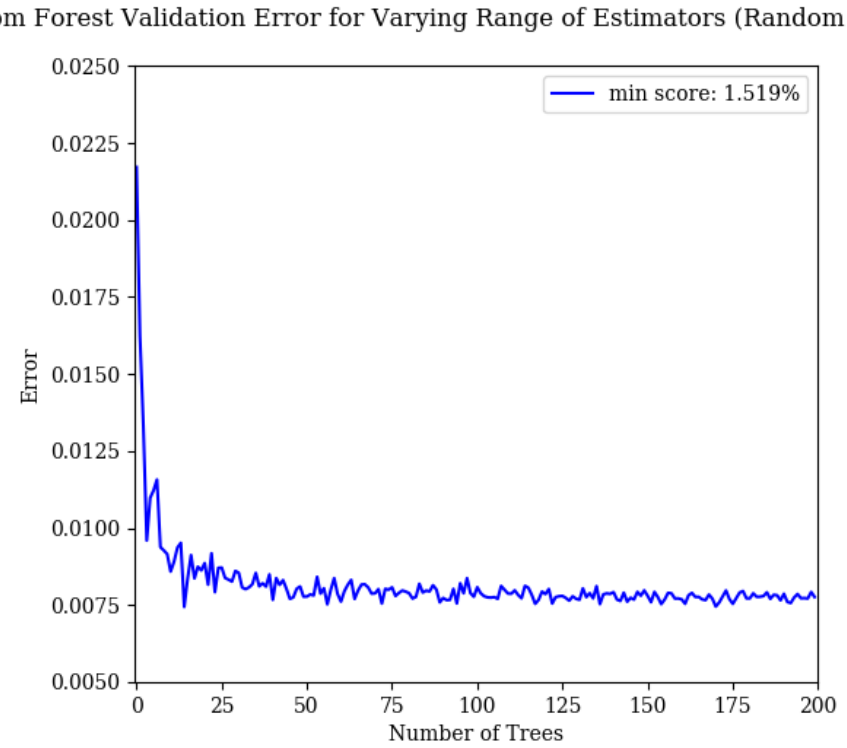


Figure 39: Validation curve for simple Random Forest structures

It is evident from the unregularised regressor performance depicted in figure 39, that the validation error of Random Forests comprising of over 50 trees ceases to converge (with a minimum prediction error of 1.519 %). It can also be noted that the validation error for differing network structures is very stochastic, this can mostly be due to partial overfitting on the training set. In order to mitigate this problem a random state hyper-parameter can be added to the regressor as a means of better randomising the individual Decision Trees. The following modified validation curve shows the implementation of the same algorithm, with the addition of the specified hyper-parameter [22]:

[22]Note: a random state of 42 was applied to the Random Forest structure, as this value coincided with conventional computational practices.

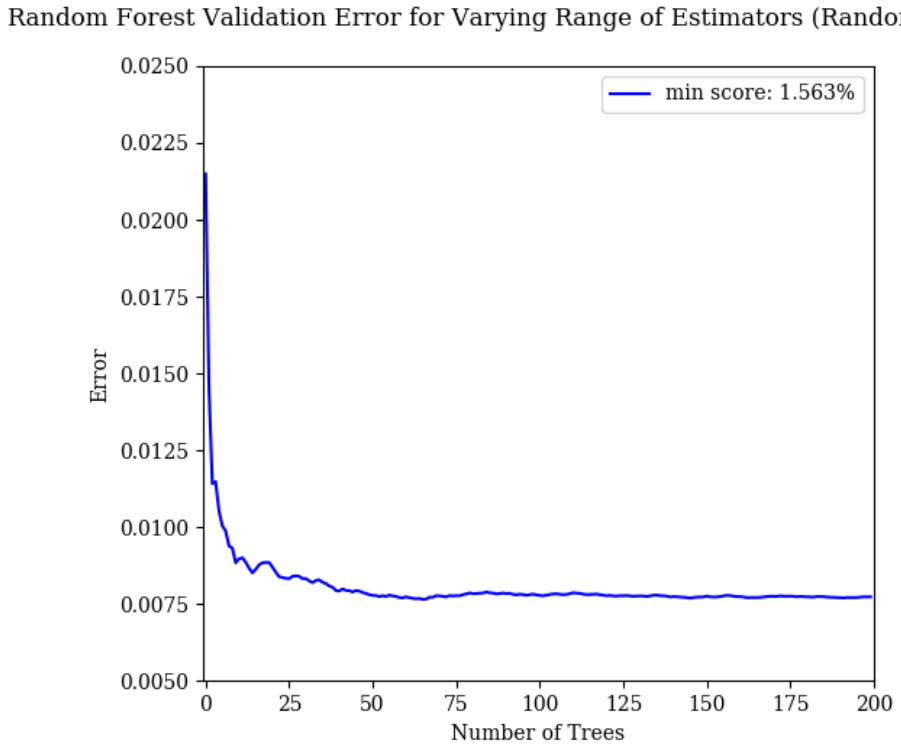


Figure 40: Validation curve for Random Forests with added random state

From figure 40, it can be seen that the additional random state parameter reduced the stochastic effects of the algorithm validation. This hyper-parameter can also increase the robustness of the final algorithm, as the trained model will be able to better generalise predictions when provided unseen instances. Thus, from the information represented by the above graph. It was deduced from observation that a Random Forest of 150 randomised Decision Trees[23], and the specified random state would be used to further test the algorithm performance on the CFD data.

[23]This number was selected, as the model performance on this structure displayed no signs of overfitting, as well as being within the stabilised/fully converged region of the graph.

Autoencoder Selection Process

To train an Autoencoder with such high dimensionality CFD data, a very deep network structure is was used. In order to test which Autoencoder structure is optimal for dimensionality reduction, a variety of network models were to be tested and compared. The table below shows the internal structures of three different networks that were tested:

Table 11: Neurone quantities per layer for varying Autoencoder structures

Autoencoder Name	$\mathcal{A}^0$	$\mathcal{A}^1$	$\mathcal{A}^2$	$\mathcal{A}^3$	Ω^4	$\mathcal{Z}^5$	$\mathcal{Z}^6$	$\mathcal{Z}^7$	$\mathcal{Z}^8$
16 Dimensional Autoencoder	10000	128	64	32	16	32	64	128	10000
32 Dimensional Autoencoder	10000	512	128	64	32	64	128	512	10000
64 Dimensional Autoencoder	10000	1024	512	128	64	128	512	1024	10000

It is evident that the naming convention of each network is based on the dimensionality of the resulting encoding space.

Similarly to the training of the Neural Network, each Autoencoder structure (displayed in table 11 above) was designed and implemented in Pytorch [35], using the same GPU device previously specified. The resulting training and validation error convergence graphs, for all three different network structures, were also plotted and compared. This comparison can be clearly highlighted in the figure below:

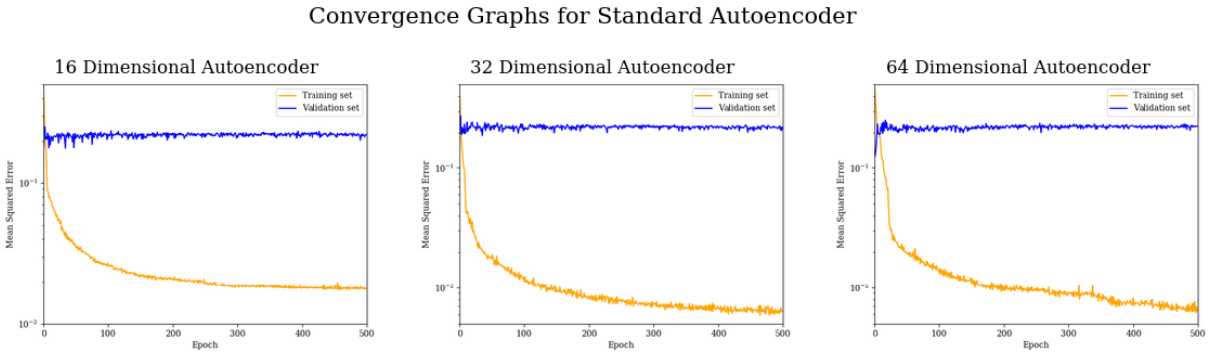


Figure 41: Convergence graphs for variable Autoencoder structures

From the resulting profiles of each of the above convergence graphs, it is apparent that the difference between testing and validation error for all three illustrations are orders of magnitude apart. This is a crucial observation as it indicates that all three of the model structures are heavily overfitting. The overfitting is most likely caused by the deep Autoencoder structure.

Thus in order to overcome this overfitting issue, a battery of regularisation models was to be applied to each of the three networks. This was done by training and validating each network on using both Dropout and Batch Normalisation regularising techniques. Each of the three networks were trained using Dropout, Batch Normalisation, and a combination of the two methods. The results of these successive training processes were then to be re-compared and a final Autoencoder structure would be selected. Depending on the type of regularisation methods used, the contrasting regularising forms were to be negated. The pseudo-algorithm over-page details the exact Python process used and applied to each of the three models:

Algorithm 4: Autoencoder training process

Data: Normalised training and testing output NumPy arrays

`Autoencoder Class(`*Visible dim, Hidden dim 1, Hidden dim 2, Hidden dim 3, Hidden dim 4*`):`
    ⇒ create Autoencoder module
    ⇒ initialise network structure
    ⇒ initialise dropout rate
    ⇒ define encoder sequence with 1D batch-norm
    ⇒ define decoder sequence
    ⇒ define feedforward sequential model

`Cost Function(`*Model prediction, Actual output*`):`
    ⇒ define `MSE` cost function

convert output data NumPy arrays to PyTorch tensors
create PyTorch Datasets and Dataloaders
define network structure, and send to GPU device `(1)`↩
initialise `Adam` optimiser and learning rate

for *Epoch* **in** *Range(500):*
    set network to training mode
    for *Input & Output* **in** *Training Set:*
        reset gradient optimiser
        make network prediction
        calculate total error `(5)`↩
        perform backpropagation on error
        recompute network weights
    end
    set network to evaluation mode
    validate prediction error from testing set error
    display epoch and corresponding error
end

plot training and validation results
save graphed results
save encoding space data array

Result: Export trained Autoencoder model state parameters and encoding dimension

It must be noted that a dropout rate (specified in the algorithm above) of 0.5 was selected to be applied. This is because this specific value corresponds with conventional machine learning practices, as previously mentioned. Furthermore it must be noted that the Dropout, and Batch Normalisation methods were only applied to the encoding layers of the Autoencoder section. This was done, as the addition of these regularisation techniques can potentially increase the overall run-time of the evaluated network, potentially compromising the real-time performance ability of the final algorithm [6].

Using the specified algorithm, the Autoencoders were retrained using the applied regularisation techniques. From the training and validation results of the 16 dimensional Autoencoder, the following convergence trends could be recorded:

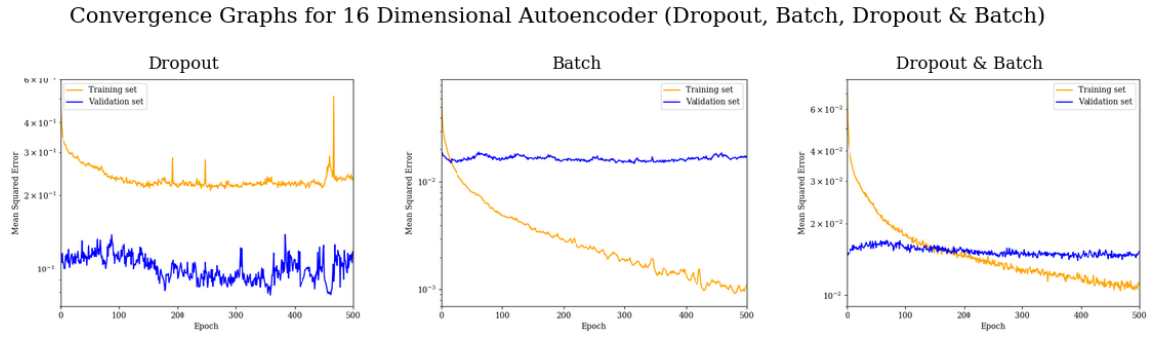


Figure 42: 16 neurone Autoencoder convergence for Dropout, Batch, and combination

From the graphs above it can be seen that the addition of dropout severely regularised the algorithm, to the point where the model was struggling to fully converge on the training data. Conversely the application of Batch Normalisation had a very small regularising effect. However it can be seen that the addition of both regularisation methods resulted in an ideal convergence, as evident in the figure above. This improved convergence was due to the complementation of both the Dropout and Batching characteristics. Although the Dropout technique was unable to converge on the original training data, the normalisation effect provided by the batching of neurones allowed for the intrinsic relationship of the normalised data batches to be better learned.

The same process as above was then repeated for the 32 dimensional Autoencoder, the results of which are depicted below:

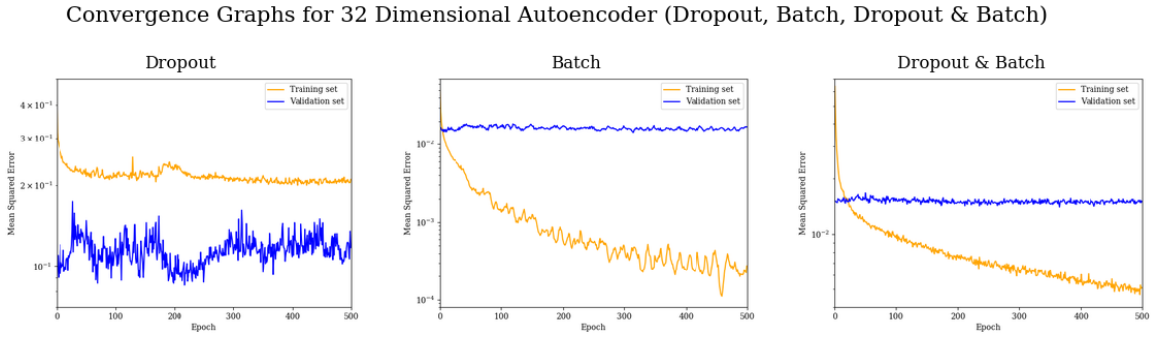


Figure 43: 32 neurone Autoencoder convergence for Dropout, Batch, and combination

Similarly to the 16 dimensional Autoencoder the convergence result using the combination of Dropout and Batch Normalisation were best produced when the collective of both regularisation models. Subsequently the result of the larger network yielded a greater bias towards the training data when compared to the 16 neurone algorithm. However based on the Dropout and Batch regularised results for the two Autoencoder structures, it was evident that both networks were to be further evaluated in order to select the best performing overall model structure.

From the results of the third regularised Autoencoder structure, it is clearly apparent that even. with the application of the regularisation techniques, none of the Autoencoder models were capable of generalising in the data. This is apparent in the diagram overleaf; as all three model types either failed to converge sufficiently, or produced testing errors that were orders of magnitude lower than the validation samples:

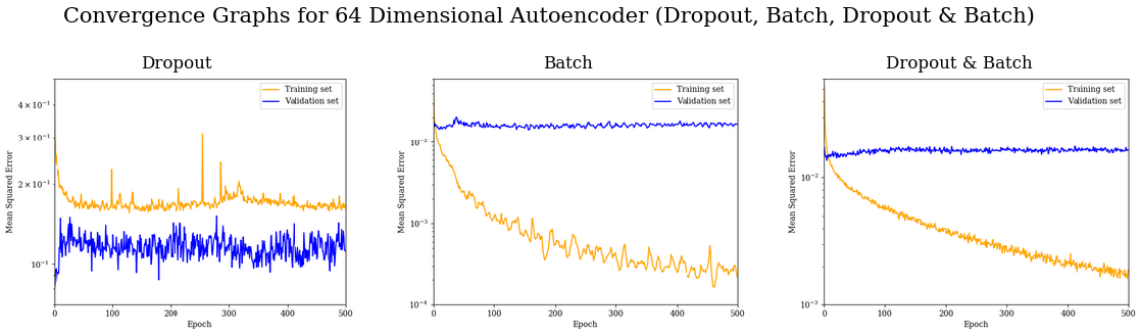


Figure 44: 64 neurone Autoencoder convergence for Dropout, Batch, and combination

Thus based on the comparison conducted above, it was deduced that only the 16 and 32 dimensional Autoencoders (with the application of Dropout and Batch Normalisation) were to be further evaluated. The Algorithm below displays the required steps required to develop a Pytorch Neural Network [35]:

Algorithm 5: Encoding Neural Network training process

Data: Normalised training and testing NumPy arrays and encoding space

```
NN Class(Input dim, Hidden layer dim, Encoding dim):
        ⇒ create NN module
        ⇒ initialise network structure
        ⇒ define feedforward sequential model
Cost Function(Model prediction, Actual output):
        ⇒ define MSE cost function
convert input and output data arrays to PyTorch tensors
create PyTorch Datasets and Dataloaders
define network structure, and send to GPU device (1)↩
initialise Adam optimiser and learning rate
for Epoch in Range(500):
    set network to training mode
    for Input & Output in Training Set:
        reset gradient optemiser
        make network prediction
        calculate total error (5)↩
        perform backpropagation on error
        recompute network weights
    end
    set network to evaluation mode
    validate prediction error from testing set error
    display epoch and corresponding error
end
plot training and validation results
save graphed results
```

Result: Export trained encoding NN model state parameters

This network algorithm above is to be used to construct a Neural Network that is capable of predicting the manifold spaces of both the 6 and 32 dimensional Autoencoders. Similarly to the standard Neural Network, a variety of differing model structures are to be trained and validated, in order to deduce which network is ideal for the prediction of the associated encoding spaces. The network structures for each Neural Network can be shown in the table below:

Table 12: Neural Network structures for differing encoding spaces

Encoding Space	**Neural Network**	$\mathcal{Z}^0$	$\mathcal{Z}^1$	$\mathcal{Z}^2$	$\mathcal{Z}^3$	Ω^4
16 Dimensional Autoencoder	16 Neurones per Layer	4	16	16	16	16
	32 Neurones per Layer	4	32	32	32	16
	64 Neurones per Layer	4	64	64	64	16
	128 Neurones per Layer	4	128	128	128	16
	512 Neurones per Layer	4	512	512	512	16
	1024 Neurones per Layer	4	1024	1024	1024	16
32 Dimensional Autoencoder	16 Neurones per Layer	4	16	16	16	32
	32 Neurones per Layer	4	32	32	32	32
	64 Neurones per Layer	4	64	64	64	32
	128 Neurones per Layer	4	128	128	128	32
	512 Neurones per Layer	4	512	512	512	32
	1024 Neurones per Layer	4	1024	1024	1024	32

Using the structures listed in table 12 and following the procedure specified in the above algorithm, the convergence results for each Neural Network model could be recorded below:

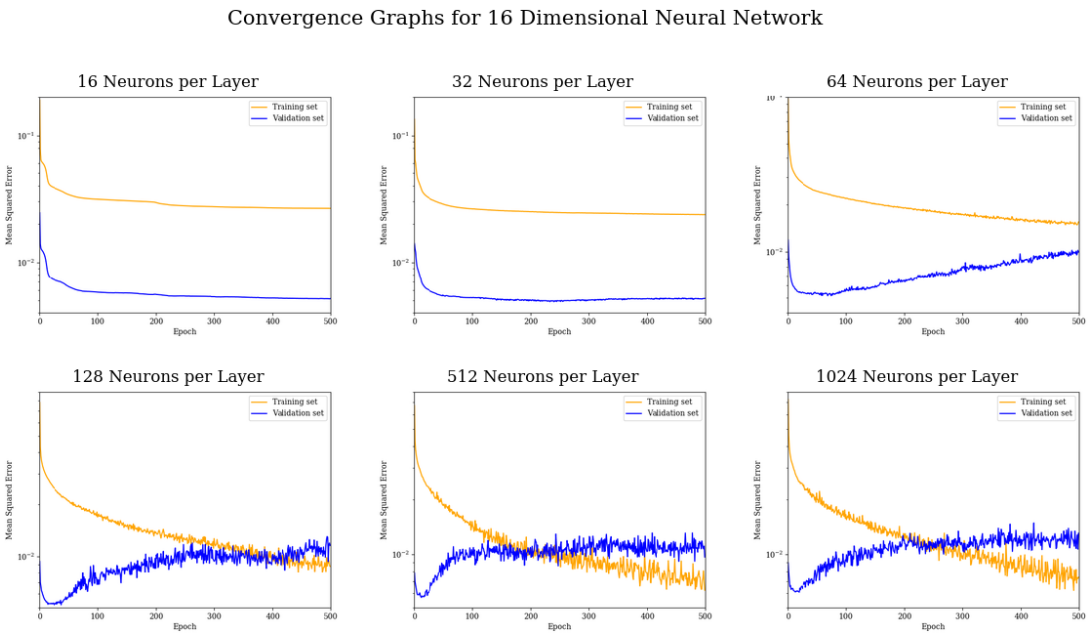


Figure 45: 16 neurone encoding space Neural Network training and validation graphs

Similarly to the validation results of the standard Neural Network, it can be noted that the first three MSE error profiles fail to converge to an appropriate level. Thus these network structures would produce insufficient results if they were to be implemented. From the results of the last three error plots, it is apparent that all these structures produce sufficient prediction results. However, from the display on the Python console [20], the lowest prediction error (both validation and training) for the 16 neurone encoding space Neural Network was produced by the 128 neurone per layer network structure. Therefore this network, in coalescence with the 16 dimensional Autoencoder are to be used for further evaluated in order to select the best performing overall Autoencoder.

Along with the 16 dimensional encoding space; a second Neural Networks structure is to be developed (as specified in table 12), in order to predict the 32 dimensional encoding space produced by the second Autoencoder model. Similarly to the selection process above, and as mentioned in Algorithm 5, the same hyper-parameter selection process was repeated for the 32 neurone output space network. The results of this evaluation can be shown in figure 46 below:

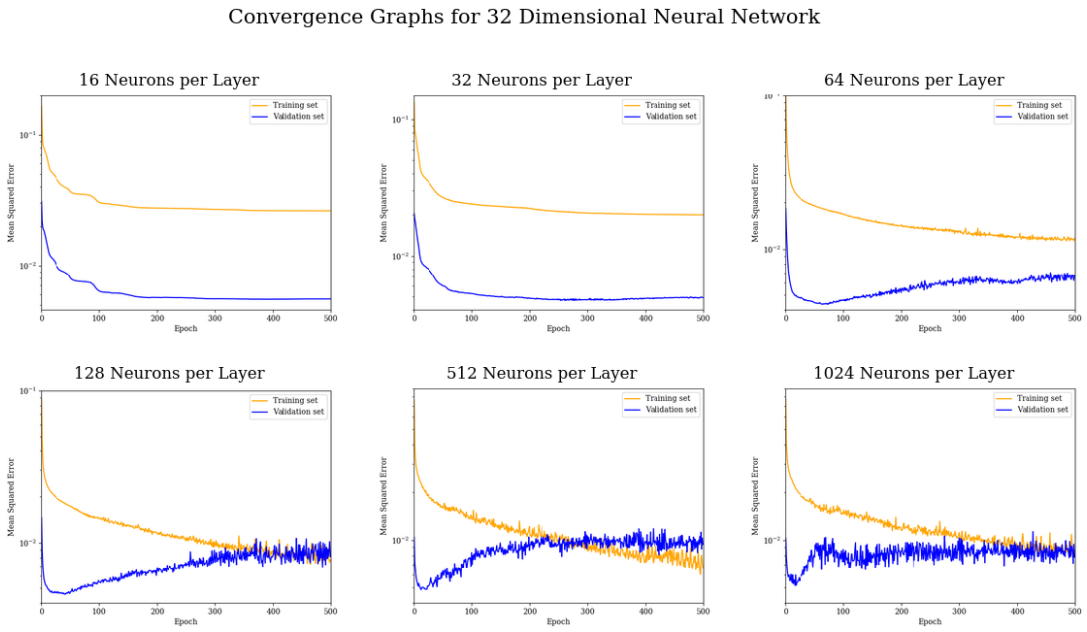


Figure 46: 32 neurone encoding space Neural Network training and validation graphs

Based on the results displayed above, similar conclusions regarding the error convergence of the 16 dimensional Neural Network can be deduced about the 32 dimensional network. From the detailed performance display on the Python console [20], it was deduced that the 32 dimensional Neural Network structure with the lowest overall testing and validation error was the 512 neurone per layer model. Thus in order to determine the optimal Autoencoder and Neural Network combination, the performance of these networks were to be further evaluated.

In order to accurately evaluate the performance of both the 16 and 32 dimensional Autoencoder structures, the resulting temperature and velocity CFD prediction contours for both network types were to be analysed and compared. Using the same process specified in figure 12, and the network combinations identified in the rigorous selection process above, the resulting decoded mean prediction result for both networks could be acquired. By splitting the 10000 dimension prediction vector into separate components, the temperature plots for both Autoencoder structures could be produced as shown:

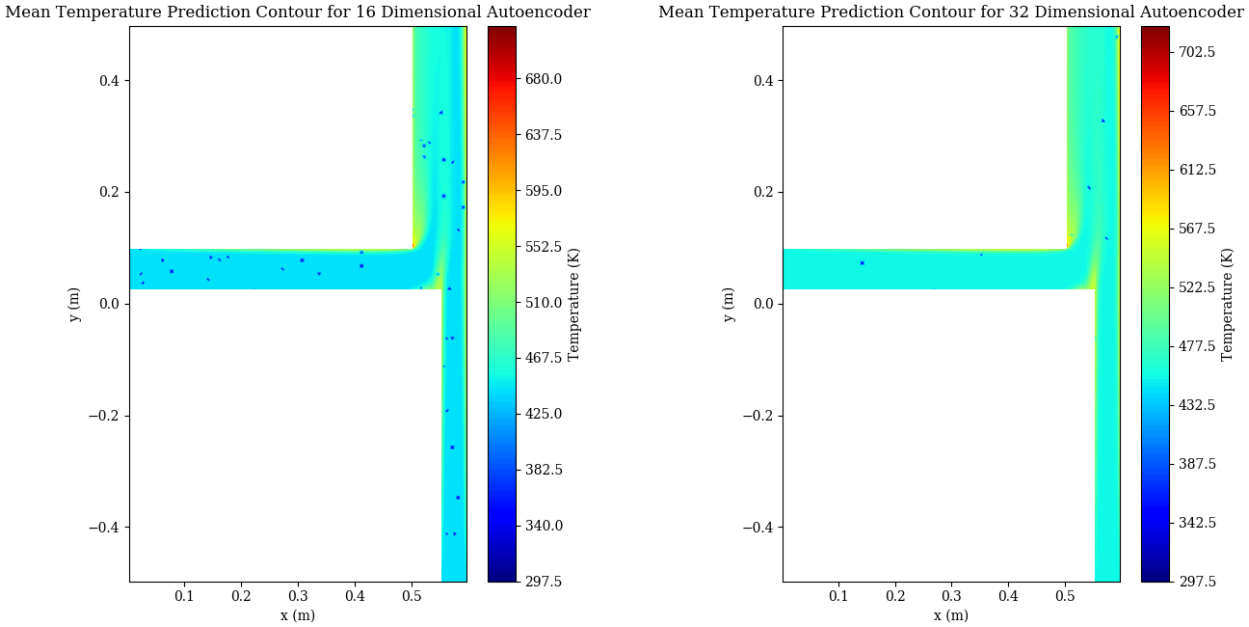


Figure 47: Autoencoder CFD temperature prediction contour comparison

From the comparison of both mean temperature prediction contours, it can be seen that the prediction accuracy of the 32 dimensional Autoencoder seemed to generalise better on both the testing and validation datasets. This is evident as less regions of high uncertainty (as seen by the irregular markings on the contour) are present in the mean prediction of the larger of the two networks.

Similarly to the temperature data plots, the mean velocity prediction contours of the Autoencoder can be seen below:

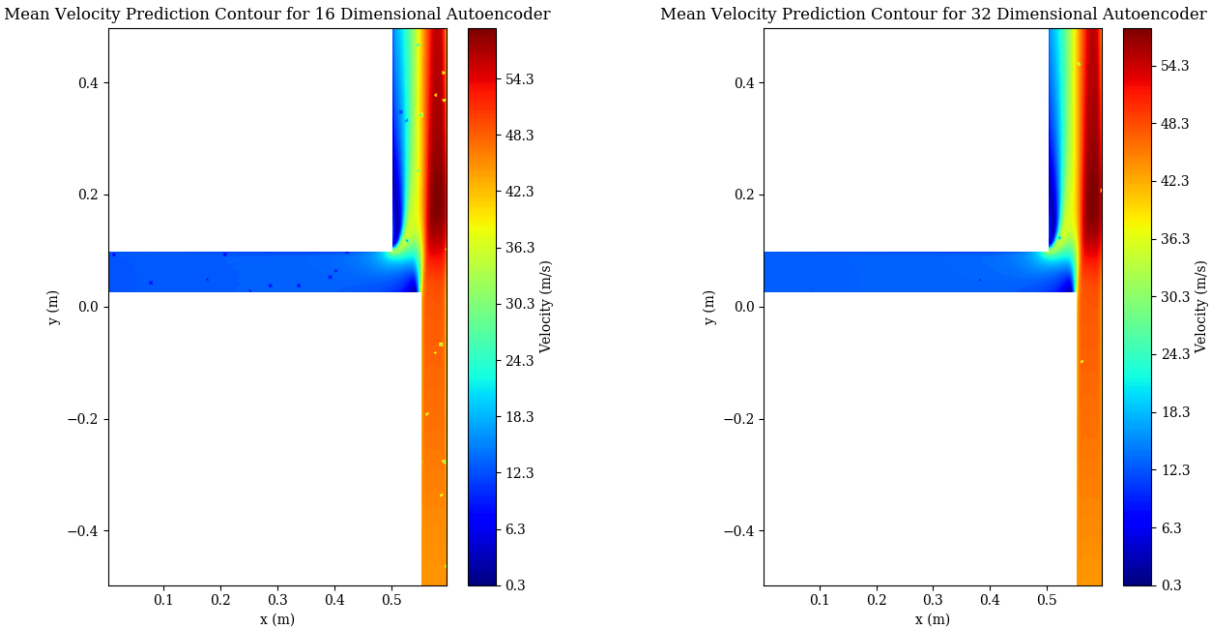


Figure 48: Autoencoder CFD velocity prediction contour comparison

For the same reasons as the temperature predictions in figure 47 it can be noted that the 32 dimensional Autoencoder also produced better velocity prediction results.

Based on the algorithm selection process, it was found that the best performing Autoencoder network consisted of a 512 neurone per layer Neural Network, coupled with the decoder of a 32 dimensional Autoencoder (with the associated Dropout and Batch Normalisation). Thus this Autoencoder network is to be further evaluated and compared against the standard Neural Network and Random Forest, in order to determine which of the evaluated ML models perform best on thee associated CFD prediction data.

Appendix IV - Detailed Model Analysis

Based on the recorded prediction statistics and observations (*see Results and Discussion*), various reoccurring similarities and characteristics were noted amounts the ML model predictions. These phenomena can be further understood by re-representing the available CFD and prediction data in alternative formats. The sections below further detail and discuss the key observations relating to the machine learning algorithm predictions.

Turbulence Prediction Results

From the ML model prediction and error contours, it was evident that all the velocity reconstructions displayed the highest error at the interior of the main pipe section nearest the secondary inlet. This stark correlation (amongst all three models) implies that an underlying physical parameter resulted in these high localised errors. By inspection of the pipe design and CFD flow patterns, it could be deduced that these errors arose due to the high uncertainty/unpredictability of the turbulent flow separation that occurs in these regions. These localised turbulence fields can be identified by recording the turbulence kinetic energy, and the turbulence dissipation rate distributions across the surface of the T-piece design. By recording the values from the original CFD simulation, the turbulence fields for the given instance data can be shown below:

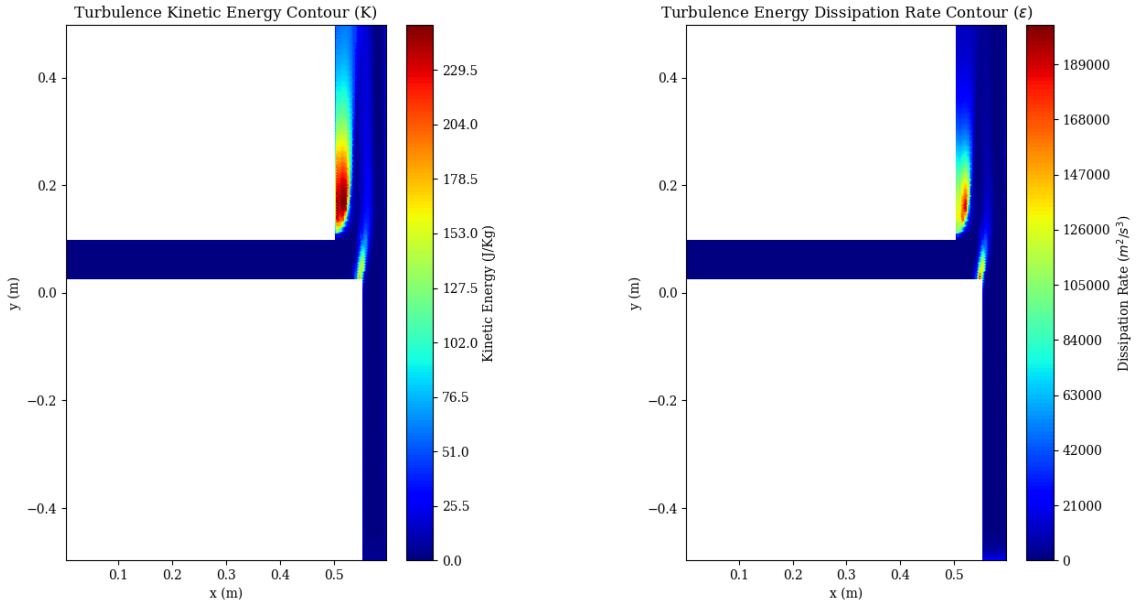


Figure 49: T-piece turbulence kinetic energy (K) and dissipation rate (ε) distribution

By comparing the regions of high turbulence with that of the velocity error prediction contours (*see Results and Discussion*), these two trends can be clearly correlated. Due to the high uncertainty associated with these turbulent zones the ML models struggle to accurately generalise the flow characteristics at these regions. This explains the high exceptionally high errors and unreliable visual contour inconsistencies.

Standard turbulence models (such as the K-Epsilon fields depicted above) delineate characteristics relating to the highest levels of turbulence energy cascades [36]. To further detail the underlying finite/low-level mixing phenomena (and subsequently potential causes for algorithm prediction uncertainty) other turbulence metrics can be considered. One such fluid property is the turbulence Reynolds number. Similarly to the standard Reynolds number, this property can be used to gauge the extent of turbulence within a fluid body. This measure of turbulence is typically used in mixing scenarios (such as T-pieces), in order to determine the effectiveness of the mixing process. The turbulence Reynolds number can be characterised by the following formula:

$$Re_t = \frac{\rho K^2}{\mu\,\varepsilon} \tag{15}$$

Where:
Re_t = Turbulence Reynolds number

Generally a Reynolds values greater than 64 defines whether the flow is characteristically turbulent [37]. By computing this turbulence Reynolds number at each cell of the CFD simulation, the following contour could be produced:

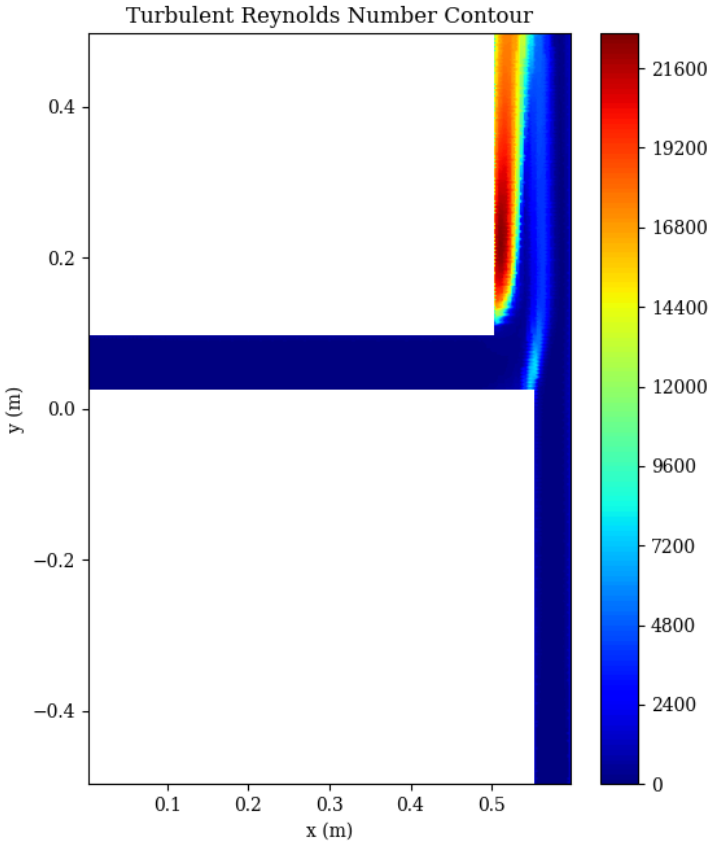


Figure 50: Turbulence Reynolds cell number contour for mixing T-piece model

From figure 50 above, the regions of effective mixing can clearly be identified. As with the K-Epsilon metrics, a correlation between the areas with high Reynolds numbers and the velocity prediction error can be noted. It can also be seen that regions with intermediate level Reynolds values overlay with that of the areas of highest temperature prediction errors. This is most likely due to the uncertainty of the partial thermal mixing which occurs at these interfaces.

Restricted Prediction Contour Resolution

From the prediction performance evaluation of the Autoencoder (*see Results and Discussion*), it is apparent that this model produced relatively high localised reconstruction errors. An adverse result of visualising these prediction inconsistencies, is that the error magnitude diversity throughout the rest if the contour is overshadowed by the variance of the local errors. To overcome this visualisation issue, the colour gradient scheme of the error contour can be restricted. This limit neglects the higher error magnitudes on the contour, allowing for a more detailed representation of the error distribution for the rest of the surface to be presented.

The error plot below shows the mean temperature prediction, and error distribution (as seen in figure 17) with an applied error contour gradient limit of 15 %:

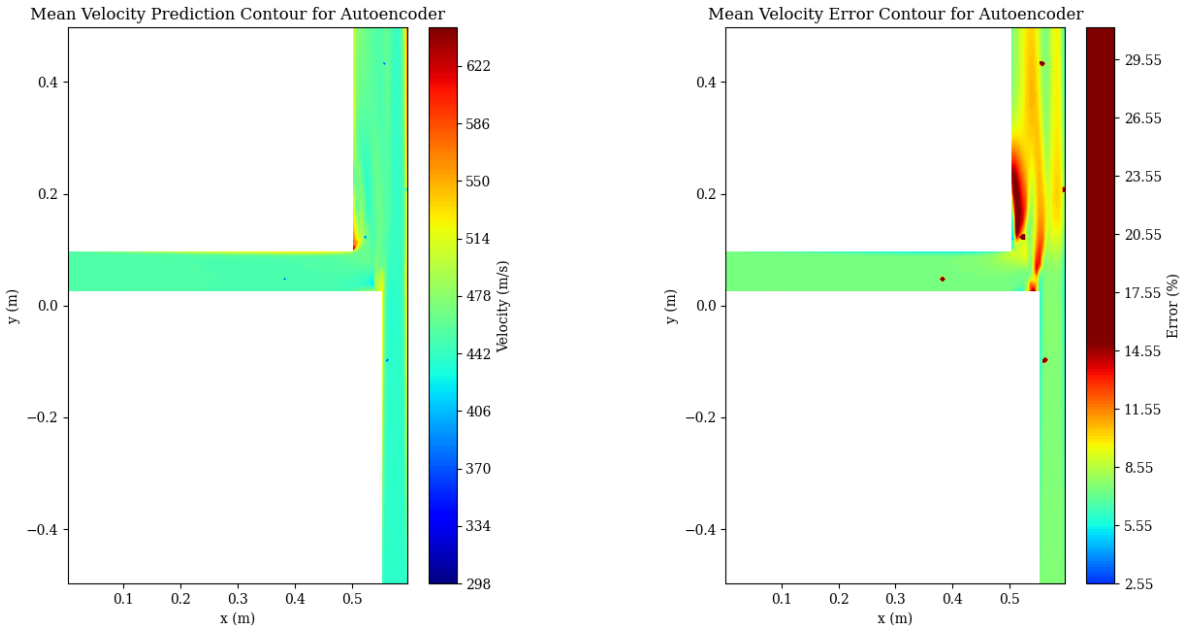


Figure 51: Restricted mean Autoencoder temperature and error prediction contour

It can be seen from the graph above, that by applying the colour bar restriction, the magnitude of the localised high errors can be negated (and subsequently expose the error detail for the rest of the CFD contour). From figure 51 it is apparent that the temperature error distribution shows similar error consistencies to that of the Neural Network and Random Forest, albeit with a slightly higher error margins.

Similarly to the above figure, by applying a gradient limit for all errors greater than 20 %, the following velocity error prediction contour can be produced:

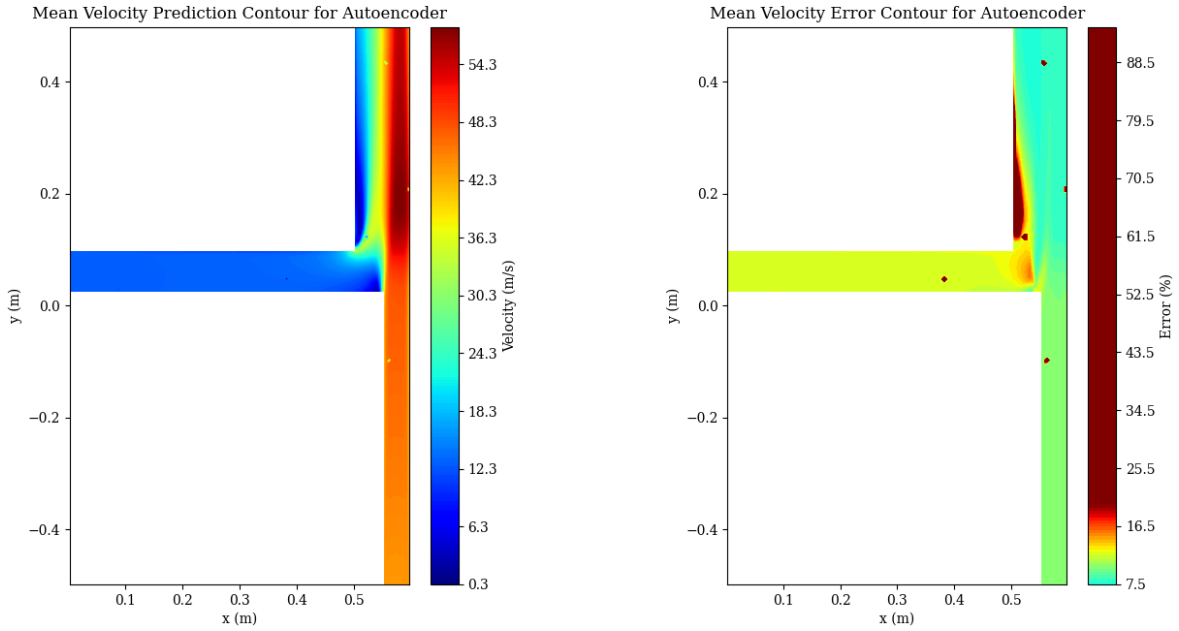


Figure 52: Restricted mean Autoencoder velocity and error prediction contour

From the above velocity contour it can be seen that with the applied limit that the prediction error distribution of the Autoencoder is relatively consistent across the whole CFD surface. By overlaying this error distribution with that of the turbulence contours shown in figures 49 and 50, a correlation between the high prediction errors of the Autoencoder and the high turbulence regions can be noted. This observation indicates that (with the exception of the localised errors) the Autoencoder struggles to generalise on the same turbulent mixing regions as that of the other ML models.

Detailed Prediction Error Analysis

By measuring each cell error for all 1000 CFD data samples, the average and maximum error statistics could be determined. These significant measurements were recorded and displayed in table 3 (*see Results and Discussion*). From these recorded values, various observations in both the prediction contour quality and data characteristics could be correlated. By further plotting a histogram of the cell prediction error results (over the entire dataset), a more detailed mapping of the prediction error distribution could be developed[24].

The following graph illustrates the logarithmically scaled error frequencies for the Neural Network's temperature and velocity cell magnitude predictions:

[24]Note: the colour scheme indicated on the histogram represents the error magnitudes visible on the ML contour reconstructions (*see Results and Discussion*). In other words, the frequency of a particular error range depicted on the histogram plots, represents the observed proportions/frequencies visible on the prediction contour maps

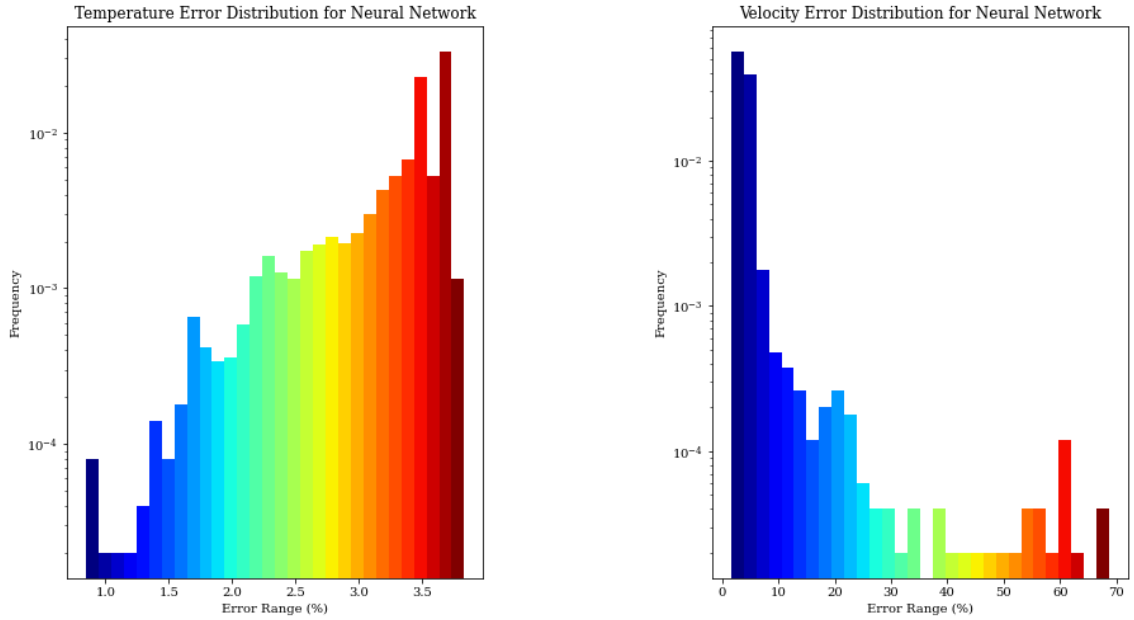


Figure 53: Logarithmic cell prediction error frequency distribution for Neural Network

From the plots above it is evident that the temperature prediction error distribution is relatively invariant, with the majority of the error being skewed to the right of the distribution plot. It can also be noted that due to the stochasticity of turbulent fluid flow, and the vast temperature differences within the T-piece; the resulting prediction error range is far larger than that of the temperature reconstruction error. Thus it can be seen that distributions with a smaller error range (such as the above temperature graph) indicate better overall data generalisation; whereas frequency trends with larger counts of lower-bound errors (as with the velocity plot) produce more accurate contour predictions with less visual inconsistencies.

Repeating the above process for the Random Forest produces the following results:

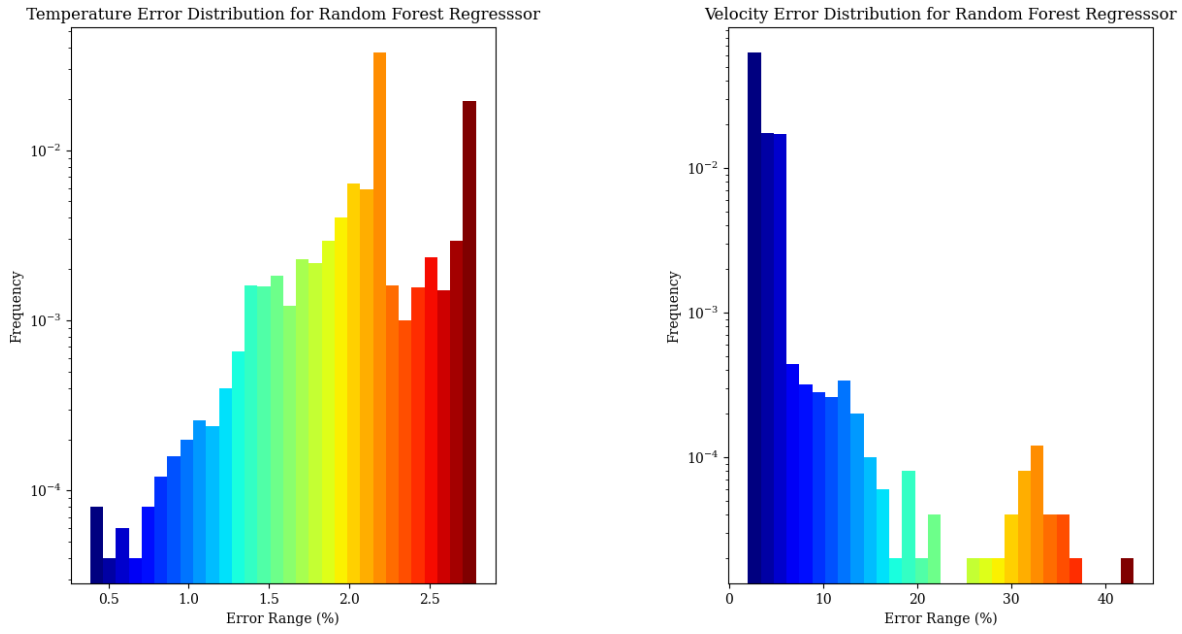


Figure 54: Logarithmic cell prediction error frequency distribution for Random Forest

Although the above trends are similar in structure to that of the Neural Network, it can be noted that the maximum error ranges of the above graphs are significantly lower. This is evident in both the overall error, and visual quality of the Random Forest's contour reconstructions. As this algorithm performed noticeably better when compared to the reproductions made by the Neural Network.

Similarly to the trends noticed in the graphs of the above ML models, correlations between the observed prediction results of the Autoencoder and error frequency trends could be noted from the figure below:

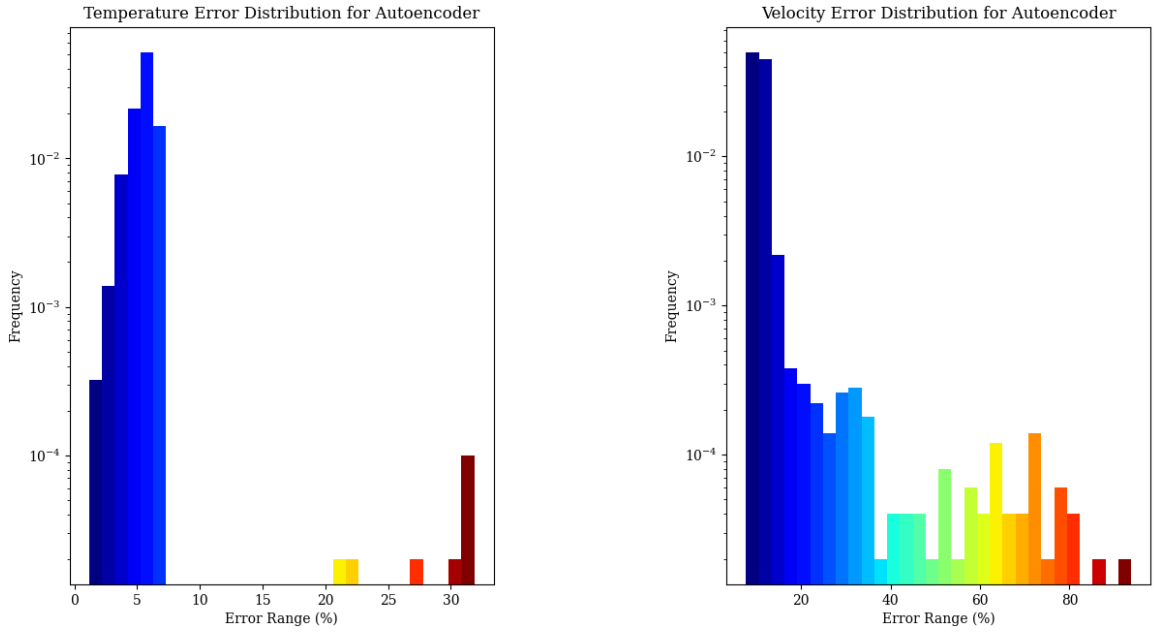


Figure 55: Logarithmic cell prediction error frequency distribution for Autoencoder

By comparing the results of the above temperature distribution, it can clearly be seen that the significantly high localised errors obscure the range of the remaining temperature distribution. This results in a more imprecise prediction result, and subsequently a poorer quality visual contour reconstruction, when compared to the other ML models. This high error range can also be identified in the Autoencoder's corresponding velocity error frequency distribution.

Lower-bound Prediction Error Analysis

As with the original dataset model error analysis, the same data visualisation process can be performed on the prediction results of the lower-bound instance set. These samples were taken by recording the error of each predicted CFD cell for all 20 design points (from which the lower-bound dataset is comprised).

The following graph depicts this cumulative frequency distribution for the lower-bound predictions of the Neural Network:

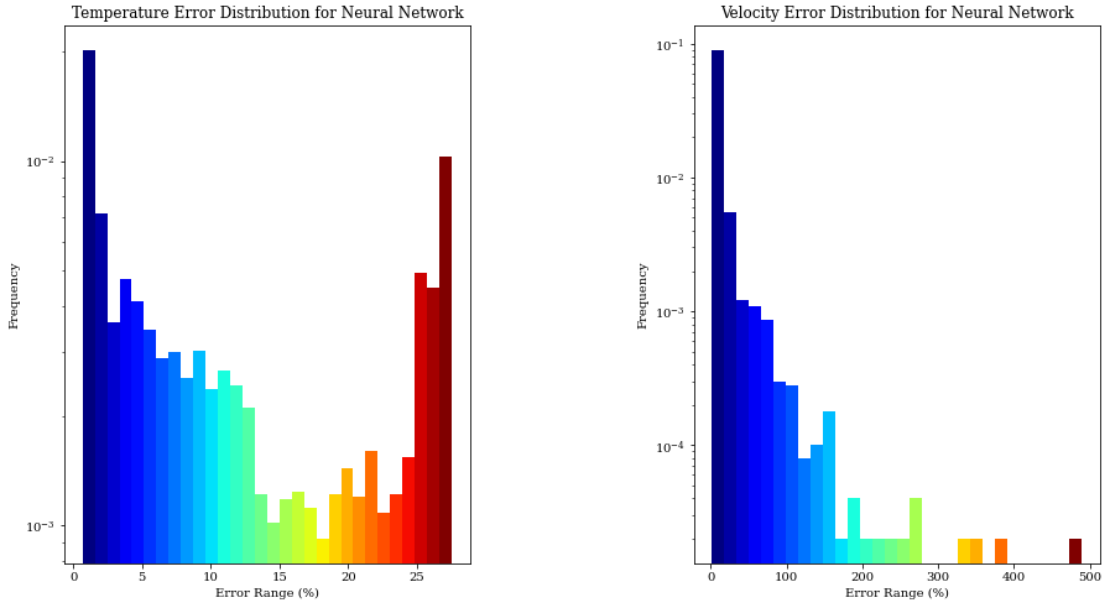


Figure 56: Lower-bound cell prediction error frequency distribution for Neural Network

From the plots above, it can be seen that the lower-bound velocity predictions follow similar trends to that of the Neural Network's velocity forecasts on the original dataset (i.e. the majority of the data lies towards the lower half of the error range, with a frequency that dissipates exponentially as the occurrence of error increases). Contrary to this observation, the temperature error distribution shows higher frequencies correlated at the upper and lower ends of the error distribution range. Moreover, it is evident that this error range is significantly larger than that of the original dataset. This larger range indicates that the Neural Network did not perform as well when reconstructing the lower-bound temperature contour as it did with the training and validation datasets.

The graph below shows the recorded error frequency histograms for the Random Forest's lower-bound data predictions:

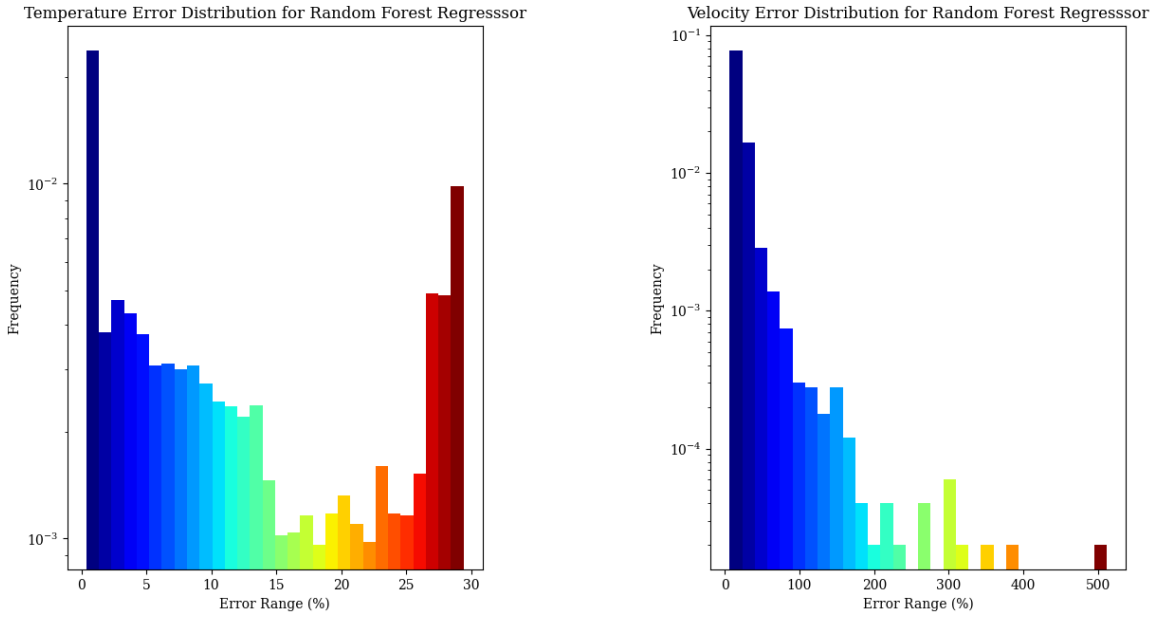


Figure 57: Lower-bound cell prediction error frequency distribution for Random Forest

By comparing the trends of these performance graphs with that of the Neural Network; it can be noted that the Neural Network produced more accurate, and therefore better quality contour reconstructions. This is evident, as the Neural Network shows frequency distributions with higher correlations at lower error ranges when compared to the Random Forest. By overlaying these observations with the reproduced CFD contour reconstructions of the ML models, the Random Forest's poorer prediction performance can be noted.

Repeating the above process for the Autoencoder produces the following results:

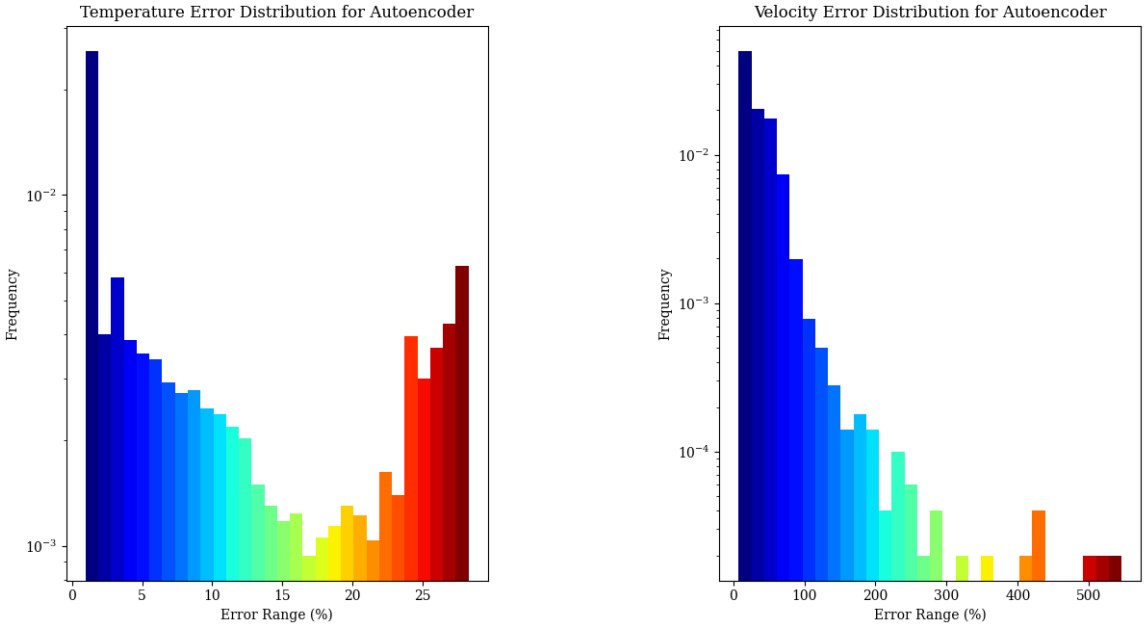


Figure 58: Lower-bound cell prediction error frequency distribution for Autoencoder

As with the Autoencoder's predictions on the training and validation dataset, it can be seen that the Autoencoder produced lower-bound predictions with similar error trends to that of the Random Forest. This result indicates that the Autoencoder was capable of making relatively more generalised predictions on the lower-bound dataset when compared to the Random Forest. Although, it is clear that (as with the other ML algorithms) the Autoencoder did not perform as well on the lower-bound sample set when compared to the performance graphs of the original dataset. This is unsurprising, as all the ML models struggled to fit the more laminar data trends of the lower-bound dataset.

Upper-bound Prediction Error Analysis

By repeating the above data recording processes for the upper-bound instance dataset, a more detailed depiction of the underlying ML outlying data prediction process can be developed. Figure 59 shows the cumulative upper-bound error distribution for the Neural Network:

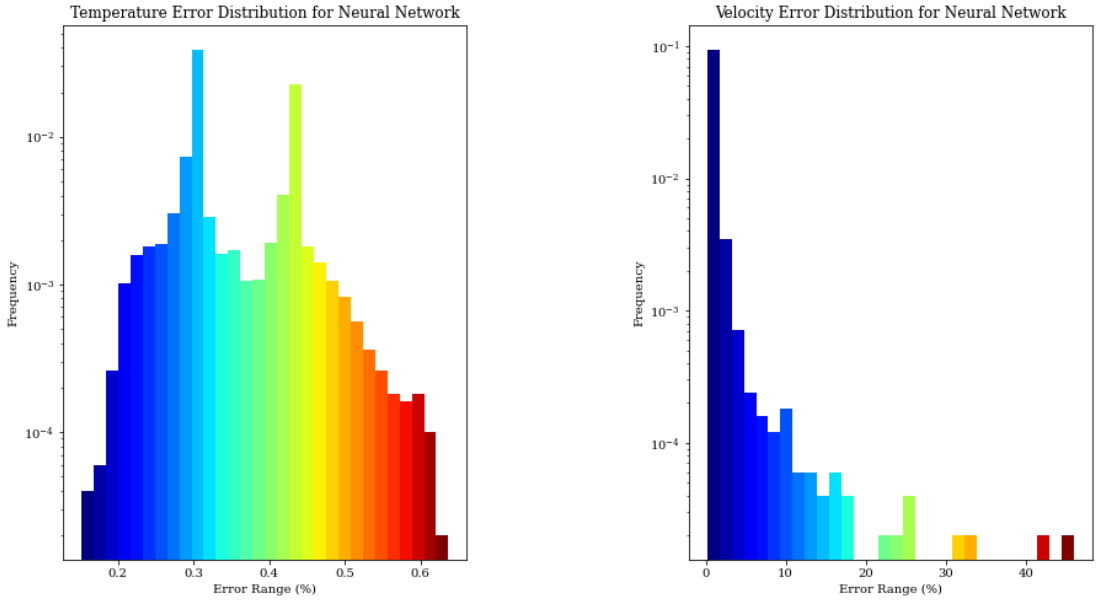


Figure 59: Upper-bound cell prediction error frequency distribution for Neural Network

From the distribution plot above it is evident that the structure of the temperature and velocity distribution more closely resembles that of the original dataset's predictions illustrated in figure 53. It can also be noted from the graphs that the error range is significantly lower than the range associated with the lower-bound prediction set. This is most likely a result of the fluid flow characteristics, as the more turbulent trends of the upper-bound dataset more closely correlate the trends of the original training and validation set. The high accuracy depicted by these plots imply that the recorded upper-bound data, and contour plots are extremely reliable, as they are almost exactly representative of the true CFD solution.

Similarly to the frequency distributions displayed in figure 59 above, the upper-bound prediction error histograms can be shown below:

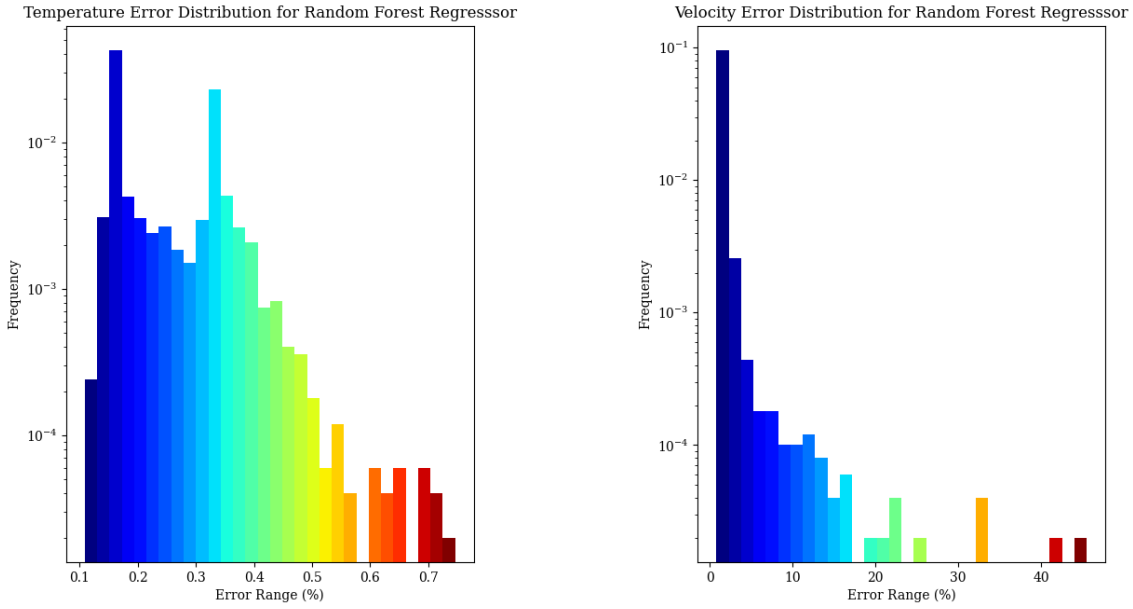


Figure 60: Upper-bound cell prediction error frequency distribution for Random Forest

On observation of the error frequency plot displayed in figure 60, it is apparent that the structure and low domain variation of the Random Forest's upper-bound predictions resemble that of the Neural Network. From further analysis, it is evident that both the precision and accuracy of the recorded error magnitudes is approximately identical to the Neural Network's prediction error distributions. These metrics indicate that the data and contour plots generated by the Random Forest, are far more reliable and representative. This remark is further motivated on observation of the upper-bound predictions and contours of the Random Forest (*see Results and Discussion*).

As with the Neural Network and Random Forest, the frequency distributions for the Autoencoder can be represented as follows:

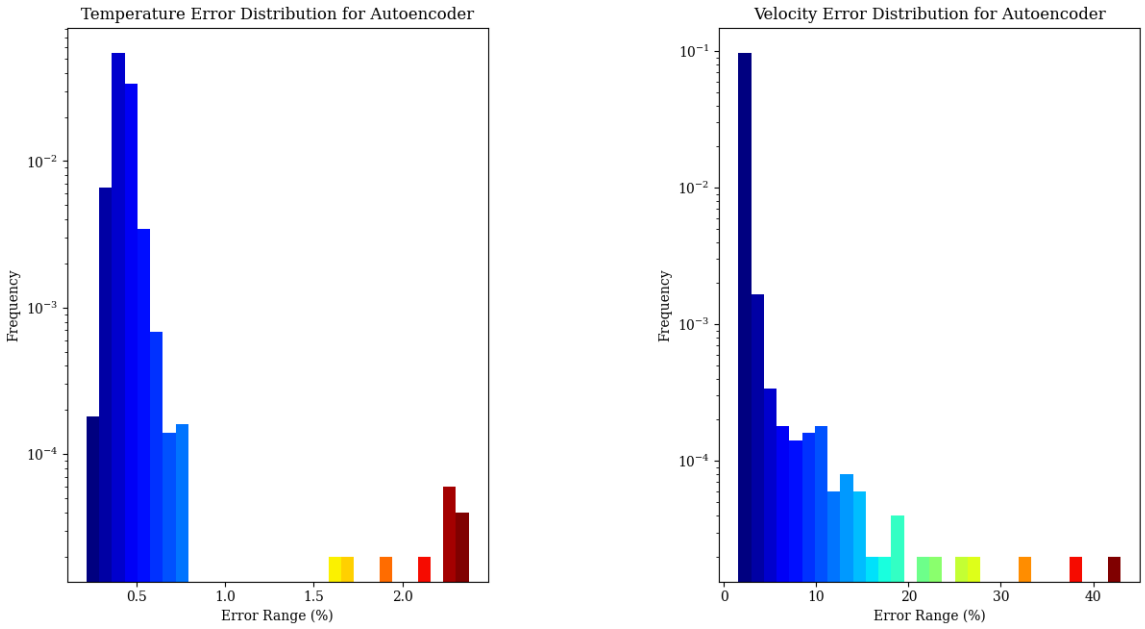


Figure 61: Upper-bound cell prediction error frequency distribution for Autoencoder

From the plots above, it can be seen that the precision of the velocity prediction error range closely resembles the values generated by both the Neural Network and Random Forest. This precision can be further noted when comparing the recorded upper-bound statistics of the ML models in table 9. However, although the Autoencoder produced very precise and reliable velocity data, due to the numerous localised regions of high error (visible on the above frequency distributions), the Autoencoder resultantly produced visually inconsistent upper-bound prediction results. This is particularly apparent on observation of the CFD contour reconstructions (*see Results and Discussion*).

Thus, by detailing the ML model prediction uncertainties, various factors and causes could be identified/noted. It was evident from analysis of the turbulence metrics obtained from the CFD simulations that the validated algorithms generally produced the least reliable data over regions of high turbulence, and fluid mixing. It was also noted that the models performed worse when the system flow characteristics resembled a more laminar regime. Another result of the detailed analyses is that the error frequency distributions indicated that the Neural Network and Random Forests produced the most accurate overall prediction results. This was indicative of the high quality, and reliability of the reconstructed contour graphs developed by the predictions of these models.

Appendix V - Implementation

To appropriately gauge whether the trained networks are capable of being remotely integrated into existing fluid systems, an analysis of the potential methods of implementation are to be explored. By loading the trained model parameters onto python based controller devices (which are commercially available and accessible), the computer modules can be added into already existing infrastructure. These devices can then be operated and monitored remotely (e.g. via a WiFi module extension) to better improve/understand the current operating conditions and safety of the system in question.

In order to measure the application potential of each microcontroller device and ML algorithm, several commonly used python modules were compared based on their flash and RAM performance capabilities[25]. From these computing metrics, the feasibility and computational potential each controller possesses was analysed. The table below shows the computing metrics for a variety of common python based microcontrollers[26]:

Table 13: Python based microcontrollers and corresponding performance metrics

Controller	**Non-Volatile (MB)**	**Volatile (MB)**
Pyboard	1.024	0.192
Raspberry Pi (module 1)	4000	512.0
Raspberry Pi (module 3t)	8000/16000/32000	1000
Raspberry Pi (module 4)	SD card/serial coms	2000/4000/8000
Adafruit Feather Huzzah	4.000	0.096
Micro-Bit	0.256	0.016

Thus from observation of the table above, it can be seen that only the Raspberry Pi (module 4) module is capable of meeting the Random Forest's large storage requirements (granted the external SD storage/communications method is sufficiently capable of supporting the Random Forest's file size). However, this storage issue coincides with observations specified during the memory analysis (*see Results and Discussion*), as the Random Forests large structure prevents these modules from being easily implemented on small scale portable systems.

[25]Note: because each model was trained and compared on a CPU device, the implementation analysis only uses the operating statistics recorded in the original ML model analysis (*see Results and Discussion*). In other words this analysis does not consider the performance advantages the Neural Network and Autoencoder may have if implemented on an external GPU device.

[26]Note: this table is not an exhaustive list of potential/applicable python based hardware.

On further analysis, it can be seen that the Pyboard controller cannot support the non-volatile memory demands of both the Neural Network and Autoencoder. Likewise, due to its limited storage capacity, the Adafruit Feather Huzzah and Micro-Bit devices can also be ruled out as potential computers capable of hosting the operations of these trained ML models.

Thus, from the evaluation above, it is evident that in order to implement the trained CFD prediction algorithms on existing systems and infrastructure, Raspberry Pi (or equivalent device) must be used. The table below shows the approximate number of continuous predictions (i.e. the maximum instance batch size) each model can run before exceeding the of the device limitations[27]:

Table 14: Estimated maximum continuous batch sample size capable of being predicted

Controller	**Neural Network Sample Size**	**Random Forest Sample Size**	**Autoencoder Sample Size**
Raspberry Pi (module 1)	2540	N/A	2650
Raspberry Pi (module 3t)	4660	N/A	4750
Raspberry Pi (module 4)	≥ 5000	≥ 2860	≥ 5000

From the table above, it is clear that the application flexibility of Neural Networks and Autoencoders make them the most pertinent models for in-situ CFD simulation reconstruction. It can also be noted that the Raspberry Pis are capable of hosting several thousand continuous predictions. Furthermore, robust modules such as the Raspberry Pi (module 3t) are specifically designed for industrial applications, further motivating the implementation feasibility of these microcontrollers. Thus from the above analysis, it is evident that the trained CFD prediction models can be easily integrated into current engineering systems. Moreover, these portable models are capable of computing and broadcasting several thousand instances on these microcontroller devices in near-real time.

[27]Note: these estimates were taken form the linear approximations used to interpolate the recordings displayed in figure 20

Appendix VI - Interim Report

UNIVERSITY OF CAPE TOWN

Department of Mechanical Engineering

RONDEBOSCH, CAPE TOWN SOUTH AFRICA

INTERIM REPORT 2020

PROJECT No: 41

Development of a machine learning regression model to predict a simple 2D flow field

Daniel Ferrini [FRRDAN014]

Project Brief

The project sets out to develop a machine learning prediction tool which learns from computational fluid dynamics data to predict flow fields. The benefit of such a tool is in-situ simulation which enables engineers to solve complex physics simulation in near real time, thus creating a digital twin that could be used to monitor and optimise various devices. In the current project a simple 2D mixing domain should be configured in ANSYS Fluent commercial CFD code. The software's built-in features must be used to generate simulation data for a range of boundary conditions. The data should then be wrangled into a data frame which is compatible with a machine learning framework such Keras, Pytorch or Sci-Kit Learn. Using the mixing domain inlet conditions as inputs and some CFD field as outputs the machine learning model must be trained on the data to reduce the prediction error. In other words, the model must be able to be provided boundary conditions and generate a contour map.

Supervisor: Dr Ryno Laubscher

Word Count: 3875

MEC4110W - Project 41

Daniel Ferrini - FRRDAN014

19/06/2020

Plagiarism Declaration

I, Daniel Ferrini, hereby declare that:

i. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.

ii. I have used the IEEE convention for citation and referencing. Each significant contribution to, and quotation in, this report / project from the work(s) of other people has been attributed and has been cited and referenced.

iii. This report/project is my own work.

iv. I have not allowed and will not allow anyone to copy my work with the intention of passing it off as his or her own work.

D.S. Ferrini
19/06/2020

Contents

Nomenclature iii
List of Abbreviations iii
Introduction 1
Problem Statement 2
    Scope 2
    Limitations 3
Literature Review 4
    Conservation Equations and CFD Setup 4
    Machine Learning 7
Project Planning 10
References 12
Appendix 13
    EBE Ethics Form 13
    Risk Identification Form 14
    Impact of Technology Statement 15
    Work Breakdown Structure 16
    Gantt Chart 17

List of Figures

1 Diagram of dimensioned T-piece to be analysed 2
2 Diagram of machine learning categories 7
3 Diagram showing machine learning training profiles 9
4 Simplified flow chart showing general process for machine learning algorithms 11
5 Work Breakdown Structure displaying the scope for machine learning project 16
6 Gantt chart showing machine learning research project lifespan 17

Nomenclature

Greek Letters

α	Predetermined Constant
δ	Kronecker Unit Vector
ε	Dissipation of Turbulent Kinetic Energy
μ	Dynamic Viscosity
μ_t	Coefficient of Eddy Viscosity
ρ	Volumetric Density
σ	Turbulent Prandtl Number
$\bar{\bar{\tau}}$	Viscus Stress Tensor
ω	Dissipation of Turbulent Kinetic Energy

Roman Letters

c_p	Constant Pressure Specific Heat
c_v	Constant Volume Specific Heat
E_{ij}	Rate of Deformation
$\vec{g}$	Gravitational Field Strength
k	Thermal Conductivity
K	Turbulent Kinetic Energy
$\mathcal{L}$	Mean Squared Error
M	Molecular Mass
n	Molar quantity
N	Sample Size
P	Fluid Pressure
R	Gas Constant
t	Time
$\vec{u}$	Velocity Vector
V	Volume
y	Exact Output Value
$\hat{y}$	Hypothesised Output Value

List of Abbreviations

CFD	Computational Fluid Dynamics
ML	Machine Learning
WBS	Work Breakdown Structure

Introduction

The use of computational physics simulations are an essential tooling method that allow engineers to accurately predict solutions for relatively complex and undefined systems. These simulations are advantageous as the data can be visualised and manipulated to provide a more detailed description of the underlying state of the analysed system. However, often these physics simulations are computationally intensive and take significantly long to resolve. As a result these simulations are limited in their application as they cannot be used in situations that require expeditious results.

The intensive temporal requirements that are inherent in many physics simulations, may be overcome through the use of machine learning (ML). This is due to the potential benefits ML algorithms have at simplifying complex data intensive tasks [1]. By training ML algorithms on data that has been obtained from physical simulations an accurate prediction model, capable of replicating the results of the simulation, can be developed. The advantage of this prediction model is that the resulting algorithm can be used for near real-time performance monitoring [2]. This enables engineers to visualise and monitor the operation of complex networks almost immediately after the associated variables are adjusted. The ability to promptly analyse the performance of a system can provide a better understanding for previously unknown *what-if* cases regarding how the system in question may function. Along with real-time visualisation this can be used as an anomaly detection tool; to further understand how the ML algorithm deviates with respect to the simulation from which the training data was sourced.

Machine learning is a powerful computational, as-well as engineering tool that can be used as an aid to optimise design processes, and better enhance methods for safety monitoring and assurance. Due to the low computational demands of ML, these algorithms can easily be implemented/adapted for engineering projects. This simple execution, along with the vast potential range of applications associated with these programs, allow for these learning algorithms to be implemented in parallel with already existing systems.

This interim report details the formulation, approach, and planning of a learning model capable of predicting the flow regime for a simple 2D mixing domain. The selection of such algorithms may be applicable as a proof that ML can be used, in conjunction with computational fluid dynamic (CFD) simulations, as a viable solution to solving complex iterative tasks in near real-time.

Problem Statement

Scope

In order to investigate whether ML algorithms can be used as a form of real-time visualisation, performance motoring, and anomaly detection for CFD simulations; the ML algorithm must be trained on a sufficient dataset. This data will be acquired by analysing a simple T-piece mixing setup using ANSYS Fluent simulation software. The simulation will highlight the mixing behaviour of incompressible air flow. Figure 62 shows the design profile for the two dimensional T-piece mixing domain which is to be analysed[28]:

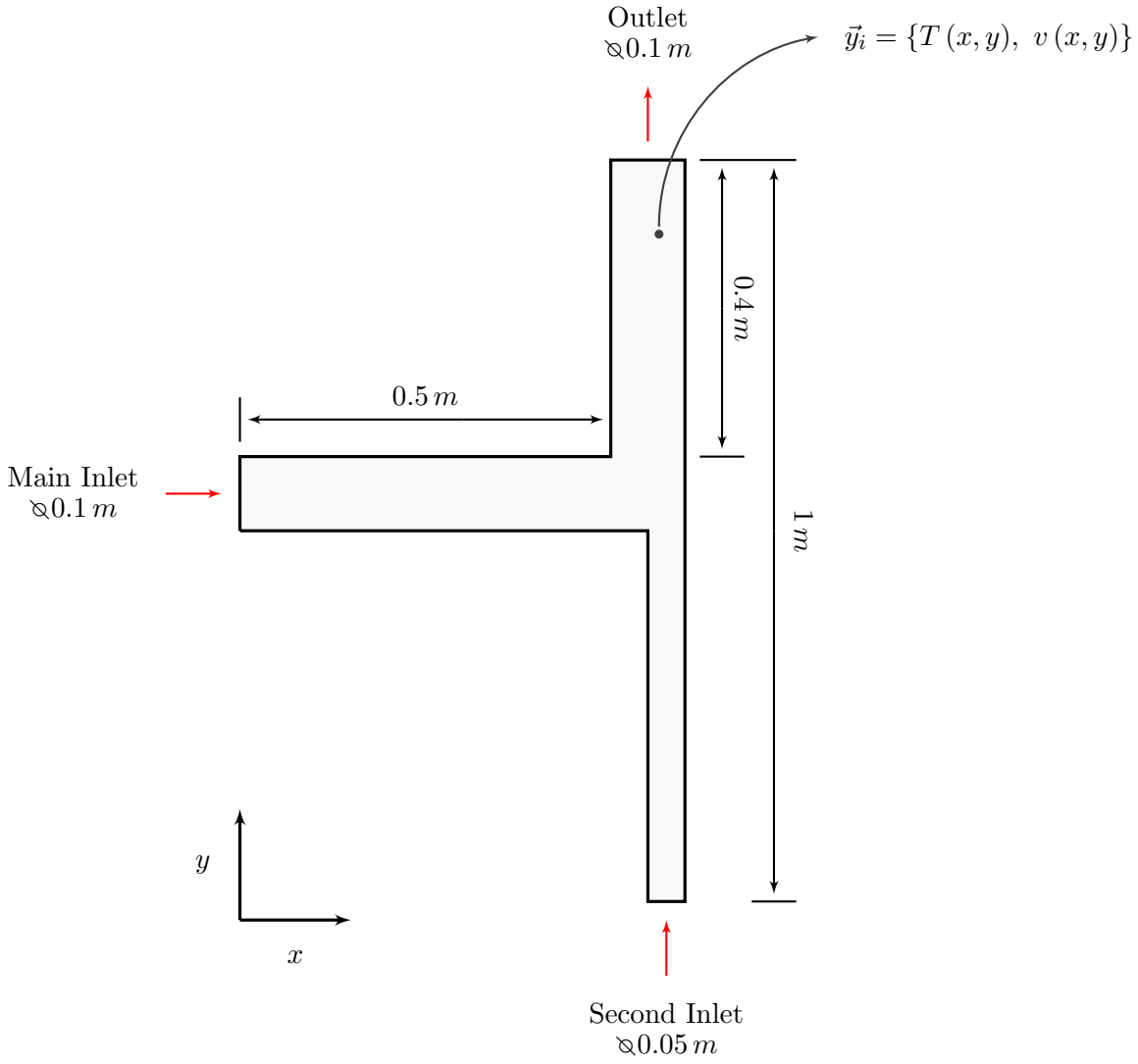


Figure 62: Diagram of dimensioned T-piece to be analysed

As shown in the diagram above, by varying the input velocity and temperature for the main and secondary inlet, an output dataset for the resulting steady state velocities and temperatures at each mesh cell can be obtained. By automating the simulation this dataset can be expanded to provide solutions for a variety of input conditions.

[28] Analysing the CFD simulation in two dimensions allows for a more simplistic flow system scenario. Due to time constraints this simplified setup allows for more data to be acquired over a shorter period, and at a lower computational expense.

The dataset obtained from the computer simulation will be separated (into training, and validation samples), from which an analysis of the data characteristics can be performed. By assessing the CFD data, a detailed approach as to which machine learning algorithms are best suited for this particular regression problem can be formulated. This information is to be wrangled into a data frame that is compatible with Python built machine learning frameworks [29] (Keras; Tensorflow; Sci-Kit Learn; etc...).

Three main regression algorithms will be evaluated (namely: an artificial neural network, a random forest regressor, and an autoencoder). These selected algorithms will be trained and validated using the CFD data. The trained models will be compared by analysing each algorithm's ability to accurately predict solutions to the testing set. The error distribution of these predictions, as well as the model's ability to generalise outlying data will be recorded and compared. Other techniques such as the final runtime, and memory usage will also be used to evaluate these respective algorithms. Based on the observations made by this assessment; conclusions will be developed as to which ML algorithm is best suited for the particular application of predicting CFD simulations, as well as to whether these algorithms show potential to be implemented as a viable method for near real-time data visualisation and anomaly detection.

Limitations

Due to the vast application of computational fluid dynamics, this project will only focus on a simple two dimensional mixing scenario. This is to ensure more time and data can be allocated to the research and development of the appropriate machine learning models. The dimensions of the T-piece along with the physical properties of the mixing fluid will remain constant. This is because this simulation will only emphasise the effects of varying the velocity and temperature parameters, as these are the necessary parameters required for effective mixing. It will also be assumed that the fluids in question are incompressible, this because the governing equations for incompressible fluid motion is generally simpler than that of compressible fluids [3]. Furthermore for simplification reasons, the density properties of the mixing fluid will be defined as an incompressible ideal gas. As this theoretical model ensures the density of the air is only dependent on the operating pressure of the system (rather than the state of each discretised fluid element) [4].

Although machine learning algorithms are capable of performing complex predictions in relatively short periods of time, these algorithms need to be trained on a sufficiently large dataset. This means that the precision and training duration is limited to the size and complexity of the training samples. As a result the algorithm best suited for this particular two dimensional application may not necessarily scale as well as other prediction models for three dimensional problems. This project will also only compare the results of the machine learning predictions to the CFD simulation from which the training data is sourced. Therefore the report will not detail the accuracy these learning algorithms have when compared to realistic experimental observations of T-piece fluid mixing.

[29] By using built-in libraries sufficient project time can be saved, as opposed to building the algorithms from scratch which is more exhaustive.

Literature Review

Conservation Equations and CFD Setup

In order to properly simulate a realistic fluid flow environment, the dynamics of the fluid must be understood on a finite scale. This can be achieved by analysing the conservation laws of the fluid. The Navier-Stokes equations are principal formulas used to resolve these conservation laws, as they characterise the physical parameters that influence the behaviour of the fluid [5].

For incompressible flow regimes the mass of the fluid entering the system is equivalent to the outlet mass. Assuming the volumetric density of the fluid is constant throughout the control volume, the general mass conservation equation (for a differential 3D flow element) can be described by the following equation [6]:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}\,) = 0 \tag{16}$$

Where:
ρ = Fluid density $[kg/m^3]$
$\vec{u}$ = Fluid velocity vector $[m/s^3]$
t = Time $[s]$
∇ = Gradient operator $[\frac{\partial}{\partial x_i}\hat{i} + \frac{\partial}{\partial x_j}\hat{j} + \frac{\partial}{\partial x_k}\hat{k}]$

This formula accounts for the change in density of the fluid over time (zero for incompressible fluids), and the rate at which mass is entering and exiting the system. It is evident from the equation that the change in mass of the system is equivalent to zero. This implies that the total inlet and outlet flow rates must be equal.

Similarly to the mass conservation of the control volume, the total momentum of the system must be conserved in order to accurately replicate natural observations. The conservation of momentum, for an incompressible Newtonian fluid, can be described by the formula:

$$\rho \left(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} \right) = \rho \vec{g} - \nabla P + \mu \nabla^2 \vec{u} \tag{17}$$

Where:
$\vec{g}$ = Gravitational field strength $[m/s^2]$
P = Fluid pressure $[Pa]$
μ = Dynamic viscosity $[N\,s/m^2]$

This formula relates the product of the mass flow rate and velocity with the external forces (gravitational, differential pressure, and viscus shear) acting on the fluid element. When analysing a flow system the internal finite elements of the control volume must obey the above conservation equation. This is necessary to correctly determine the major, and minor loss factors that characterise the flow dynamics [3].

The third conservation equation, required to replicate natural fluid flow, is the law of conservation of energy. Assuming the finite element being analysed is incompressible (i.e. the element volume is constant) and that the conductive heat transfer obeys Fourier's law [7], the equation for the conservation of energy can be represented as follows:

$$\rho c_v \left(\frac{\partial T}{\partial t} + \vec{u} \cdot \nabla T \right) = -P \nabla \vec{u} + k \nabla^2 T + \bar{\bar{\tau}}\, \nabla \vec{u} \tag{18}$$

Where:
c_v = Constant volume specific heat capacity $[J/kg\,K]$
T = Fluid temperature $[K]$
k = Thermal heat conductivity $[W/m\,K]$
$\bar{\bar{\tau}}$ = Viscus stress tensor $[Pa]$

This equation relates the total thermal energy of the fluid (the sum of the enthalpy and internal energy) to externally introduced energy terms (i.e. the boundary work due to the pressure, thermal conductivity, and shear stress). The individual metrics of the viscus stress tensor can further be represented in the following relationship:

$$\tau_{ij} = \mu \left(\frac{\partial \vec{u}_i}{\partial x_j} + \frac{\partial \vec{u}_j}{\partial x_i} - \frac{2}{3} \frac{\partial \vec{u_k}}{\partial x_k} \delta_{ij} \right)$$

Where:
δ = Kronecker unit vector $\left[\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \right]$

The Kronecker unit vector is a term that arises from Stokes' hypothesis [7]. This includes an additional shear term along the main diagonal of the viscus stress tensor.

These conservation equations can be used to accurately model naturally observed fluid flow scenarios, and are generally the governing equations used when modelling finite CFD systems. The Navier-Stokes equations are also particularly useful for computational simulations, as they allow for accurate modelling of circulation regions and flow separations [7]. Thus due to this added characteristic, Navier-Stokes equations can also be used to approximate turbulent fluid flow. There are several turbulent modelling options that are applicable for the development of an accurate simulation model. The two most common partially analytical models are the K-Omega, and K-Epsilon methods [4].

Predominantly the K-Omega model is best suited for predicting turbulent flow fields near the walls and boundaries of control volumes, whereas the K-Epsilon model produces more accurate predictions far from boundaries (i.e. within the fluid body) [8]. These turbulent models can be combined using the Menter's Shear Stress Transport (SST) model. However this equation requires adjustments, which are time consuming and beyond the scope of the project. As a result the K-Epsilon model is to be used in the simulation environment (as this model provides a more accurate description of the flow characteristics at the mixing interface, and is typically used to model pipe flow systems) [8].

$$\text{K-Epsilon} \quad = \quad \begin{cases} \frac{\partial(\rho K)}{\partial t} + \frac{\partial(\rho K \vec{u}_i)}{\partial x_i} = \frac{\partial}{\partial x_j} \left[\frac{\mu_t}{\sigma_K} \frac{\partial K}{\partial x_j} \right] + 2\mu_t E_{ij} E_{ij} - \rho\varepsilon \\ \frac{\partial(\rho\varepsilon)}{\partial t} + \frac{\partial(\rho\varepsilon \vec{u}_i)}{\partial x_i} = \frac{\partial}{\partial x_j} \left[\frac{\mu_t}{\sigma_\varepsilon} \frac{\partial \varepsilon}{\partial x_j} \right] + 2\mu_t \alpha \frac{\varepsilon}{K} E_{ij} E_{ij} - \rho\beta \frac{\varepsilon^2}{K} \end{cases} \tag{19}$$

Where:
K = Turbulent kinetic energy $[J]$
ω, ε = Dissipation of turbulent kinetic energy $[J/s]$
α, β = Adjustable constants [30]
σ = Turbulent Prandtl numbers
μ_t = Coefficient of eddy viscosity $[m^2/s]$
E_{ij} = Rate of deformation $[s^{-1}]$

[30]Note these constants have been derived through experimental observation, and iterative computational processes relating to the dynamics of air [4].

The interface of the wall boundary and fully developed fluid stream can be subdivided into two domains (namely a viscous sublayer and turbulent region). Noting that the K-Epsilon model emphasises the mixing behaviour within the fluid body, it is imperative that the input parameters to this turbulence model do not allocate meshing resources when resolving the viscous sublayer at the boundary [4]. Thus, by applying the empirically derived standard wall functions as a near wall treatment the logarithmic behaviour associated with the viscous sublayer can be approximated using fewer discrete mesh cells.

When modelling a fluid at a finite level, the mass of the fluid can be described in terms of its volumetric density. This method is beneficial as the density can be described by the physical properties of the fluid. Knowing that the mixing fluid to be simulated is incompressible air, the properties and behaviour of the gas can be conveniently generalised by the Ideal-gas law as follows:

$$PV = nRT$$

Where:
V = Volume of gas $[m^3]$
n = Molar quantity of gas $[mol]$
R = Universal gas constant $[8.314\, J/\, Kmol]$

The density of the incompressible ideal gas can be represented by rearranging the above equation in terms of the substance's molar mass:

$$\rho = \frac{PM}{RT} \tag{20}$$

Where:
M = Molecular mass of substance $[kg/mol]$

This formula simplifies the computation of the entire control volume, as the result is independent of local relative pressures that act on the fluid [4]. This means that the density of the fluid element can be determined from the operating pressure of the system, rather than computing the pressures associated with each mesh cell. Thus the volumetric density determined from this ideal-gas law can be substituted into the conservation equations, so as to accurately predict the dynamic behaviour of the fluid.

To correctly predict the heat flow within the simulated gas, an accurate representation of the material's specific heat is required. For incompressible fluids the specific heat at constant volume can be approximated by the specific heat at constant pressure (i.e. $c_v \approx c_p$) [7]. This means that governing equations that are dependent on the principals of heat transfer such as equation (3) can be expressed in terms of the specific heat at constant pressure. By representing the specific heat as a temperature dependent piecewise-polynomial function, a more accurate expression for this heat capacity variable can be determined. The general form for this piecewise-equation can be described below:

$$c_p(T) = \begin{cases} A + BT + CT^3 + DT^4 + ... & a \leq T < b \\ E + FT + GT^3 + HT^4 + ... & b \leq T < c \end{cases} \tag{21}$$

Where:
A, B, C, D, E, F, G, H = Predetermined coefficients[31]
a, b, c = Specified temperature boundaries $[K]$

[31]Similarly to the constants of the turbulent modelling equations, the temperature range, constants, and coefficients of the piecewise-polynomials are empirically determined for computational accuracy [4].

Machine Learning

Machine learning algorithms are programs that are capable of learning from experience. This experience is acquired by measuring the algorithm's performance with respect to a particular task. Thus machine learning is defined as a program that is capable of improving its prediction performance when exposed to a given task [10]. Machine learning models can broadly be categorised as either supervised, semi-supervised, and unsupervised; depending on the nature of the problem and the amount of supervision the algorithms require [11]. The type of model best suited for a particular task can be determined by the availability of training data and type of solution the final algorithm is required to output. These categorised algorithms can therefore be further subdivided as a result of the learning information available [12]. Figure 63 shows the types of subdivisions applicable to different machine learning models.

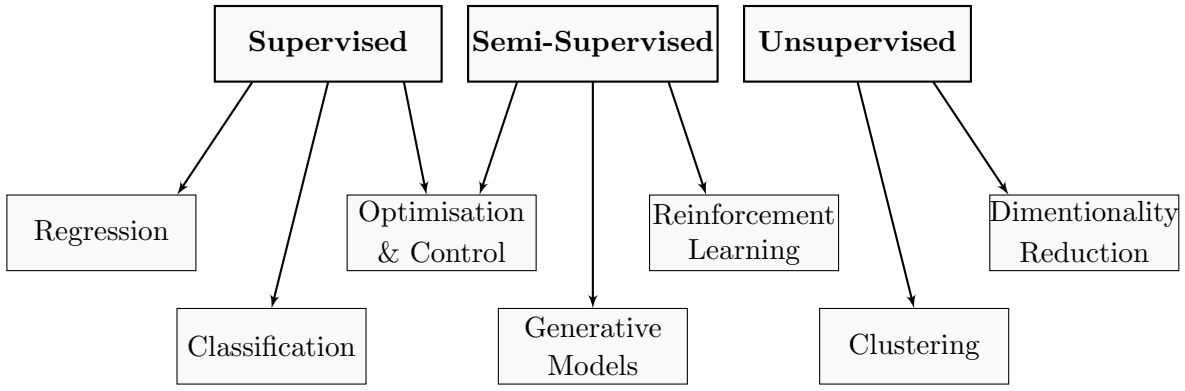


Figure 63: Diagram of machine learning categories

In the context of the mixing T-piece, the ML algorithm is expected to predict the flow regime of the system from the given input characteristics (i.e. CFD boundary equations). The performance of this estimated output is to be compared to the results of the CFD simulation with the same specified input conditions. It is evident from the structure of this particular nodus that the required ML program is a supervised learning task. This is evident as there is an available unique solution set (known as labels) from which the algorithmic approximation can be validated [11]. More specifically this problem is an example of a regression task as the program is expected to predict a numerical target value [11]. These prediction are in the form of temperature and velocity magnitudes measured at the nodes of each cell in the CFD mesh.

Machine learning algorithms work by computing a series of given training instances (these features are typically denoted by the variable X). The model then predicts the corresponding output for each given input instance (these values are represented by the symbol Y)[32]. In a supervised machine learning scenario, the error of this hypothesised prediction is compared to the original output data [11]. The most prominent way of computing the prediction error of a machine learning algorithm is the mean squared error (MSE) cost function [13]. This error/loss function can be represented by equation (**??**) as such:

[32]For models that are required to solve multi-dimensional problems these input and output parameters are represented in the form of matrices $\bar{X}$ and $\bar{Y}$ respectively.

$$\mathcal{L}(\hat{y}, y) = \frac{1}{N}\sum_{k=1}^{N}(\hat{y} - y)^2 \tag{22}$$

Where:
$\mathcal{L}(\hat{y}, y)$ = Mean squared error function
N = Number of samples in output set
$\hat{y}$ = Hypothesised output value
y = Exact output value

This cost function was derived from the formula for the standard deviation/root mean square error (RMSE) of a dataset. The MSE is typically used over the RMSE, as it is less computationally intensive to minimise whilst still maintaining its analytical decorum [11]. Although the MSE is the most common error analysis tool, other methods such as the mean absolute error (MAE) may be more appropriate in solving data problems with many outliers. This is because the RMSE and subsequent MSE is more sensitive to outliers [11].

Depending on the type of algorithm used; the measured prediction error is then evaluated and the machine learning function modified so as to minimise the error. By continuous correcting and minimising the prediction error, the algorithm develops an approximate function that best fits the training data. This correlated function can then be passed new unseen input variables and be able to accurately map these instances to the corresponding output variables [10]. These algorithms are particularly useful as they can be used to correlate complex linear, polynomial, and logarithmic data profiles. Typically the training of such algorithms can be categorised as either online or batched learning.

Online learning is useful in environments that demand regular adjustments due to the constant fluctuation of data, and when the available dataset is too large to be stored in the memory of the host virtual machine (known as out-of-core training) [11]. This learning system is beneficial in applications that do not have a readily supply of training data (such as reinforcement learning) [11]. This learning process works by training off of small groups of data known as mini-batches.

Batched learning systems are explicitly trained and validated offline. For relatively small datasets (less than 500 000 entries) it is conventional practice to separate the training and validation data in accordance with Pareto's' principle (i.e. a training-testing ration of 80 % : 20 %) [11]. Batched learning is generally time consuming, and requires a sufficient source of training data. However once these algorithms are trained, they can be implemented easily and with relatively low computational intensity [11]. Thus for the application of real-time flow analysis this form of training is most applicable, due to the available source of CFD data and the numerical efficiency of the resulting function [14].

Given the correct ML model and a sufficiently sampled training set, these algorithms are capable of predicting the trends of seemingly complex data distributions. However, due to the natural noise and variance that occurs in data sampling, oversimplified and unregularised ML models are not realistically useful as functional algorithms. This is because they are prone to underfitting and overfitting [10]. As a result these algorithms are incapable of generalising the trends of new instance data as shown in figure 64 over page:

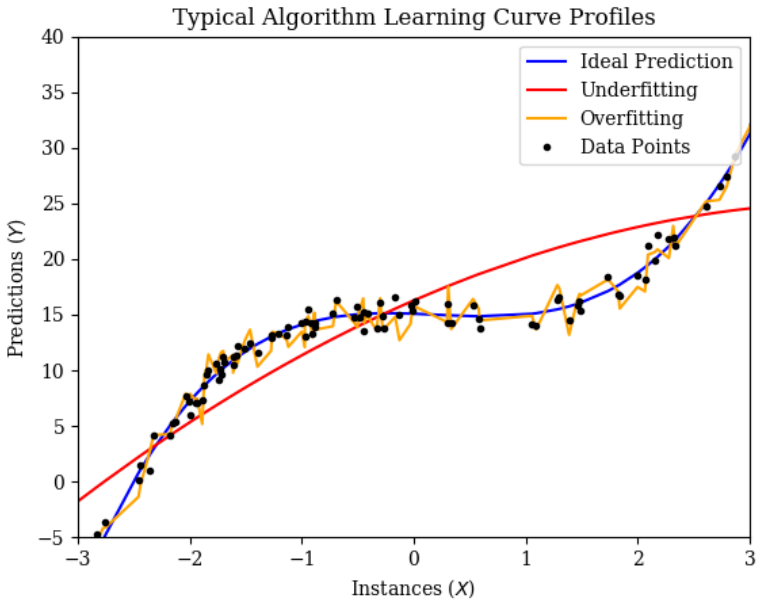


Figure 64: Diagram showing machine learning training profiles

It is evident from the graph above that if the algorithm is too simplistic the function has a tendency to underfit the training data. Typically this occurs when the machine learning model is highly regularised. This regularisation can occur when factors such as the model hyper-parameters are overly restricted, or when there is an insufficient number of features from which the ML function is comprised [10]. It can be seen in figure 64 that the underfitting curve is incapable of matching the higher degree distribution line which the data points correlate to. Underfitting can be overcome by using a complex ML model with more parameters that can be modified, reducing the effect of regularisation hyper-parameters, and by training the algorithm using input features that have a better correlations to the data profile [11].

Conversely to phenomena of underfitting; overfitting occurs when the approximation function generated by the ML algorithm over-compliments the training data distribution. This means that the model generalises particularly well on the given training set, but is incapable of accurately approximating new instances [10]. This is a result of the ML program mapping the output data to a higher degree than the original dataset. Overfitting can be observed when the prediction error of the training set converges to a relatively low value, whilst the validation error increases (indicating the model is struggling to predict the output values of the testing dataset). The effects of overfitting can be reduced by limiting the complexity of the ML model, adding regularisation parameters to inhibit the algorithm's sensitivity to outliers and variable data, reduce noise during the sampling of data, use larger datasets, and reduce the dimensionality of the data that the algorithm must predict [11].

In order to produce a very accurate model that is capable of matching complex data profiles whilst still generalising well for new instances, a trade-off between the model's complexity and regularisation must be made. This is known as the bias/variance trade-off [11]. The bias refers to the incorrect assumption of the functions complexity. This implies that the complexity if the algorithm is underestimated (producing a simpler and more restricted model). Variance refers to the function's tendency to overfit the data, as it results in a greater sensitivity to outlying data [10]. By balancing the extent of this trade-off ratio, an ideal structure for the model's complexity can be produced.

Project Planning

Before any process in the project could commence the implications of undertaking the task were to be considered. This was achieved by assessing the risks and ethical implications of the project scope. These evaluations were documented in the EBE ethics form, and Risk identification form (*see Appendix*). Along with these forms an assessment into the resulting technological impact this research project may cause was conducted. This process was essential for understanding the future implications the findings of this research would have (*see Appendix*).

To ensure the components of the project scope are satisfied before the required deadlines, appropriate planning is required. This challenge can be overcome by decomposing the project scope into manageable sections. Each section is correlated to a particular project milestone and submission deadline. By visually representing the relationships between each element in the project scope, a work breakdown structure (WBS) could be constructed. The WBS for this specific project (*see Appendix*) displays each submission milestone as an individual branch in the overall network.

Theoretically once each deliverable in the WBS is complete the milestones, and subsequent project requirements would be fulfilled. However, not only do the components of the project need to be complete, these requirements must be accomplished within the provided deadlines. Conventionally this is done through the application of a Gantt chart. The Gantt chart can be constructed by assigning each deliverable (from the WBS) a realistic time frame. By augmenting the work over a timeline/calendar each milestone can be complete before the respective submission is required. The Gantt chart for this particular machine learning project (*see Appendix*) provides insight into the amount of time designated to each deliverable.

The characteristics associated with the Gantt chart are as follows:

- The project life-span begins on the 25$^{\text{th}}$ of February, and ends on the 3$^{\text{rd}}$ of November respectively.
- All physical submissions have a five day float to account for unexpected delays throughout the project life cycle.
- Leniency was granted when the estimated completion times were developed (to prevent the need of fast tracking).
- More time was allocated to tasks that were perceived to possess higher levels of importance in-terms of the scope
- By planning and accumulating literature before the milestones are executed, ensures that time is not lost working on unimportant project features.

In order to produce elegant and well-designed machine learning algorithms a provisional flow chart (to conventional standards) was produced. This flow diagram provides a visual representation of the basic tasks the code is required to execute. This diagram aids in the decomposition of the project's computer orientated tasks (similarly to the work breakdown structure). By breaking down the coding requirements, each milestone can be executed correctly before compiling the code into a larger data pipeline. This planning process reduces the time required to execute the overall deliverable. The generalised code for the project can be shown in the flow chart over-page:

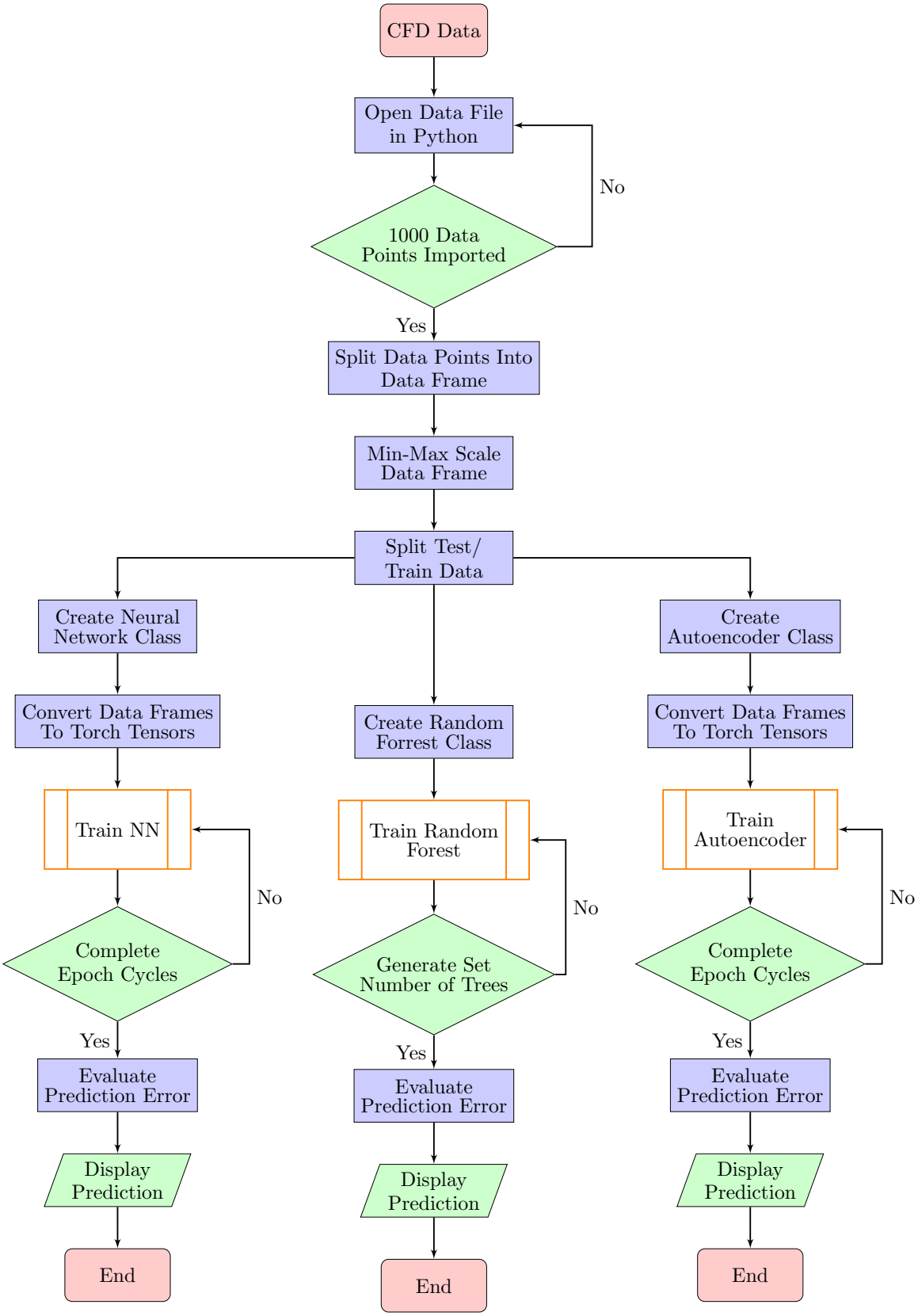


Figure 65: Simplified flow chart showing general process for machine learning algorithms

References

[1] P. Cheema, G. Mathews, B. Thornber and G. Vio, "Bayesian Inversion and Regression Trees for Mixing Length Model Development", in 20th Australasian Fluid Mechanics Conference, Perth, 2016.

[2] R. Srinivasan and A. Malkawi, "Real-time Simulations Using Learning Algorithms for Immersive Data Visualization in Buildings", international journal of architectural computing, vol. 3, no. 3, pp. 265 - 280, 2020. [Accessed 4 April 2020].

[3] G. Nimadge and S. Chopade, "CFD ANALYSIS OF FLOW THROUGH T-JUNCTION OF PIPE", International Research Journal of Engineering and Technology, vol. 4, no. 2, pp. 906 - 911, 2017. Available: http://www.irjet.net. [Accessed 26 March 2020].

[4] ANSYS FLUENT User's Guide, 14th ed. Canonsburg, PA: ANSYS, Inc., 2011, pp. 211 - 504.

[5] H. Ayhan and C. Sökmen, "CFD MODELING OF THERMAL MIXING IN T-JUNCTION: EFFECT OF BRANCH PIPE DIAMETER RATIO", in The 15th International Topical Meeting on Nuclear Reactor Thermal - Hydraulics, Pisa, Italy, 2013.

[6] W. Zuo, "Introduction of Computational Fluid Dynamics", Friedrich-Alexander-Universität Erlangen-Nürnberg.

[7] C. Sert, "Finite Element Analysis in Thermofluids", Middle East Technical University, 2018.

[8] A. Boatemaa, "Effects of Injection Pipe Orientation on Mixing Behavior in Contributing to Thermal Fatigue in a T-junction of a Pipe", MPhil, Department of NUCLEAR ENGINEERING UNIVERSITY OF GHANA, 2016.

[9] A. Bakker, "Applied Computational Fluid Dynamics", Hanover, United States, 2002.

[10] I. Goodfellow, Y. Bengio and A. Courville, Deep learning. Cambridge, MA: MIT Press.

[11] A. Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, 1st ed. Sebastopol, CA: O'Reilly Media, Inc., 2017.

[14] S. Brunton, B. Noack and P. Koumoutsakos, "Machine Learning for Fluid Mechanics", Annual Review of Fluid Mechanics, vol. 52, no. 1, pp. 477-508, 2020. Available: 10.1146/annurev-fluid-010719-060214.

[13] B. Hanna, "Coarse-Grid Computational Fluid Dynamics (CG-CFD) Error Prediction using Machine Learning", Ph.D, North Carolina State University, 2018.

[14] L. Grbcîć, L. Kranjčević, S. Družeta and I. Lucîn, "Efficient Double-Tee Junction Mixing Assessment by Machine Learning", Water, vol. 12, no. 238, pp. 1-16, 2020. Available: http://www.mdpi.com/journal/water. [Accessed 21 March 2020].

Appendix

EBE Ethics Form

Application for Approval of Ethics in Research (EiR) Projects
Faculty of Engineering and the Built Environment, University of Cape Town

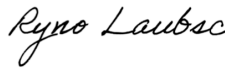
ETHICS APPLICATION FORM

Please Note:
Any person planning to undertake research in the Faculty of Engineering and the Built Environment (EBE) at the University of Cape Town is required to complete this form **before** collecting or analysing data. The objective of submitting this application *prior* to embarking on research is to ensure that the highest ethical standards in research, conducted under the auspices of the EBE Faculty, are met. Please ensure that you have read, and understood the **EBE Ethics in Research Handbook** (available from the UCT EBE, Research Ethics website) prior to completing this application form: http://www.ebe.uct.ac.za/ebe/research/ethics1

APPLICANT'S DETAILS		
Name of principal researcher, student or external applicant		Daniel Ferrini
Department		Mechanical Engineering
Preferred email address of applicant:		frrdan014@myuct.ac.za
If Student	Your Degree: e.g., MSc, PhD, etc.	BSc Mechanical and Mechatronic Engineering
	Credit Value of Research: e.g., 60/120/180/360 etc.	46
	Name of Supervisor (if supervised):	Dr Ryno Laubscher
If this is a researchcontract, indicate the source of funding/sponsorship		N/A
Project Title		Development of a machine learning regression model to predict a simple 2D flow field

I hereby undertake to carry out my research in such a way that:

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

APPLICATION BY	Full name	Signature	Date
Principal Researcher/ Student/External applicant	Daniel Ferrini		28/04/20
SUPPORTED BY	Full name	Signature	Date
Supervisor (where applicable)	Ryno Laubscher		18/05/2020

APPROVED BY	Full name	Signature	Date
HOD (or delegated nominee) Final authority for all applicants who have answered NO to all questions in Section 1; and for all Undergraduate research (Including Honours).			
Chair: Faculty EIR Committee For applicants other than undergraduate students who have answered YES to any of the questions in Section 1.			

Risk Identification Form

Department of Mechanical Engineering
Initial Risk Identification Form

Rev 1.1

- **Completion of this Risk Identification Form is required before any detail design or manufacturing is started.**
- **The purpose is to identify special safety requirements upfront, and to ensure that the necessary safety knowledge have been acquired, or specialist have been contracted.**
- **Failure to complete this form properly may result in refusal to perform your actual test / run your equipment.**
- **This form is incomplete without the attached supporting documentation as required for certain conditions.**

Student Name	Daniel Ferrini
Supervisor	Dr Ryno Laubscher
Project title and number	Development of a machine learning regression model to predict a simple 2D flow field [41]

Section A: Vessels under pressure

1. Does your design/apparatus have an operating pressure above 50 kPa(g), and holds gas or two-phase fluid heated to above atmospheric saturation conditions? *If YES, then complete the remaining section.*	YES	~~NO~~
2. Study the South African classification of pressure equipment (SANS 347:2007). Note that a container made up of welded pipes is considered a pressure vessel.	Sign initials	
3. Does your pressure vessel fall within the Standard Engineering Practice (category 0)? *Attach a brief report showing the calculations performed and steps followed to arrive at the classification.*	YES	NO
4. If **YES** to Question 3, you acknowledge the following: • Your final design will make use of appropriate pressure vessel calculations, and will be properly documented and verified by your supervisor. • You will complete a hydraulic test of 1.25 times the design pressure using water before actual testing or operation. This test will be witnessed by your supervisor and a photo of the pressure gauge reading will be included in your report. • Alternatively a pneumatic test may be performed in an enclosed environment. • If your setup is a refrigeration system, you will study the requirements of SANS 10147, and include in your final report the key aspects applicable to your design.	Sign initials	
5. if **NO** to Question 3, you acknowledge the following: • A suitable certified pressure vessel design & manufacturing company have been identified and has agreed to ensure all pressure regulations are met. *Proof of this must be attached.* • Your final report will contain a detail pressure vessel design pack, including a test certificate and certificate of compliance by the consulting company.	Sign initials	

Section B: Electrical installations

1. Does your design/apparatus make use of electricity above 50V where any element of the electrical design/manufacturing is done by yourself, i.e. not part of a bought-out component? *If YES, then complete the remaining section.*	YES	~~NO~~
2. Read the Machine Lab Safety Rulebook (on Vula), and then consult Mr Maysam Soltanian (Electrical Engineering Machines Lab) regarding specific requirements. *Attach a brief summary of discussion outcomes and requirements.*	Sign initials	
3. Have you been advised to contract an external electrician to assist?	YES	~~NO~~
4. if **YES** to Question 3, you acknowledge the following: • An electrician has been identified and he has agreed to assist. *Proof of this must be attached.* • You will include the electrical certificate in your final report.	Sign initials	
5. if **NO** to Question 3, you acknowledge the following: • You will generate proper electrical diagrams of your setup. • You will continue to consult Mr Soltanian w.r.t. wiring, equipment and testing. • You will have your Supervisor and/or Mr Soltanian witness the first power-on event.	Sign initials	

Section C: Hazardous chemicals and fuels

NO PROJECTS INVOLVING THE STORAGE OF HEATED FUEL MAY BE DONE

1. Does your design/apparatus make use of materials (chemicals, gasses, etc.) which may be hazardous to health, or the environment? **OR** Does your setup involve the use of flammable fuel, including LP Gas? *If YES, then complete the remaining section.*	YES	~~NO~~
2. Consult with Mrs Penny Louw (CME lab) or Prof Genevieve Langdon (BISRU) or Mr Sa-aadat Parker (Composites) regarding the specific requirements in terms of handling, storage, record keeping etc. *Attach a brief summary of discussion outcomes and requirements.*	Sign initials	
3. Attach the Material Safety Declaration Sheet (MSDS) for any hazardous material to be used.	Sign initials	

I have read all attached documentation and is satisfied that the necessary safety considerations will be adhered to during the physical execution of the project.	**Supervisor Signature** Ryno Laubscher	**Date** 18/05/2020

Impact of Technology Statement

Department of Mechanical Engineering

MEC4110W – Final-Year Project

Impact of Engineering Activity Rev5

By completing this assessment, you will help to demonstrate that you have met the requirement of ECSA's Graduate Attribute 7: Impact of Engineering activity. If it is determined that you have not successfully completed this task, you will be required to rework and resubmit this document. Should this still not be considered acceptable, you will not pass MEC4110W.

You are required to include this document in your Interim Report as an appendix. Your supervisor will read it and then sign as the reviewer.

In the space provided below, please write up to 300 words on how the technology in your project impacts on society, and then sign and date the form. You can consider "society" in one of or all three spheres: 1) your fellow students and other staff members, 2) the institution more generally, and/or 3) the broader society. You may need to consider the "downstream" impact of the technology in your project if you are undertaking a focused research-based project.

The application of machine learning as an aid to analysing CFD systems is a relatively unstudied field. As a result the findings of this research task may contribute a significant amount to the understanding of machine learning integrated CFD systems. This generalised research task could potentially be applicable not only to T-piece mixing domains, but also to many similar fluid simulation scenarios. By taking the same approach to other fluid problems and by applying the known and tested principals, this research can be extended and used to similarly analyse different fluid system designs and conditions. The theory developed from the accumulated research can also be applicable in other types of physics simulations (not only CFD). By integrating these developed machine learning models into software and other physics simulators, the algorithms tested in this project also have the potential to provide faster and relatively accurate solutions to complex problems that may otherwise be very time consuming to execute using conventional methods. This will inevitably allow these already powerful simulation tools to become more efficient and precise. Along with this 'downstream' application, the results of this research can also be immediately integrated into the broader society (both publicly and privately). The main goal of this research is to utilise machine learning for real-time analysis such as anomaly detection and what-if conditioning. Hence, these findings can be applied in parallel with already existing fluid systems to further improve the productivity and/or safety of these related systems. Due to the computational orientation of the project, the findings of this research can be easily be added or incorporated into the desired environment without needing to change the already existing system/infrastructure.

Signed	Date
	28/05/20

Reviewer's comments: _____

_____

_____

Reviewer's signature: _____ Date: _____

Work Breakdown Structure

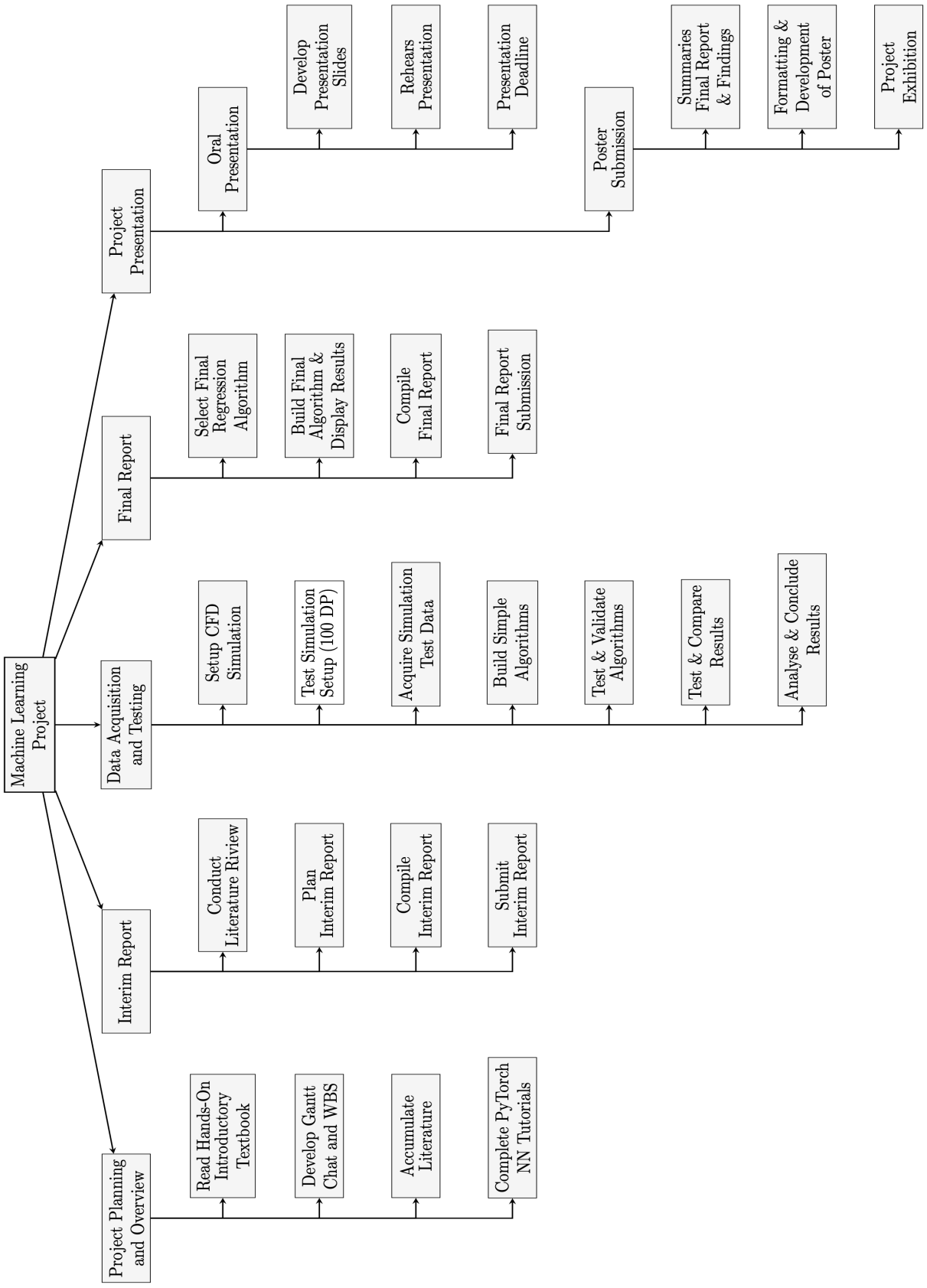


Figure 66: Work Breakdown Structure displaying the scope for machine learning project

Gantt Chart

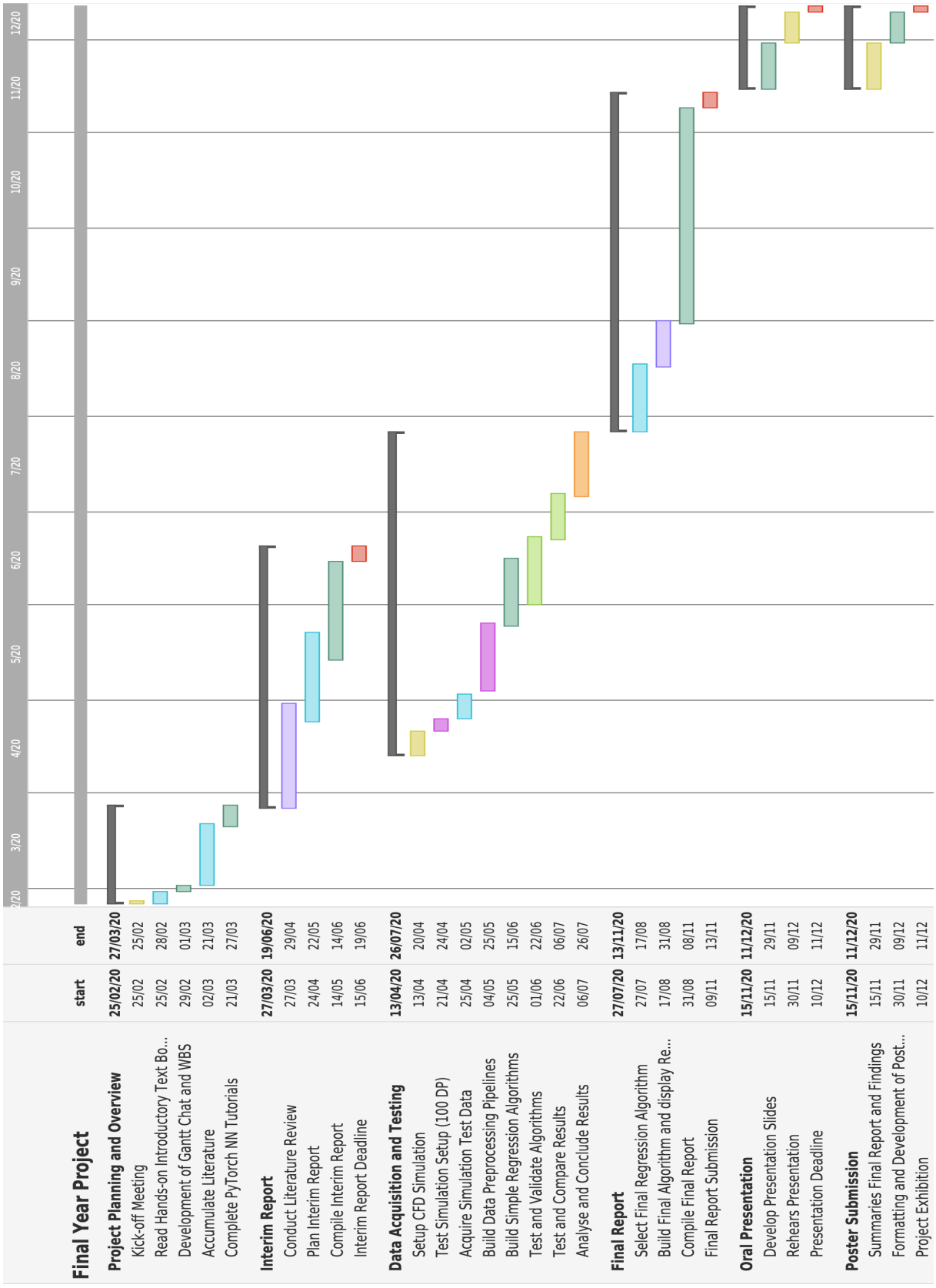


Figure 67: Gantt chart showing machine learning research project lifespan