ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

A FAST 3D FLOW FIELD PREDICTION AROUND BLUFF BODIES USING DEEP LEARNING

M.Sc. THESIS

Farhad NEMATI TAHER

Department of Mechanical Engineering

Heat-Fluid Programme

AUGUST 2023

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

A FAST 3D FLOW FIELD PREDICTION AROUND BLUFF BODIES USING DEEP LEARNING

M.Sc. THESIS

Farhad NEMATI TAHER
(503191155)

Department of Mechanical Engineering

Heat-Fluid Programme

Thesis Advisor: Assoc. Prof. Dr. Abdussamet SUBAŞI

AUGUST 2023

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

DERİN ÖĞRENME KULLANILARAK KÜT CİSİMLER ETRAFINDAKİ 3 BOYUTLU AKIŞ ALANININ TAHMİNİ

YÜKSEK LİSANS TEZİ

Farhad NEMATI TAHER
(503191155)

Makina Mühendisliği Anabilim Dalı

Isı Akışkan Programı

Tez Danışmanı: Doç. Dr. Abdussamet SUBAŞI

AĞUSTOS 2023

Farhad NEMATI TAHER, a M.Sc. student of İTU Graduate School, student ID 503191155, successfully defended the thesis entitled "A FAST 3D FLOW FIELD PREDICTION AROUND BLUFF BODIES USING DEEP LEARNING", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. Abdussamet SUBAŞI** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Nazım Kemal ÜRE** Istanbul Technical University	.....
	Assoc. Prof. Dr. Murat ÇELİK Bogazici University	.....

Date of Submission : 26 May 2023
Date of Defense : 28 August 2023

To my family,

FOREWORD

It is my great honor to express my heartfelt gratitude to my advisor, Assoc. Prof. Dr. Abdussamet SUBAŞI, whose unwavering encouragement, patience, and generous support throughout this journey have enabled me to develop a profound understanding of the subject matter.

Lastly, I would like to extend my deepest regards and blessings to my family, and all those who have supported me in any way during the completion of my thesis.

August 2023

Farhad NEMATI TAHER
(Mechanical Engineer)

TABLE OF CONTENTS

Page

ABBREVIATIONS

Adam	**:** Adaptive Moment Estimation
ASCII	**:** American Standard Code for Information Interchange
CNN	**:** Convolutional Neural Networks
CFD	**:** Computational Fluid Dynamics
ITU	**:** Istanbul Technical University
lr	**:** Learning Rate
ML	**:** Machine Learning
MLP	**:** Multilayer Perceptron
RANS	**:** Reynolds Averaged Navier-Stokes
ReLU	**:** Rectified Linear Unit
REP	**:** Point-wise Relative Error Percentage
SGD	**:** Stochastic Gradient Descent
TUI	**:** Text User Interface

SYMBOLS

D	**:** Length of prism in z-direction [m]
L	**:** Edge lengths of the right rhombic prism [m]
$\boldsymbol{p}$	**:** Pressure [Pa]
$\boldsymbol{p^*}$	**:** Dimensionless pressure [-]
$\boldsymbol{p'}$	**:** Normalized pressure [-]
$\boldsymbol{\widetilde{p}'}$	**:** Predicted pressure [-]
$\boldsymbol{p_0}$	**:** Atmospheric pressure [Pa]
R	**:** Distance of each point from the center of mass [m]
Re	**:** Reynolds number [-]
$\boldsymbol{x}$	**:** Input value [-]
$\boldsymbol{u_\infty}$	**:** Upstream velocity [m/s]
$\boldsymbol{u}$	**:** U velocity [m/s]
$\boldsymbol{u^*}$	**:** Dimensionless u velocity [-]
$\boldsymbol{u'}$	**:** Normalized u velocity [-]
$\boldsymbol{\widetilde{u}'}$	**:** Predicted u velocity [-]
$\boldsymbol{v}$	**:** V velocity [m/s]
$\boldsymbol{v^*}$	**:** Dimensionless v velocity [-]
$\boldsymbol{v'}$	**:** Normalized v velocity [-]
$\boldsymbol{\widetilde{v}'}$	**:** Predicted v velocity [-]
$\boldsymbol{w}$	**:** W velocity [m/s]
$\boldsymbol{w^*}$	**:** Dimensionless w velocity [-]
$\boldsymbol{w'}$	**:** Normalized w velocity [-]
$\boldsymbol{\widetilde{w}'}$	**:** Predicted w velocity [-]
$\boldsymbol{\lambda}$	**:** Regularization parameter [-]
μ	**:** Dynamic viscosity [Pa.s]
α	**:** Tip angle of the object [º]
β	**:** Inclination angle of the object [º]
θ	**:** Rotation angle of the object [º]

LIST OF TABLES

Page

LIST OF FIGURES

Page

A FAST 3D FLOW FIELD PREDICTION AROUND BLUFF BODIES USING DEEP LEARNING

SUMMARY

Deep learning, as a subset of machine learning, has various applications in the field of fluid mechanics, including flow prediction, flow control and optimization, fluid-structure interactions, and image-based fluid flow analysis. This study focuses specifically on flow prediction, aiming to estimate the behavior of flow field, such as velocity and pressure distributions, in an external turbulent flow using a deep learning model. The chosen deep learning architecture for this study is PoinNet, originally designed for machine vision and capable of processing unordered point clouds. Unlike convolutional neural networks, PoinNet does not require data interpolation, allowing direct utilization of unstructured grid system nodes as input, thus preserving the accuracy of the numerical solution. This model takes spatial coordinates of the grid system as input and produces the corresponding pressure and velocity quantities at the given input points.

Computational Fluid Dynamics (CFD) is employed to generate the required dataset for the training phase. A right rhombic prism with a side edge of 30 mm is selected as the base shape, with four variable geometrical parameters: Tip Angle (α), Inclination Angle (β), Rotation Angle (θ), and length in the specific direction (D). By varying only α and β, a smaller dataset consisting of 961 models is generated, while considering all parameters results in a larger dataset of 3511 different samples. The coordinate system is established at the center of mass of the solid shape.

A Tet-hybrid unstructured grid system is generated in the computational domain due to its ease of implementation in complex domains. A finer mesh is created around the object, accompanied by inflation layers applied to the object's walls to accurately capture the gradients in boundary layers. Considering the significant number of simulations involved, striking a balance between accuracy and computational time is crucial, leading to an effort to minimize the number of cells while maintaining reasonable accuracy.

The numerical solutions are obtained using Ansys-Fluent. After thorough testing of various turbulent models, considering both accuracy and computational time, the Standard k-ε epsilon turbulence model with a coupled pressure-velocity coupling scheme is adopted for the turbulent flow computations. Air under room temperature conditions, with constant thermophysical properties, is chosen as the working fluid. The upstream velocity is set to a constant value of 4.87 m/s. Velocity inlet and outflow boundary conditions are applied at the inlet and outlet of the computational domain, respectively. Meanwhile, no-slip and zero-shear stress conditions are imposed on the walls of the bluff body and the side walls of the domain, respectively.

Given the impracticality of manually setting up all the models, exporting the solution data, and saving the results of numerical solutions in Ansys-Fluent, it is coupled with MATLAB using the ANSYS_AAS toolbox and launched in AAS mode to automate

the simulation procedure. A MATLAB code and a Text User Interface (TUI) journal file are developed to provide Fluent settings for each case. The node-based solution data is exported as an ASCII file format for each sample.

Before concatenating all the exported files into a 3D array to form the dataset, some data manipulations are required. Firstly, it is unnecessary to predict the entire CFD domain with the neural network since the area of interest is limited to the region close to the object. Additionally, computational resources impose limitations on the number of inputs. Therefore, a specific algorithm is employed to filter out some points, restricting the number of inputs to 10,000.

Secondly, a single network is utilized to predict the results for four different variables, each with different ranges. To ensure equal contributions of output variables in determining network weights, data normalization is essential. After making the velocity and pressure variables dimensionless, the adjusted min-max method was utilized to scale them in the range of 0.1 to 0.9.

The PointNet neural network architecture has been modified to establish a mapping from the coordinates of the unstructured nodes in a 3D grid system as input to the corresponding values of the velocity components and pressure at the related nodes. This is a nonlinear regression problem that is modeled by a supervised learning neural network. An input array of size 10,000×3 goes through the network, and an output array of size 10,000×4 is produced. The 3 represents the coordinates of each node, and the 4 represents the number of predicted parameters, which are velocity components and pressure.

The training process involves optimizing the 11,231,821 trainable parameters in the model by minimizing the loss function. After shuffling the samples, the data for both datasets are partitioned into training, validation, and testing subsets, with proportions of 80%, 10%, and 10% for the smaller dataset with 961 models, and 65.5%, 20.5%, and 14% for the larger dataset with 3511 models, respectively. The implementation, training, and testing of the neural network have been carried out using the TensorFlow framework and the Keras library in this study. The training process utilized the resources provided by UHem; the National Center for High-Performance Computing located at Istanbul Technical University.

Prediction of a 3D flow field in a proper manner requires an increased number of input points. Consequently, the original PointNet architecture was modified to meet this challenge. The architecture was adjusted by changing the number of layers and the number of neurons. To optimize the hyperparameters of the deep learning model, such as the initial learning rate, the number of layers in each Multi-Layer Perceptron (MLP), and the number of neurons in different layers, a Bayesian optimization approach is employed. Constraints are applied to the number of layers and neurons in each MLP to define the search area in the Bayesian method. Due to the extensive number of possible combinations for different layers and neuron numbers, limitations were imposed on the maximum number of trials, restricting it to 80 tests. Additionally, the maximum number of epochs is set to 50.

The training process spanned 5000 epochs for the smaller dataset (961 samples) and 1400 epochs for the larger dataset (3511 samples). The larger dataset demonstrated a faster convergence speed in reaching the same loss values compared to the smaller dataset. However, due to the increased computational burden and the absence of further improvement in convergence, it was necessary to terminate the training at this stage. The final training and validation loss values for the smaller dataset are

1.25×10^{-4} and 1.34×10^{-4}, respectively, which were obtained at epoch 4600. For the larger dataset, the best achieved training and validation loss values are 3.08×10^{-4} and 2.54×10^{-4} at epoch 1400, respectively.

After the training phase, the generalizability of the network was tested using both datasets. For the smaller dataset, the network was evaluated using 96 previously unseen samples in the test subset. The test loss value for the smaller dataset is 1.38×10^{-4}, which is slightly larger than the validation loss. No abnormalities are evident in this evaluation. Similarly, the larger dataset underwent testing with a test subset comprising 490 samples. For this dataset, the test loss value was measured at 2.59×10^{-4}. This evaluation also indicates promising performance and aligns with the trends observed in the smaller dataset evaluation. Point-wise Relative Error Percentage (REP) and Average Relative Error Percentage (AREPs) were also considered to evaluate the performance of the deep learning model. The average error of the entire test set is close to the minimum value, indicating that most of the samples in the test subsets have a relative error value close to the minimum value.

The model is further evaluated through comparisons made between predicted data and numerical simulations. This evaluation involves analyzing velocity and pressure distribution contours, error distribution contours, correlation charts, and determination coefficients for each output parameter.

The model's performance in accurately predicting 3D flow fields with a satisfactory level of precision is impressive. Moreover, the network's ability to generate predictions significantly faster than conventional CFD solvers while maintaining excellent to reasonable accuracy is a notable advantage.

The ability to obtain accurate and fast flow field predictions has the potential to enhance engineering processes, optimize designs, and reduce computational costs. It opens up opportunities for more efficient simulations and facilitates quicker decision-making in fields such as fluid dynamics, aerodynamics, and engineering design.

DERİN ÖĞRENME KULLANILARAK KÜT CİSİMLER ETRAFINDAKİ 3 BOYUTLU AKIŞ ALANININ TAHMİNİ

ÖZET

Makina öğrenmesinin bir alt dalı olan derin öğrenme, akışkanlar mekaniği alanında akış alanı tahmini, akış kontrolü ve optimizasyonu, akışkan-yapı etkileşimleri ve görüntü tabanlı akış analizi gibi çeşitli uygulamalara sahiptir. Bu tez çalışmasında, derin öğrenme modeli kullanarak türbülanslı bir dış akışta 3-boyutlu akış alanının (hız ve basınç dağılımlarının) tahmin edilmesi amaçlanmaktadır. Bu çalışma için PoinNet derin öğrenme mimarisi seçilmiş olup, konvolüsyonel sinir ağlarının aksine, verilerin interpolasyonuna ihtiyaç duymamakta ve düzensiz ağların düğüm noktalarını doğrudan girdi olarak kullanabilmekte ve bu sayede sayısal çözümün hassasiyetini koruyabilmektedir. Bu model, ağ yapısının koordinatlarını girdi olarak almakta ve verilen girdi noktalarında karşılık gelen basınç ve hız değerlerini tahmin etmektedir.

Ağın eğitim aşaması için gerekli olan veriler Hesaplamalı Akışkanlar Dinamiği (HAD) simülasyonları ile üretilmiştir. Kenar uzunluğu 30 mm olan bir eşkenar dörtgen prizma temel şekil olarak seçilmiştir. Eşkenar dörtgen prizma; uç açısı (α), eğim açısı (β), dönel açı (θ) ve z-yönündeki uzunluk (D) olmak üzere dört geometrik parametreye sahiptir. Bu parametrelerden sadece α ve β değiştirilerek 961 farklı eşkenar dörtgen pirizması içeren küçük bir veri seti oluşturulurken, tüm geometrik parametreler dikkate alındığında 3511 farklı şekilden oluşan büyük bir veri kümesi elde edilmiştir. Koordinat sistemi şeklin ağırlık merkezinde olacak şekile oluşturulmuştur.

Tet-hibrit ağ yapısı karmaşık geometriler için kolay uygulanabilirliği nedeniyle tercih edilmiştir. Sınır tabakalarındaki değişimleri doğru bir şekilde yakalamak için cisme yakın bölgelerde daha sık bir ağ yapısı oluşturulmuştur. Simülasyonlar Ansys-Fluent kullanılarak gerçekleştirilmiştir. Farklı türbülans modellerinin hem doğruluk hem de hesaplama süresi göz önünde bulundurularak kapsamlı bir şekilde test edilmesinin ardından Standart k-ε epsilon türbülans modeli seçilmiştir. Çalışma kapsamında iş akışkanı olarak sabit termodinamik özelliklere sahip oda sıcaklığındaki hava seçilmiştir. Hesaplama alanının, giriş sınır şartı olarak 4.87 m/s serbest akış hızı ve çıkış sınır şartı olarak da çıkış (outflow) sınır şartı kullanılmıştır. Cismin duvarlarında kaymama sınır şartı uygulanırken, yan duvarlara sıfır kayma gerilmesi (zero-shear stress) koşulu uygulanmıştır.

Her bir simülasyonun ayarlanması, koşturulabilmesi ve sonuçların dışa aktarılabilmesi işlemlerinin otomatize edilebilmesi için ANSYS_AAS araç kutusu kullanarak Ansys-Fluent ile MATLAB programları konuşturulmuştur. Simülasyon sonuçlarının derin öğrenmede kullanılabilmesi için tüm sonuçların bir matriste toplanması ve verilerin manipüle edilmesi gerekmektedir. Bu kapsamda; öncelikle akış alanı tahmini yapılacak cismin yakın komşuluğundaki 10,000 noktanın seçileceği bir algoritma oluşturulmuştur. Daha sonra x, y ve z yönlerindeki hız ve basınç olmak üzere dört farklı değişkenin tek bir derin öğrenme ağı üzerinden tahmini yapılacağı için farklı alt

ve üst değere sahip her bir değişkenin eşit katkı sağlaması için veriler min-max yöntemi kullanılarak 0.1 ila 0.9 aralığında normalize edilmiştir.

PointNet mimarisi üzerinde, 3-boyutlu akış alanını tahmin edebilmek için çeşitli değişiklikler yapılmıştır. Her bir cisim için x, y ve z koordinatları ağın girdileri; x, y, ve z yönündeki hızlar ve basınç ise ağın çıktıları olarak belirlenmiştir. Bu kapsamda herbir cisim için ayrı ayrı 10,000×3 boyutunda bir giriş matrisi ağa girmekte ve tamin sonucunda da 10,000×4 boyutunda bir matris ağ tarafından üretilmektedir.

Ağın eğitilmesi süreci, kayıp fonksiyonunu en aza indirecek model parametrelerinin belirlenmesidir. Küçük ve büyük olmak üzere iki farklı veri seti için ağlar ayrı ayrı eğitilmiştir. Bu kapsamda veriler karıştırıldıktan sonra küçük veri seti için verilerin %80'i eğitim, %10'u doğrulama ve %10'u da test verisi olarak ayrılırken; büyük veri seti için verilerin %65.5'i eğitim, %20.5'i doğrulama ve %14'ü de test verisi olarak ayrılmıştır. Çalışma kapsamında; ağın eğitilmesi ve test edilmesinde TensorFlow ve Keras kütüphaneleri kullanılmıştır. Eğitim sürecinde, İstanbul Teknik Üniversitesi bünyesinde bulunan Ulusal Yüksek Başarımlı Hesaplama Merkezi (Uhem) tarafından sağlanan kaynaklar kullanılmıştır. Derin öğrenme modelinin hiperparametrelerini (başlangıç öğrenme hızını, her Çok Katmanlı Algılayıcıdaki (MLP) katman sayısını ve farklı katmanlardaki nöron sayısını) optimize etmek için Bayes optimizasyonu yaklaşımı kullanılmıştır. Farklı katmanlar ve nöron sayıları için olası kombinasyonların sayısının oldukça fazla olması nedeniyle, maksimum deneme sayısı 80 ile sınırlanmıştır. Ayrıca, maksimum EPOCH sayısı da 50 olarak belirlenmiştir.

Eğitim süreci, küçük olan veri seti (961 örnek) için 5000 EPOCH sürerken, büyük veri seti (3511 örnek) için 1400 EPOCH sürmüştür. Büyük veri seti, küçük veri setine göre aynı kayıp değerlerine ulaşma konusunda daha hızlı bir yakınsama hızı sergilemiştir. Ancak artan hesaplama yükü ve yakınsamada daha fazla ilerlemenin olmaması nedeniyle bu aşamada eğitimi sonlandırılmıştır. Küçük veri seti için nihai eğitim ve doğrulama kayıp değerleri sırasıyla 1.25×10^{-4} ve 1.34×10^{-4} olarak, 4600. EPOCH'ta elde edilmiştir. Büyük veri seti için de en iyi elde edilen eğitim ve doğrulama kayıp değerleri sırasıyla 3.08×10^{-4} ve 2.54×10^{-4} olarak, 1400. EPOCH'ta elde edilmiştir.

Eğitim aşamasından sonra, ağın genel tahmin yeteneği her iki veri seti için ayrı ayrı test edilmiştir. Küçük veri seti için ağ tarafından daha önce görülmemiş 96 örnek denenmiştir. Bu deneme için kayıp değeri 1.38×10^{-4} olarak belirlenmiştir. Benzer şekilde, büyük veri seti için 490 örnek için ağın tahminleme performansı test edilmiştir. Bu test soucunda ise kayıp değeri 2.59×10^{-4} olarak belirlenmiştir. Bu değerlendirmelere ek olarak Noktasal Göreceli Hata Yüzdesi (REP) ve Ortalama Göreceli Hata Yüzdesi (AREP) ayrıca derin öğrenme modelinin performansını değerlendirmek için dikkate alınmıştır. Bu değerlendirmelerle birlikte eğitilen ağların 3-boyutlu akış alanlarını yüksek bir hassasiyet seviyesiyle doğru bir şekilde tahmin ettiği ve Hesaplamalı Akışkanlar Dinamiği (CFD) simülasyonlarına kıyasla tahin süresinin oldukça kısa olduğu sonucuna varılmıştır. Dolayısıyla derin öğrenmenin; 3-boyutlu akış alanını doğru ve hızlı bir şekilde tahmin etme yeteneği göz önüne alındığında, mühendislik süreçlerini hızlandırma, tasarımları optimize etme, hesaplama maliyetlerini azaltma ve mühendislik tasarımı gibi alanlarda daha hızlı karar verme potansiyelinini yüksek olduğu sonucuna varılmıştır.

1. INTRODUCTION

The determination of the flow field in classical fluid mechanics applications typically entails solving the Navier-Stokes (NS) equations through computational fluid dynamics (CFD) techniques. Despite advancements in computational speed and power, as well as the development of efficient numerical methods, this process remains time-consuming, particularly for complex prototypes, often taking hours or even days. Situations that involve extensive iterations of flow solutions, such as fluid-structure interaction and design optimization, further contribute to the computational expense and prolong the overall design cycle. In the early stages of design, designers frequently require the ability to rapidly explore various design alternatives and make preliminary decisions, typically without the need for high-fidelity simulations. Therefore, there is a demand for an effective and precise method capable of obtaining flow solutions at a significantly faster pace than the traditional CFD approach, allowing designers to obtain instant or real-time feedback for continuous design iterations. One possible option is using quick approximation models or surrogates as an alternative to high-accuracy CFD simulations. Data-driven methods, as alternative surrogates, have the potential to enhance or replace costly high-fidelity CFD analyses with more economical approximations. The practicality and significance of data-driven surrogate models have increased due to the substantial growth in computational speed and power. Deep learning algorithms have made significant advances in the field of data-driven learning in recent years, and have effectively contributed to the advancement of innovative computational methodologies. They offer a rapid and proficient alternative for approximating functions in high-dimensional complex spaces. Specifically, deep neural networks (DNNs), commonly employed in data mining practices, exhibit favorable compatibility with large-scale, high-dimensional datasets, enabling the extraction of features at multiple scales.

1.1 General Definition and Concepts

In this section, the machine learning concepts utilized in this study are briefly reviewed.

1.1.1 Machine learning

Machine learning is a specialized domain within the field of artificial intelligence (AI) that concentrates on the creation of algorithms and models enabling computers to learn and make predictions or decisions autonomously, without explicit programming. It encompasses the utilization of statistical methods to automatically recognize patterns within data, allowing for precise predictions or informed actions to be taken based on those patterns.

1.1.2 Deep learning

Deep learning is a specific branch of machine learning that focuses on the development and training of multi-layered artificial neural networks that can learn and extract complicated patterns and representations from immense amounts of data. The term "deep" in deep learning does not imply a deeper understanding achieved by the method but rather refers to the idea of building layers of representations. A model's depth indicates the quantity of layers contributing to the representation of the data in that model. Modern deep learning models typically comprise tens or more sequential layers, allowing them to automatically learn from training data. This stands in contrast to other machine learning approaches that generally rely on only one or two learning layers, leading to their classification as shallow learning methods. [1].

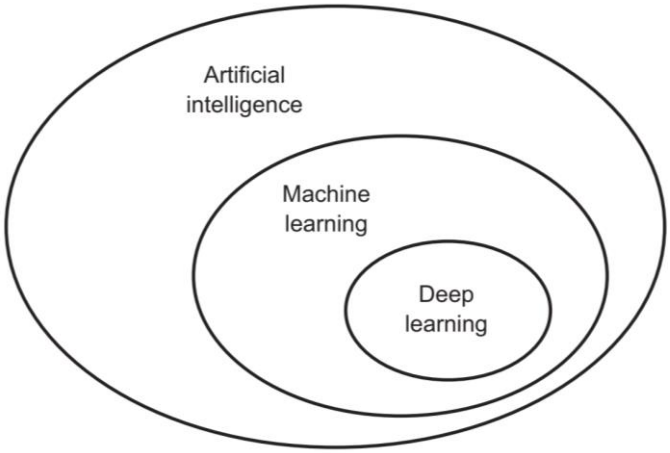


Figure 1.1 : Deep learning, machine learning, and artificial intelligence [1].

1.1.3 Neural networks

Neural networks, also known as artificial neural networks (ANNs), constitute a subfield of machine learning approaches and serve as a fundamental component in deep learning algorithms. Their design and functioning are inspired by the human brain, aiming to replicate the interconnection and signaling of biological neurons. ANNs are composed of multiple layers of nodes. Each node, representing an artificial neuron, is interconnected to other nodes and possesses assigned weights and thresholds. When the output of a node surpasses a specific threshold, it becomes activated, and its output is transmitted to the following layer. Conversely, if the output falls below the threshold, no data is forwarded to the next layer of the network [2].

Each neuron in a neural network has a set of learnable parameters, such as biases and weights. The weights determine the strength or significance of the connections between neurons, whilst the biases add an extra parameter to change each neuron's output. These parameters are iteratively adjusted throughout the training phase based on a defined optimization strategy, generally employing techniques like gradient descent and backpropagation.

The key operation in a neural network is the weighted sum of inputs followed by an activation function. The weighted sum calculates the linear combination of inputs and their corresponding weights, while the activation function introduces non-linearity to the output. The activation function allows the neural network to learn complex and nonlinear relationships within the data.

There are various types of activation functions employed in ANNs, such as the hyperbolic tangent (tanh) function, rectified linear unit (ReLU) function, and sigmoid function. Each activation function has its characteristics and affects the network's learning dynamics.

Neural networks can be exploited in a variety of tasks such as classification, regression, and clustering. They demonstrated outstanding performance in image recognition, speech recognition, and natural language processing, which makes them a valuable tool in machine learning and artificial intelligence.

1.1.4 Multi-layer perceptron (MLP)

A Multi-Layer Perceptron (MLP) is a variant of feedforward neural networks. The term "ANN" is a broader and more general concept that encompasses various

architectures and types of neural networks, including MLP. MLP is the simplest and most basic form of a neural network and serves as the foundation for more complex architectures in deep learning. An MLP is composed of several layers of interconnected nodes or neurons, with each neuron being a processing unit that performs computations on the input data. The layers are organized sequentially, with a layer for inputs, one or more hidden layers, and a layer for outputs (see Figure 1.2). Each neuron in one layer of an MLP is connected to every neuron in the next layer. This structure is known as a fully connected or dense architecture. The connections between neurons are characterized by weights, which represent the strength of the connections.

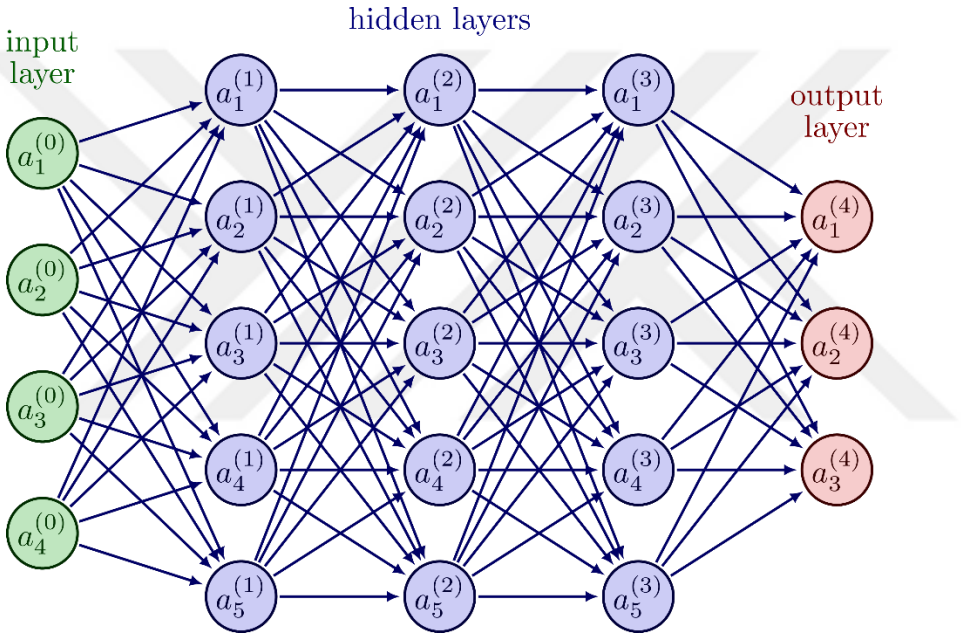


Figure 1.2 : Schematic of a multi-layer perceptron [3].

1.1.5 Convolutional neural network (CNN)

A Convolutional Neural Network (CNN) is a specialized variant of neural networks commonly exploited for analyzing visual data, such as images or videos. CNNs are particularly effective in tasks involving object detection, image classification, and image segmentation due to their ability to automatically learn and extract meaningful features from raw pixel data. The distinguishing feature of CNNs is the inclusion of convolutional layers (see Figure 1.3).

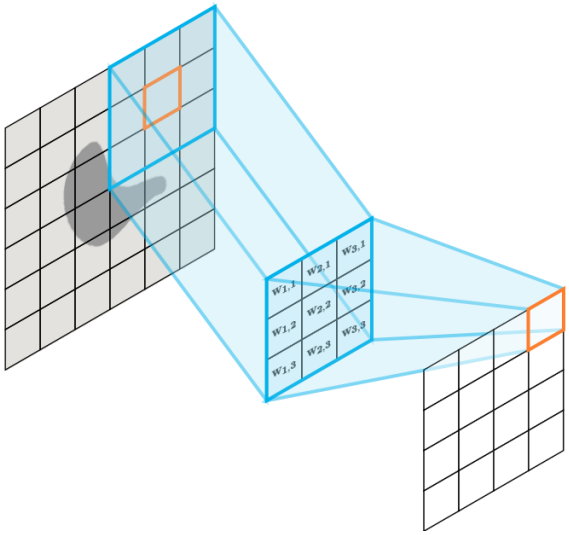


Figure 1.3 : Visual representation of the convolution operation on an input image with a kernel [4].

These layers utilize a set of adaptive filters or kernels to process the input image through a convolution operation. During the convolution operation, the filters are slid across the input image, conducting element-wise multiplications at each position and subsequently summing the results to generate a feature map. By using multiple filters, CNNs can capture different patterns and features at different spatial locations of the image.

The strength of CNNs lies in their ability to automatically learn hierarchical representations of visual data. By leveraging the convolutional and pooling operations, CNNs can capture local patterns and gradually build complex and abstract representations of the input. This hierarchical approach allows CNNs to effectively handle images of varying sizes and capture important features regardless of their position within the image.

1.1.6 Adam optimizer

An optimization algorithm is a method used to iteratively update the trainable parameters of a model, such as weights and biases, to minimize a given loss or objective function. It plays a crucial role in training machine learning models and obtaining the optimal value of parameters that best suit the data.

Adam (short for Adaptive Moment Estimation) is a popular optimization algorithm variant utilized for updating neural network model parameters during training. It is a

stochastic gradient descent method that relies on adaptive estimation of both first-order and second-order moments. [5].

As stated by Kingma et al. (2014), the Adam method offers computational efficiency, low memory requirements, and invariance to diagonal rescaling of gradients. It is particularly well-suited for handling large-scale problems with significant amounts of data and parameters [6].

1.1.7 Relu activation function

The activation function in a neural network denotes a mathematical function that is applied to the output of neurons within a neural network. Its role is to introduce non-linear characteristics into the network, enabling it to learn and model complicated relationships in the data. The choice of activation functions significantly influences the output and learning dynamics of neural networks. Among the various options, the Rectified Linear Unit (ReLU) activation function holds substantial importance, particularly in deep learning models, due to its widespread adoption as a non-linear activation function. The ReLU function is defined as follows:

$$f(x) = max\,(0, x) \tag{1.1}$$

By taking an input value x, it returns the maximum value between 0 and x. In other words, the ReLU function behaves in the following manner: if the input is positive, it outputs the input value as is, while if the input is negative, it outputs 0. As a consequence, the ReLU activation function exhibits a piecewise linear behavior, allowing positive values to pass through unchanged while setting negative values to zero. This simple characteristic defines its fundamental functionality.

The key advantage of the ReLU activation function is its simplicity and computational efficiency. The function is easy to compute and does not involve complex mathematical operations, making it highly suitable for training deep neural networks that encompass a substantial number of parameters. Additionally, ReLU helps mitigate the vanishing gradient problem that can occur in networks with deep architectures, allowing for more effective training.

1.1.8 Sigmoid activation function

The sigmoid activation function, also referred to as the logistic function, is a widely employed non-linear activation function in neural networks. It transforms the input value into a range spanning from 0 to 1, resulting in a smooth and continuous output. The mathematical definition of the sigmoid function is as follows:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.2}$$

where e is the base of the natural logarithm and x is the input value.
The sigmoid activation function has some drawbacks. One issue is that the output of the sigmoid function saturates for large positive or negative input values, resulting in gradients close to zero. This saturation can cause the vanishing gradient problem and hinder the learning process, especially in deep neural networks.

In modern deep learning architectures, sigmoid still finds applications in specific scenarios, such as the final layer of a binary classifier where a probability interpretation is desired or in certain types of recurrent neural networks.

1.1.9 Batch normalization

Batch normalization is a method employed in deep learning models to enhance the learning process and the overall performance of the network. Its primary concept involves normalizing the inputs of each layer by subtracting the mean of the batch and dividing by the standard deviation of the batch. This normalization is applied to mini-batches of training examples rather than individual examples. The normalized inputs are then scaled and shifted by learnable parameters, known as the batch normalization parameters, which allow the network to restore representation power.

Batch normalization offers several benefits. First, it reduces the internal covariate shift, which stabilizes the training process and allows for faster convergence. Normalizing the inputs, helps the network to adapt to changes in the distribution of the input data and makes the optimization process more efficient. Additionally, batch normalization acts as a form of regularization by adding noise to the network's hidden units. This noise helps in overfitting reduction and enhancing the generalization ability of the model. Batch normalization also provides some robustness to the selection of the learning rate and can reduce the sensitivity of the network to the initialization of the

weights. It allows for using higher learning rates without the risk of instability and makes the network less dependent on careful weight initialization.

1.1.10 Max pooling

Max pooling is a down-sampling technique commonly used in CNNs for image and feature extraction tasks. It reduces the spatial dimensions of the input data while retaining the most prominent features. The process of max pooling involves partitioning the input into non-overlapping regions, typically squares or rectangles, and selecting the maximum value within each region. The selected maximum value becomes the output for that region, effectively reducing the spatial resolution. Max pooling offers several benefits. First, it helps to extract and preserve the most salient features from the input, as the maximum value represents the strongest response within each region. This helps the network focus on the most relevant information and discard less significant details. Second, max pooling introduces a degree of translation invariance, making the network more robust to small shifts or variations in the input. Since the maximum value is selected, the precise location of a feature becomes less important, allowing the network to detect the presence of features regardless of their exact positions. Another advantage of max pooling is its ability to decrease the network's complexity of computation by down-sampling the feature maps. This makes the subsequent layers more efficient to compute and decreases the quantity of parameters of the network, which can help prevent overfitting. Typically, max pooling is applied after a convolutional layer in CNNs, resulting in a reduction of the spatial dimensions and simultaneously increasing the depth or number of channels. It is often followed by additional convolutional or fully connected layers to further process the extracted features. However, it's important to note that max-pooling may discard some spatial information, and alternative pooling techniques, such as average pooling or adaptive pooling, can be used depending on the task's requirements.

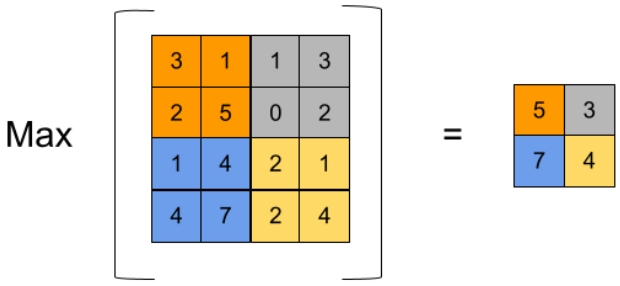


Figure 1.4 : An example of applying a 2×2 max pool on a 4×4 input [7].

1.1.11 L2 regularization

L2 regularizer also referred to as weight decay, is categorized as a type of layer weight regularizer. These regularizers impose penalties on layer activity or layer parameters throughout the training process. These penalties are aggregated and incorporated into the loss function, which the network then seeks to optimize [8].

The L2 regularization term is calculated by summing up the squared magnitudes of all the weights in the model and multiplying it by a regularization parameter, often denoted as λ. The regularization term is then added to the initial loss function. The modified loss function with L2 regularization can be written as:

Loss-with-regularization = Loss-without-regularization + λ × (sum of squared weights)

The purpose of the regularization term is to penalize large weights in the model. By doing so, L2 regularization promotes a trade-off between achieving a good fit to the training data and maintaining small weights in the model. This helps prevent overfitting by reducing the model's reliance on any particular feature and promoting more generalizable representations.

L2 regularization is widely used in various machine learning algorithms and neural network architectures. It is a simple and effective technique for preventing overfitting and improving the generalization performance of models.

1.2 Literature Review

The literature review provides a thorough overview of current research and understanding of the topic of employing deep learning in fluid mechanics. The purpose of this section is to review and synthesize the relevant works, theories, and approaches that have contributed to the knowledge and advancement of flow field prediction using deep learning. This review defines the background for the current study and highlights gaps, disputes, and areas for further research by critically examining and synthesizing the available literature.

Guo et al. (2016) used CNNs to present a broad and adaptable approximation model for predicting steady laminar flow in a 2D and 3D space. They investigated numerous options for geometry representation and network design. The results demonstrated that CNNs could estimate the velocity field significantly faster than traditional CFD

solvers, with low rates of error. In the initial phases of design, this method allows for instant feedback and real-time design iterations. The authors also discussed the potential extension of the approach to higher Reynolds number flows and the possibility of using the approximation models to warm start high-accuracy CFD simulations, decreasing the required number of iterations to achieve convergence. Overall, the CNN-based CFD predictions offered an efficient solution for flow performance feedback, design exploration, and optimization [9].

Yilmaz and German (2017) applied deep learning methodologies to predict airfoil performance, aiming to overcome the limitations of traditional surrogate modeling approaches. They used CNNs with neural network-based classifiers to map the relationship between airfoil geometry and performance. Results showed over 80% accuracy in predicting airfoil performance. The study demonstrated that deep learning can automatically detect features from unstructured data, eliminating the need for specific geometric parameterizations and enabling generalization. This approach has the potential to reduce costs and time associated with expensive CFD simulations and wind tunnel experiments, making it a promising tool for early aircraft and airfoil design phases [10].

Farimani et al. (2017) used deep learning to create a data-driven framework for quick reasoning and simulation of transportation phenomena. They employed conditional Generative Adversarial Networks (cGAN) to construct solutions for heat conduction and fluid flow based on observations without considering the governing equations. The cGAN approach achieved high test accuracy and computational efficiency, outperforming iterative numerical methods. It also allowed for learning causal models from complex or unknown physical systems using experimental data [11].

Wu et al. (2019) introduced ffsGAN, a surrogate model founded on generative adversarial networks (GANs) and CNNs, for efficient and accurate transonic flow field prediction in supercritical airfoil design. By leveraging the power of deep learning and GANs, ffsGAN establishes a direct mapping from parameterized airfoil profiles to flow field profiles. The model is trained using 500 airfoils and demonstrates strong generalization capability. Experimental results confirm the effectiveness of ffsGAN in rapidly evaluating detailed aerodynamic performance. The study concludes that ffsGAN can replace expensive simulations and experiments, offering precise flow field evaluation in seconds. Future work will focus on modeling changeable flow field

conditions and incorporating detailed flow field structures into airfoil optimization [12].

Tamaddon-Jahromi et al. (2020) investigated the potential of using deep learning methods to address inverse problems that exhibit both linear and nonlinear characteristics. To create the training dataset, forward problems were solved by proposing a combination of machine learning and computational mechanics. After the training stage, the model was utilized for determining the boundary conditions based on assumed measurements. The method was tested on various heat conduction, convection-conduction, and natural convection problems, achieving high model accuracy. The study demonstrated that the combination of machine learning and computational mechanics was an effective approach to solve complex inverse problems. The ML approach provided robustness, less reliance on individual users, and the ability to handle significantly more complex systems compared to traditional trial-and-error methods [13].

In the article by Bhatnagar et al. (2019), the authors proposed an approximation model founded on CNNs aimed at efficiently forecasting aerodynamic flow patterns. The CNN takes inputs such as the angle of attack, Reynolds number, and the geometry of the airfoil in the form of a Signed Distance Function. The training process involves utilizing the solutions of Reynolds Averaged Navier-Stokes (RANS) upon various airfoil shapes. The CNN has the ability to recognize critical features automatically with minimal human intervention and predict the flow field at a substantially faster rate than the RANS solver. This allows for near-real-time analysis of how the aerodynamic forces and flow field are affected by airfoil shape and operating conditions. The study explores different network architectures and demonstrates the effectiveness of convolution operations, parameter sharing, and gradient sharpening techniques in enhancing the predictive capabilities of CNN. The results show that the CNN-based approach enables rapid simulation-driven optimization and design, offering a more efficient design process with accurate predictions [14].

Sekar et al. (2019) used deep learning approaches to forecast incompressible laminar steady flow fields across airfoils. The approach involved combining CNN and MLP models. Geometrical parameters from airfoil shapes were extracted using CNN, while the MLP network utilized these parameters along with the Reynolds number and angle of attack to predict the flow field. The required training database was generated using

the OpenFOAM solver, and once trained, the model could obtain the flow field surrounding an airfoil within seconds. The outcomes of the investigation demonstrated the efficiency and accuracy of the approach, indicating the capability of deep learning in predicting flow fields in fluid mechanics applications [15].

Chen et al. (2019) used supervised neural networks to predict velocity and pressure fields surrounding 2D random shapes in laminar flows. They created a dataset of random shapes annotated with pressure and velocity fields calculated by solving N-S equations. On this dataset, several U-net architectures were trained, and their predictive efficiency on unseen shapes was assessed using specific error functions. The authors concluded that U-net-based architectures showed promise in accurately predicting velocity and pressure fields for 2D objects with random shapes in laminar flows. They also explored the possible applications of these predictions in more complex situations, such as turbulent viscosity maps and 3D fields, emphasizing the use of such tools as surrogate models for optimization [16].

Yao Zhang et al. (2018) explored CNN approach applicability for aerodynamic meta-modeling applications. Their study aimed to develop a suitable CNN architecture for predicting lift coefficients of airfoils under various flow conditions and object geometries. Multiple CNN structures were trained on diverse parameters such as flow Reynolds numbers and angles of attack. The CNN models exhibited competitive prediction accuracy, comparable to multi-layered perceptron (MLP) solutions while providing greater flexibility in geometric representation. The study demonstrated the potential of using CNN architectures, such as AeroCNN-II, for aerodynamic meta-modeling, allowing for the analysis of 2D aerodynamic problems with different flow conditions and airfoil shapes within a single framework. By incorporating both geometric and non-geometric boundary conditions into an image-like array, the CNN models showed promising results and highlighted the applicability of deep learning techniques in engineering meta-modeling tasks [17].

Sekar et al. (2019) introduced an approach for inverse airfoil design utilizing CNNs. The CNNs were applied to effectively obtain airfoil shapes from pressure coefficient distributions, eliminating the need for complex parameterization. Extensive research was undertaken on several hyperparameters and CNN frameworks. The outcomes demonstrated that the CNN approach has competitive prediction accuracy and processing efficiency. It emerged as a promising tool for real-time airfoil design and

optimization, offering a gradient-free, fast, robust, global, and accurate alternative to conventional methods [18].

Hajgató et al. (2020) used CFD and deep neural networks to predict flow fields in distorted U-shaped pipes. They demonstrated that employing a deep convolutional neural network as a surrogate for CFD models can achieve a substantial speed-up of several orders of magnitude. The trained network successfully predicted flow fields based on different geometries, with acceptable accuracy and significant computational efficiency [19].

In the study by Fukami et al. (2019), machine learning techniques are employed to conduct high-resolution analysis on under-resolved turbulent flow field data, with the goal of reconstructing high-resolution flow fields from low-resolution data. Two machine learning models are created and evaluated: the CNN and the hybrid Down-sampled Skip-Connection Multi-Scale (DSC/MS) models. As demonstrated in the two-dimensional cylinder wake test, the models are capable of reconstructing laminar flow from low-resolution data. Furthermore, the CNN and DSC/MS models perform admirably in reconstructing turbulent flows from exceedingly coarse flow field pictures [20].

Hui et al. (2020) proposed employing CNNs to anticipate the pressure distribution around airfoils in aerodynamic design. By leveraging deep learning techniques, they offer an alternative to the time-consuming CFD method and the limitations of surrogate modeling. The authors demonstrate that their CNN-based approach achieves accurate predictions of the pressure coefficient. They further highlight the effectiveness of the SDF as a flexible parametrization method for improving CNN performance. Their findings indicate that deep learning can successfully approximate nonlinear solutions and provide faster computational results compared to traditional methods, showcasing its potential for efficient and accurate aerodynamic meta-modeling tasks [21].

In a study conducted by Thuerey et al. (2020), the efficacy of deep learning models in estimating RANS solutions was examined. They employed a modernized U-net architecture and evaluated multiple trained neural networks to calculate pressure and velocity distributions. The study focused on turbulent flow in airfoil geometries. They got outstanding results by studying the impact of training data size and the number of

weights, with a mean relative pressure and velocity inaccuracy of less than 3% across all previously unknown airfoil types. [22].

In their study, Eichinger et al. (2020) introduced a CNN-based technique for constructing reduced-order surrogate models in CFD simulations. They focused on predicting steady laminar flow field images in a channel with varying obstacles based on the channel's geometry image. Comparing different CNN architectures, they found that the U-Net architecture yielded highly accurate predictions and demonstrated good generalization to unseen geometries. They also explored transfer learning and sequential learning strategies to reduce training data requirements. The approach provided a faster alternative to traditional CFD simulations, with the evaluation of CNNs being approximately 100 times faster than performing simulations using OpenFOAM. Overall, their study showcased the potential of CNNs in constructing efficient surrogate models for steady laminar flow simulations with varying geometries [23].

Shahane et al. (2021) developed CNN models to analyze 2D laminar steady flow over elliptic cylinders. The CNNs accurately predicted velocity and pressure fields, while multilayer perceptron neural networks (MLPNNs) estimated lift and drag coefficients. CFD simulations were conducted using COMSOL software, and the trained networks showed good agreement with validation data. This research highlights the potential of CNNs and MLPNNs for fluid mechanics analysis, aiding design optimization and sensitivity studies [24].

J. An et al. (2020) presented a new deep learning-based framework for turbulent combustion simulation. CFDNN is the framework built on a CNN with a U-Net architecture. Simulation results of hydrogen combustion in a cavity with different inlet velocities are used to train the model. The model's input is the flow field data at time t_0, and the model's output is the flow field information at $t_0 + \Delta t$ which is the next time step. This output information is subsequently fed back to the CNN as another input. The trained CFDNN demonstrates accurate predictions beyond the training set. Additionally, CFDNN achieves a significant acceleration of two orders of magnitude compared to traditional CFD solvers, making it highly efficient [25].

Ribeiro et al. (2021) introduced DeepCFD, a CNN model founded on U-nets for approximating solutions to non-uniform steady laminar 2D flows. DeepCFD was up to three orders of magnitude faster, compared to standard CFD approaches while

maintaining low error rates. The authors showed that the distinct decoders in the U-Net design exceeded other models with regard to precision and performance [26].

In a two-dimensional steady laminar flow scenario, Li-Wei Chen et al. (2021) created U-net based deep neural network (DNN) models for determining flow fields and utilized them as surrogate models for optimizing the shape of airfoils. The DNN models utilized level-set and Béziers curve approaches for shape parameterization and were trained on high-fidelity datasets. The trained DNN models accurately predicted flow fields and yielded satisfactory aerodynamic forces, demonstrating their capability beyond flow field prediction. The DNN-based optimization framework, integrated with automatic differentiation, effectively minimized drag and showcased promise for general aerodynamic design problems [27].

Bouhlel et al. (2020) developed mSANN, a machine learning methodology for rapid and accurate airfoil design optimization. Using gradient-enhanced artificial neural networks trained on CDF data, mSANN achieved similar optimization results to high-fidelity simulations but with significantly reduced computational time. The approach outperformed existing models, providing accurate predictions for a wide range of airfoil shapes and flow conditions. Optimization problems were solved in seconds, enabling interactive design processes [28].

Waxenegger-Wilfing et al. (2020) established an artificial neural network (ANN) model to forecast the maximum temperature of the wall in the cooling channels of a supercritical methane-cooled rocket combustion chamber. The ANN was trained using CFD simulation data and achieved a mean absolute error of 16.0 K in predicting wall temperature for unknown cases. The prediction time for the whole channel segment was decreased to 0.6 s, which is more than 103 times quicker than typical CFD simulations. [29].

Kashefi et al. (2021) introduced an innovative deep learning framework for making flow field predictions in irregular domains. Their approach utilized the PointNet architecture to learn the mapping between spatial positions and CFD quantities. The framework preserved the accuracy of CFD data, accurately represented object geometry, and avoided the need for data interpolation. The authors demonstrated the effectiveness of their framework in predicting flow fields around various objects and an airfoil, achieving significantly faster predictions compared to traditional CFD solvers while maintaining reasonable accuracy. It is important to note that the flow

considered in their study was 2D and laminar. The proposed framework offered advantages in terms of accuracy, efficiency, and generalizability, making it a promising approach for flow field predictions in complex domains [30].

Pant et al. (2021) presented Deep Learning – Reduced Order Modelling (DL-ROM), an original framework for the effective prediction of future time steps in CFD simulations. The framework was trained using 3D Autoencoder and 3D U-Net architectures. DL-ROM generates highly precise reconstructions and predicts future time steps by exploiting the learned reduced state rather than iteratively solving the computationally costly N-S equations. Ground truth supervision is not required, resulting in significant computational savings. The method was tested on multiple CFD datasets and demonstrated a nearly two-order-of-magnitude drop in computational runtimes whilst maintaining acceptable error thresholds. [31].

Du et al. (2021) introduced a rapid and collaborative design model for optimizing the aerodynamics of airfoils. The approach employs a B-spline-based generative adversarial network (BSplineGAN) for shape parameterization, which eliminates implausible airfoils and decreases the design space. Neural network surrogates, including multilayer perceptron, and recurrent neural networks, enable predictions of drag and lift and pressure distribution responses for various Mach and Reynolds numbers. The optimization results obtained by the proposed framework align well with direct CFD-based optimizations, indicating its accuracy [32].

Yang and Mesri. (2022) introduced an improved approach for using Physical Informed Neural Networks (PINN) to predict fluid flow dynamics, specifically focusing on two-dimensional incompressible flows around complex geometries. The PINN model is enhanced by incorporating a One-hot matrix model that considers boundary conditions and non-uniform weights based on physical properties. The One-hot matrix helps classify different constraints in the computational domain, improving prediction accuracy on boundaries, while also allowing for adjustment of local weights to focus on key areas and high-gradient mesh nodes. The proposed approach demonstrates improved accuracy in predicting flow around sharp rectangular obstacles, even with low-resolution datasets. Future work will explore applying this approach to three-dimensional test cases with more complex geometries [33].

Liu et al (2023) developed an aerodynamic design optimization framework using deep learning (DL) to suppress dynamic stall in rotor airfoils. They replaced costly CFD

simulations with a DL-based surrogate model, resulting in faster and more accurate predictions of aerodynamic coefficients. The optimized airfoil showed a significant reduction in peak drag and moment coefficients by 82.5% and 88.6%, respectively, while lift coefficients increased, indicating improved dynamic stall characteristics. The DL-based surrogate model also shortened the optimization time by at least one order of magnitude compared to traditional CFD methods [34].

Zuo et al. (2022) employed a data-driven technique using a CNN and multi-head perceptron (MHP) to predict the sparse flow field surrounding airfoils. The CNN extracts geometry parameters from grayscale images, which are then combined with other flow field parameters such as flow field coordinates, Reynolds number, and angle of attack as inputs to the MLP and MHP. The MHP architecture achieves better prediction results than the multi-layer perceptron for sparse flow field data. The CNN network exhibits high prediction capability, with a correlation coefficient of 0.9999 between prediction and ground truth values. The deep learning method offers flexibility and can be further improved through pre-training and expanded training sets. The proposed MHP architecture decouples the multi-variable prediction task, reducing interference and improving accuracy and generalization. The results demonstrate the effectiveness of the method in predicting flow fields around different airfoil geometries [35].

1.3 Motivation

As observed in the literature review, most of the studies concerning the application of machine learning in fluid mechanics focus on two-dimensional (2D) scenarios. However, due to the complexity of the majority of industrial and real-world flow problems, simplifying them into two-dimensional (2D) flows is often ineffective. As a result, there is a growing need to develop effective deep learning models to tackle these challenges, similar to the common practice in CFD techniques. The aim of this study is to develop a deep learning framework specifically designed for predicting flow fields in 3D external flows. To the best of our knowledge, this study represents the first utilization of PointNet for predicting the 3D flow field around bluff bodies.

2. DATA PREPARATION

The details of the data generation method and process are elaborated in this section. First, a description of the geometry used in the data generation process is provided. Subsequently, the modeling and mesh generation procedures, along with the employed solution approach, are discussed. The primary goal of this chapter is to provide readers with a clear understanding of the procedures used to obtain the data required for neural network training.

2.1 Geometry Representation

In this study, a "right rhombic prism"[1] and its special form, the cube, were selected as base shapes for the generation of a dataset to train a neural network. The modeled objects were exposed to an external incompressible flow, resulting in two distinct datasets containing 961 and 3511 samples, respectively.

As depicted in Figure 2.1, for the first dataset, the edge lengths of the right rhombic prism (L) were fixed at 30 mm, while two geometrical parameters, namely the Tip Angle (α) and Inclination Angle (β), were varied within the ranges of 10° to 90° and -87° to +90°, respectively. By varying these parameters, a total of 961 distinct samples were obtained.

In addition to the previously mentioned parameters, the second dataset considered the Rotation Angle (θ) and the length of the right rhombic prism in the Z-direction (D). Rotation Angle (θ) represents the angle of rotation of the prism around the X axis, while the length in the Z-direction (D) is the distance between the rhombic faces of the prism, as shown in Figure 2.2. By varying these additional parameters within the ranges of -5 to +5 degrees and 30 to 40 mm, respectively, a larger dataset consisting

[1] Right rhombic prism has two rhombic faces and four square faces. Rhombic faces are always parallel.

of 3511 unique samples was generated. A summary of all parameters is presented in Table 2.1.

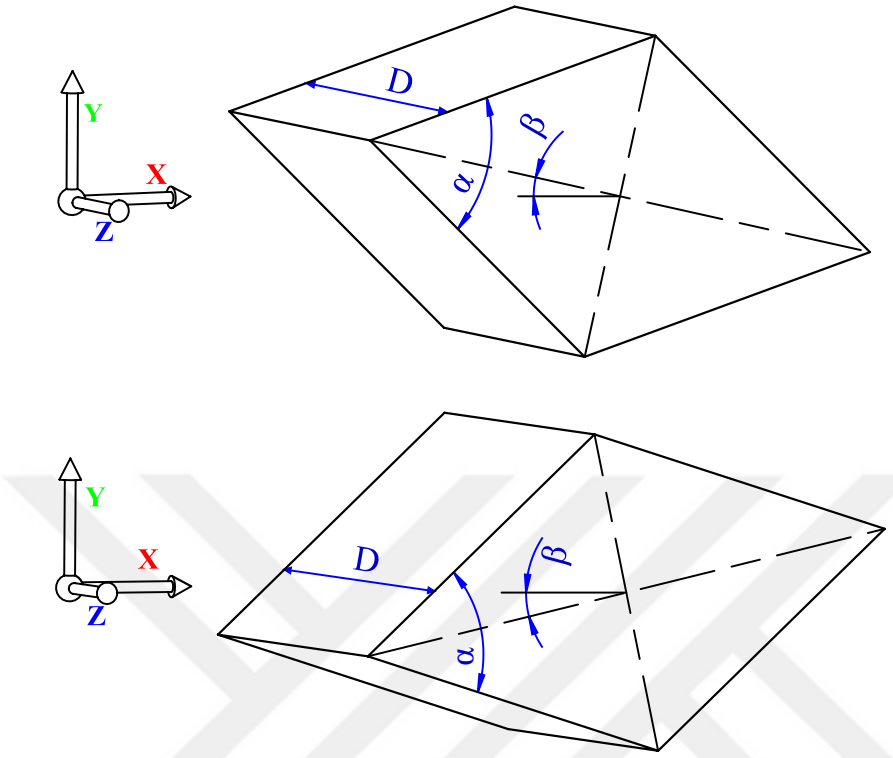


Figure 2.1 : Geometric parameters for the Right Rhombic Prism - Tip Angle (α) and Inclination Angle (β). Note: In the upper figure, β is positive, while in the lower figure, β is negative.

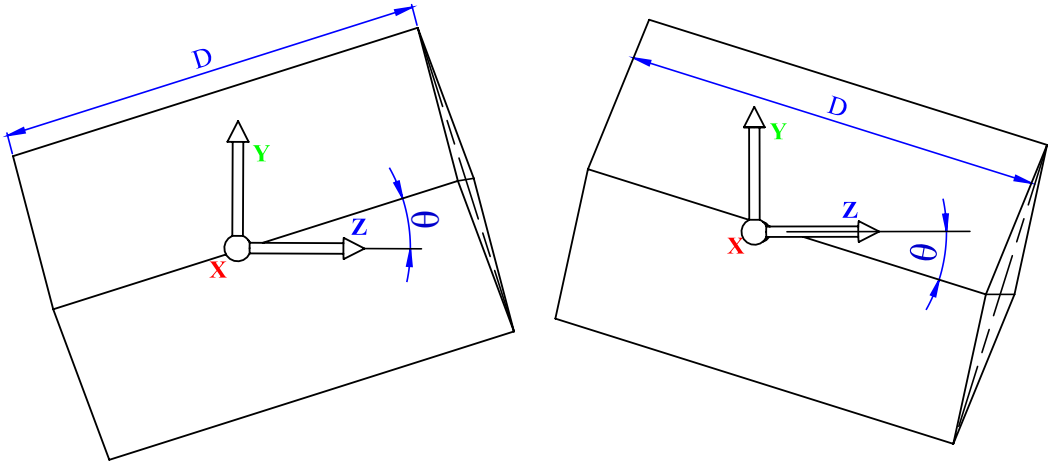


Figure 2.2 : Additional geometric parameters for the solid object - Rotation Angle (θ) and Length in the Z-Direction (D). Note: In the left figure, θ is positive, while in the right figure, θ is negative.

Table 2.1 : Description of the geometrical parameters utilized for data generation.

Tip angle (α)	Inclination angle (β)	Rotation angle (θ)	Length in the Z-direction (D)
10°	-87 to +90 by 3° intervals	-5 to +5 by 5° intervals	30 to 40 by 10 mm intervals
15°	-87 to +90 by 3° intervals	-	-
20°	-87 to +90 by 3° intervals	-5 to +5 by 5° intervals	30 to 40 by 10 mm intervals
25°	-87 to +90 by 3° intervals	-	-
30°	-87 to +90 by 3° intervals	-5 to +5 by 5° intervals	30 to 40 by 10 mm intervals
35°	-87 to +90 by 3° intervals	-	-
40°	-87 to +90 by 3° intervals	-5 to +5 by 5° intervals	30 to 40 by 10 mm intervals
45°	-87 to +90 by 3° intervals	-	-
50°	-87 to +90 by 3° intervals	-5 to +5 by 5° intervals	30 to 40 by 10 mm intervals
55°	-87 to +90 by 3° intervals	-	-
60°	-87 to +90 by 3° intervals	-5 to +5 by 5° intervals	30 to 40 by 10 mm intervals
65°	-87 to +90 by 3° intervals	-	-
70°	-87 to +90 by 3° intervals	-5 to +5 by 5° intervals	30 to 40 by 10 mm intervals
75°	-87 to +90 by 3° intervals	-	-
80°	-87 to +90 by 3° intervals	-5 to +5 by 5° intervals	30 to 40 by 10 mm intervals
85°	-45 to +45 by 3° intervals	-	-
90°	-42 to +45 by 3° intervals	-5 to +5 by 5° intervals	30 to 40 by 10 mm intervals

2.2 Numerical Details

This sub-chapter provides a comprehensive account of the procedures used to generate the dataset for our study. Specifically, it explains the steps taken in modeling the 3D geometry, meshing the domain, and solving the N-S equations using Ansys-Fluent. By presenting these details in a clear and organized manner, this sub-chapter aims to provide readers with a thorough understanding of our methodology.

2.2.1 3D modeling and mesh generation

Figure 2.3 illustrates the 3D bluff body and its enclosure which comprise the computational domain. The center of the coordinate system was positioned at the object's center of mass, with the flow passing through the object in the X-direction. The CFD domain's walls have been configured to maintain fixed distances from the

object. To reduce the impact of wall distance on the CFD results and to minimize the number of mesh cells, which in turn reduces the computational cost of simulations, the values of these distances were determined by analyzing various sizes. The distances from the front, back, and side walls of the enclosure are 4.5L, 10L, and 3L, respectively, where L represents the object's edge size, which is equal to 30mm. Figure 2.4 depicts some examples of modeled samples.

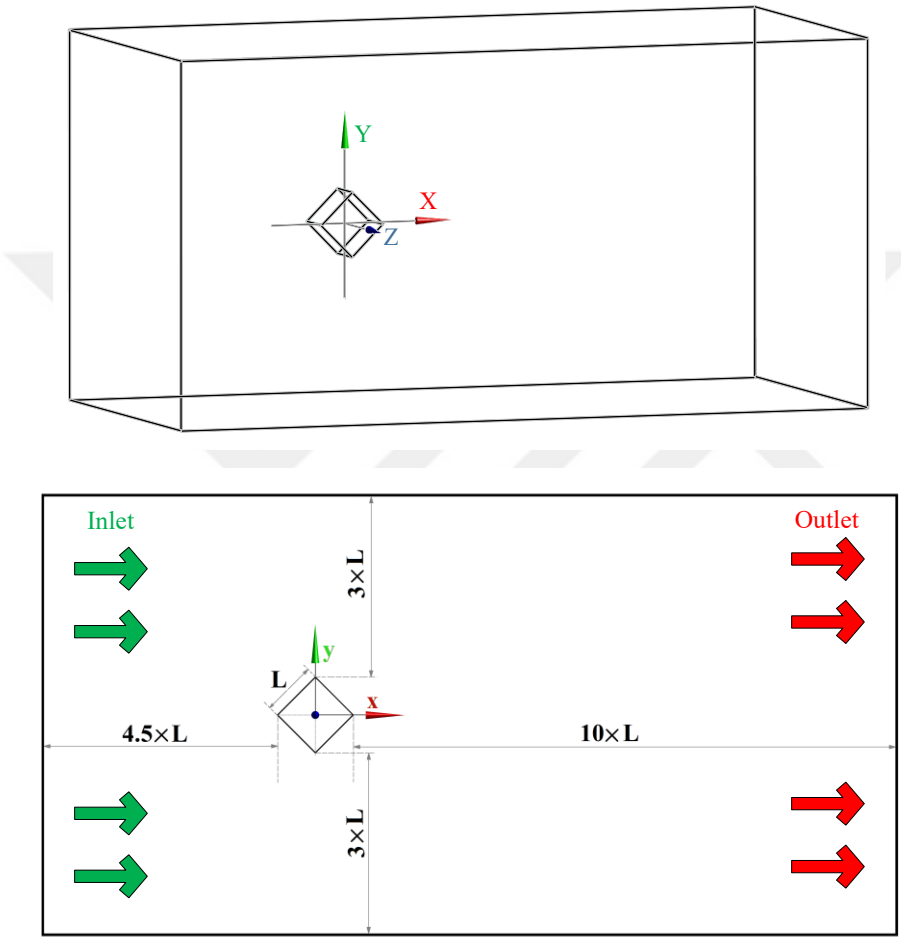


Figure 2.3 : Computational domain. The upper figure shows the 3D model of the geometry and the lower figure shows the wall distances.

Generating a proper grid structure is crucial to obtain accurate CFD results. Due to its rapid mesh generation ability and effortless implementation, the Tet-Hybrid unstructured grid system was chosen to mesh the computational domain. A finer mesh was created around the object to capture the flow physics accurately, while inflation layers were applied to the object walls to accurately capture the velocity gradients within boundary layers. Figure 2.5 illustrates the cross-sections of selected samples with an unstructured mesh.

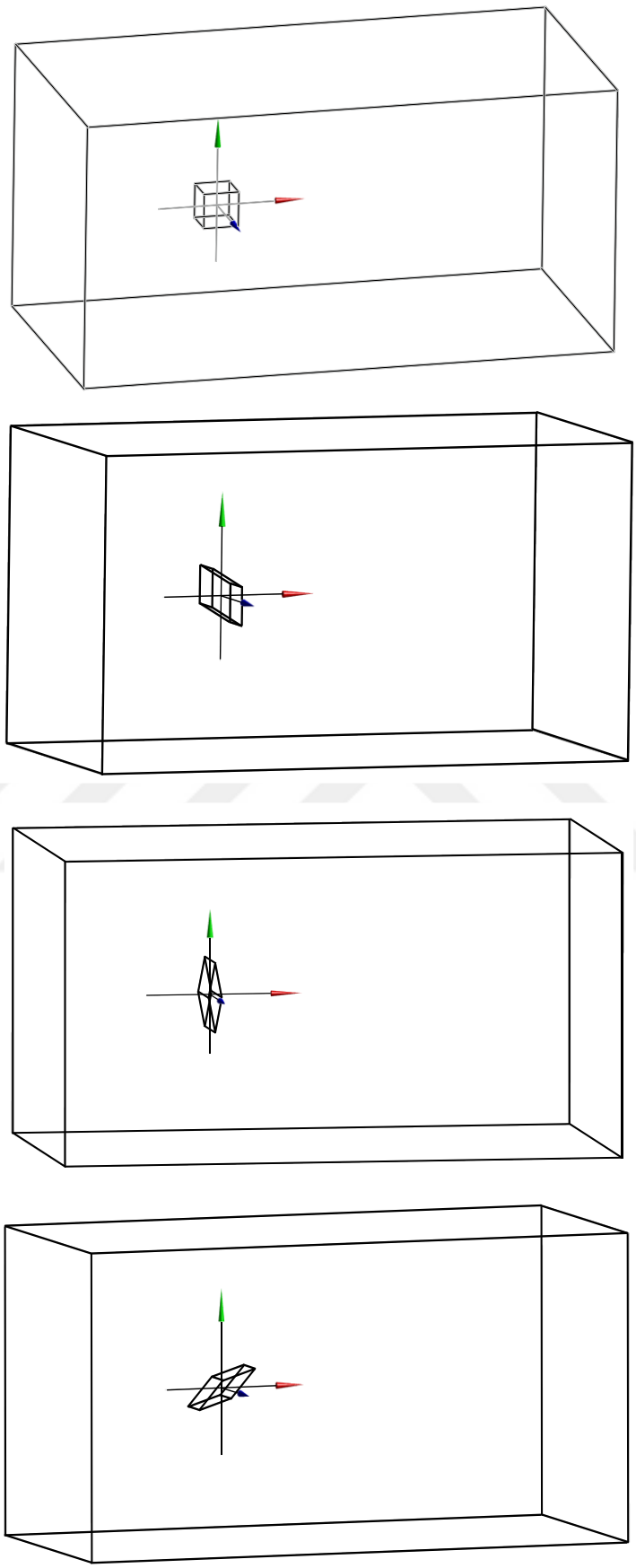


Figure 2.4 : CFD domains of selected samples.

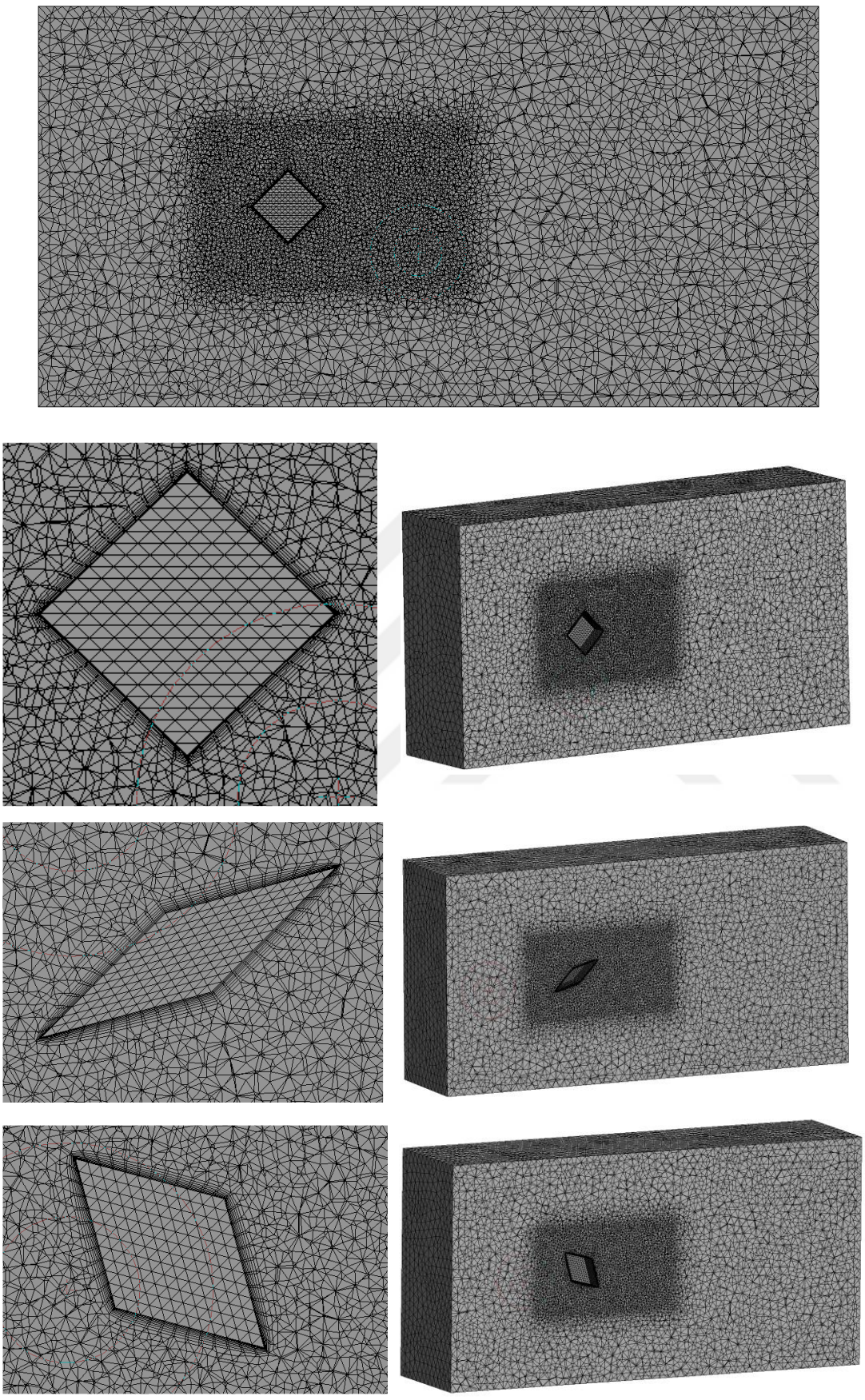


Figure 2.5 : Mesh details of selected samples in an x-y plane. Cross-section at $z = 0$.

2.2.2 Governing equations

CFD simulations rely on the solution of governing equations that describe fluid flow behavior. In this study, Ansys-Fluent was utilized in the subsequent stage to obtain numerical solutions for turbulent flow. A thorough test was conducted to select the most efficient and accurate turbulence model among various models available in Ansys-Fluent, such as the Spalart-Allmaras, Standard k-ω, Standard, RNG, and Realizable k-ε models. Ultimately, the Standard k-ε turbulence model with a coupled pressure-velocity coupling scheme was adopted for the turbulent flow computation. The governing equations and transport equations associated with the Standard k-ε Model are given below.

Continuity:

$$\frac{\partial}{\partial X_{\mathrm{i}}}(\rho \bar{u}_{\mathrm{i}}) = 0 \tag{2.1}$$

Momentum:

$$\frac{\partial}{\partial X_{\mathrm{i}}}(\rho \bar{u}_{\mathrm{i}} \bar{u}_{\mathrm{k}}) = \frac{\partial}{\partial X_{\mathrm{i}}}\left(\mu \frac{\partial \bar{u}_{\mathrm{k}}}{\partial X_{\mathrm{i}}}\right) - \frac{\partial \mathrm{P}}{\partial X_{\mathrm{k}}} \tag{2.2}$$

The turbulence kinetic energy k:

$$\frac{\partial}{\partial t}(\rho k) + \frac{\partial}{\partial X_{\mathrm{i}}}(\rho k u_{\mathrm{i}}) = \frac{\partial}{\partial X_{\mathrm{j}}}\left(\alpha_{\mathrm{k}} \mu_{\mathrm{eff}} \frac{\partial k}{\partial X_{\mathrm{j}}}\right) + G_{\mathrm{k}} + \rho\varepsilon \tag{2.3}$$

The rate of dissipation ε:

$$\frac{\partial}{\partial t}(\rho \varepsilon) + \frac{\partial}{\partial X_{\mathrm{i}}}(\rho \varepsilon u_{\mathrm{i}}) = \frac{\partial}{\partial X_{\mathrm{j}}}\left(\alpha_{\varepsilon} \mu_{\mathrm{eff}} \frac{\partial \varepsilon}{\partial X_{\mathrm{j}}}\right) + C_{1\varepsilon} \frac{\varepsilon}{k} G_{\mathrm{k}} - C_{2\varepsilon} \rho \frac{\varepsilon^2}{k} \tag{2.4}$$

For an expanded explanation of the equations please refer to Ref [36].

2.2.3 Boundary conditions

The velocity inlet and outflow boundary conditions were implemented at the entrance and exit of the CFD domain, respectively. Air was selected as the working fluid. Fluid density and viscosity were assigned values of 1.255 kg/m^3 and 1.79×10^{-5} Pa.s, respectively. An upstream velocity of 4.87 m/s was adopted for all samples. Reynolds number is calculated as:

$$Re = \frac{\rho u_{\infty} L}{\mu} \tag{2.5}$$

where ρ denotes the fluid density, u_∞ refers to the upstream velocity, and μ represents the dynamic viscosity. The Reynolds number depends only on the characteristic length L while the density, free stream velocity, and viscosity are constant. Two types of boundary conditions, zero shear stress, and symmetry were evaluated on the side walls of the enclosure. Despite minor variations between the results, the symmetry condition resulted in a longer solution convergence time; therefore, the zero-shear stress condition was implemented on the side walls of the enclosure. The walls of the object were assigned a non-slip boundary condition. It was assumed that the fluid flow was in a steady-state and turbulent regime.

2.2.4 Solution procedure

To obtain numerical approximations of the governing equations, the finite volume method was employed and iteratively solved. The pressure-based solver with a coupled pressure-velocity scheme, utilizing second-order upwind schemes for momentum, second-order schemes for pressure, as well as first-order schemes for turbulent kinetic energy and turbulent dissipation rate, was selected as the solver setting. The computational process was conducted until the monitored convergence criteria, including the residual values for the momentum and continuity equations, reached a value of 10^{-6}.

2.2.5 Validation and verification of CFD solution

2.2.5.1 Mesh independence test

To ensure the accuracy of the numerical results, a comprehensive investigation into grid independence is necessary. To achieve reliable results and validate the grid independence, a total of seven Tet-Hybrid grid systems with varying sizes were tested at a velocity of 4.87 m/s. The determined grid sizes and outcomes of the mesh independence test for the drag coefficient of the 4 different samples are presented in Figure 2.6 to Figure 2.9. The results indicate that the difference between the last four grids is less than 1.5% in the worst test sample (Figure 2.9). Therefore, taking into account both computation cost and accuracy, the grid configuration (body and face grid sizes) resulting in the fourth mesh size (from the left to right in the figures) was chosen for all samples.

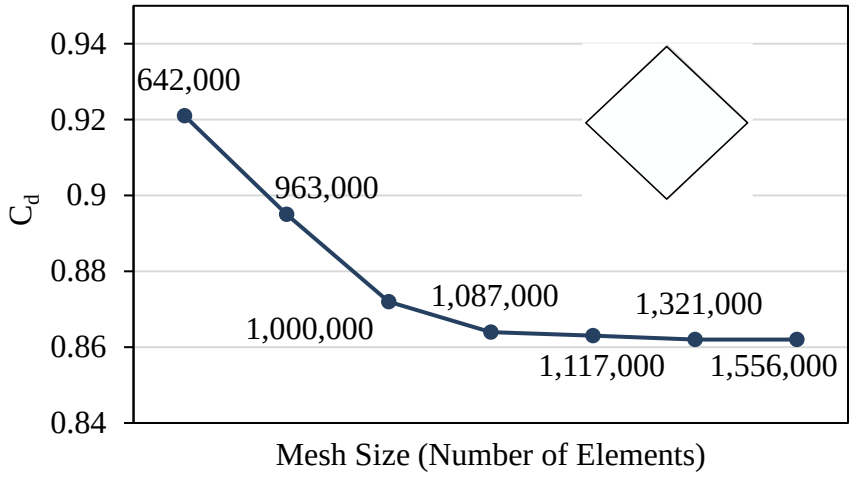


Figure 2.6 : Mesh independence test for the sample with α = 90° and β = 0°.

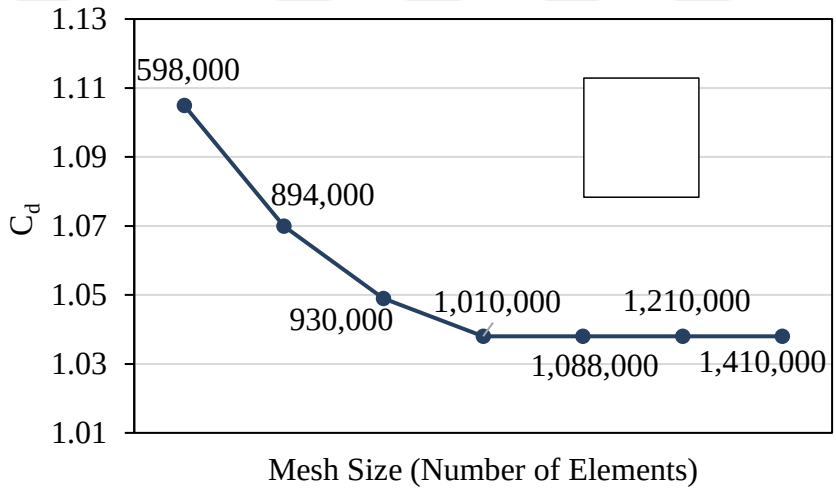


Figure 2.7 : Mesh independence test for the sample with α = 90° and β = 45°.

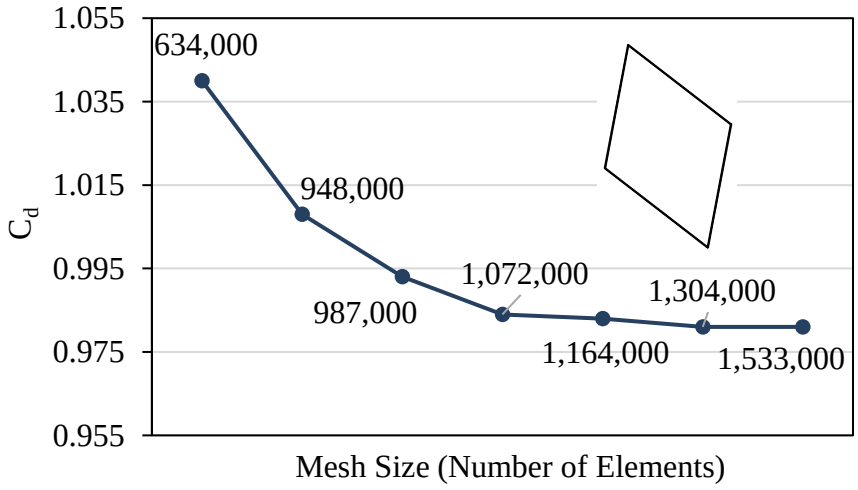


Figure 2.8 : Mesh independence test for the sample with α = 60° and β = 70°.

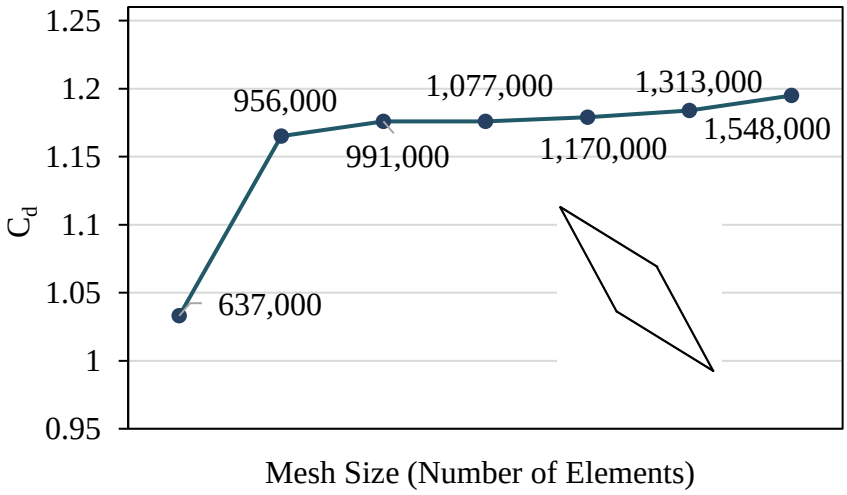


Figure 2.9 : Mesh independence test for the sample with $\alpha = 30°$ and $\beta = 45°$.

2.2.5.2 Validation

To verify the accuracy of our numerical results, it is necessary to perform a comparison between the CFD simulation results and those obtained from an experimental study. Several sources like Hoerner [37], have reported that the drag coefficient of a cube is 1.05 at the Reynolds number of 1×10^4. At the same Reynolds number, our simulation of the base cube with identical geometrical properties resulted in a drag coefficient of 1.038 which represents an error of less than 1.15%. Additionally, the drag coefficient of a cube, oriented at 45° to the streamline, is reported as 0.8 at the same Reynolds number. Our simulation for this particular sample resulted in a drag coefficient of 0.864, representing an error of 8%. Taking into account the balance between computational cost and accuracy, the model demonstrates an acceptable level of precision.

2.2.5.3 Numerical simulation results

For some of the samples, the obtained velocity and pressure contours are illustrated in Figure 2.10 as an example. The contour analysis reveals valuable insights into the flow behavior around the bluff body.

In the velocity contours, we can observe the presence of flow separation and recirculation zones behind the body. These regions are represented by areas of lower velocity or even reversed flow, indicated by darker blue colors on the contours. Flow separation occurs when the flow detaches from the surface of the body due to the adverse pressure gradient encountered. This phenomenon leads to the formation of

recirculation zones where the flow moves in the opposite direction, creating eddies and vortices. Additionally, a reduction in flow velocity in front of the body can be seen. This reduction in velocity is caused by adverse pressure gradients due to the presence of the body.

The pressure contour shows distinct regions of high and low pressure around the body. Along the front surfaces, a region of relatively high pressure is observed, indicating that the flow experiences higher resistance as it encounters the leading edges of the body. This suggests that the flow has a tendency to be compressed as it encounters these edges where the flow decelerates. Moving further downstream, we notice a transition to lower pressures around the sides and rear of the body. These regions exhibit a decrease in pressure, implying a tendency for the flow to expand as it moves past the body.

The obtained results from the numerical simulations reveal significant observations. The identified key points, including flow separation, recirculation zones, pressure differences, and the impact of the bluff body, are expected to be consistent between the numerical simulations and the outcomes of the deep learning model.

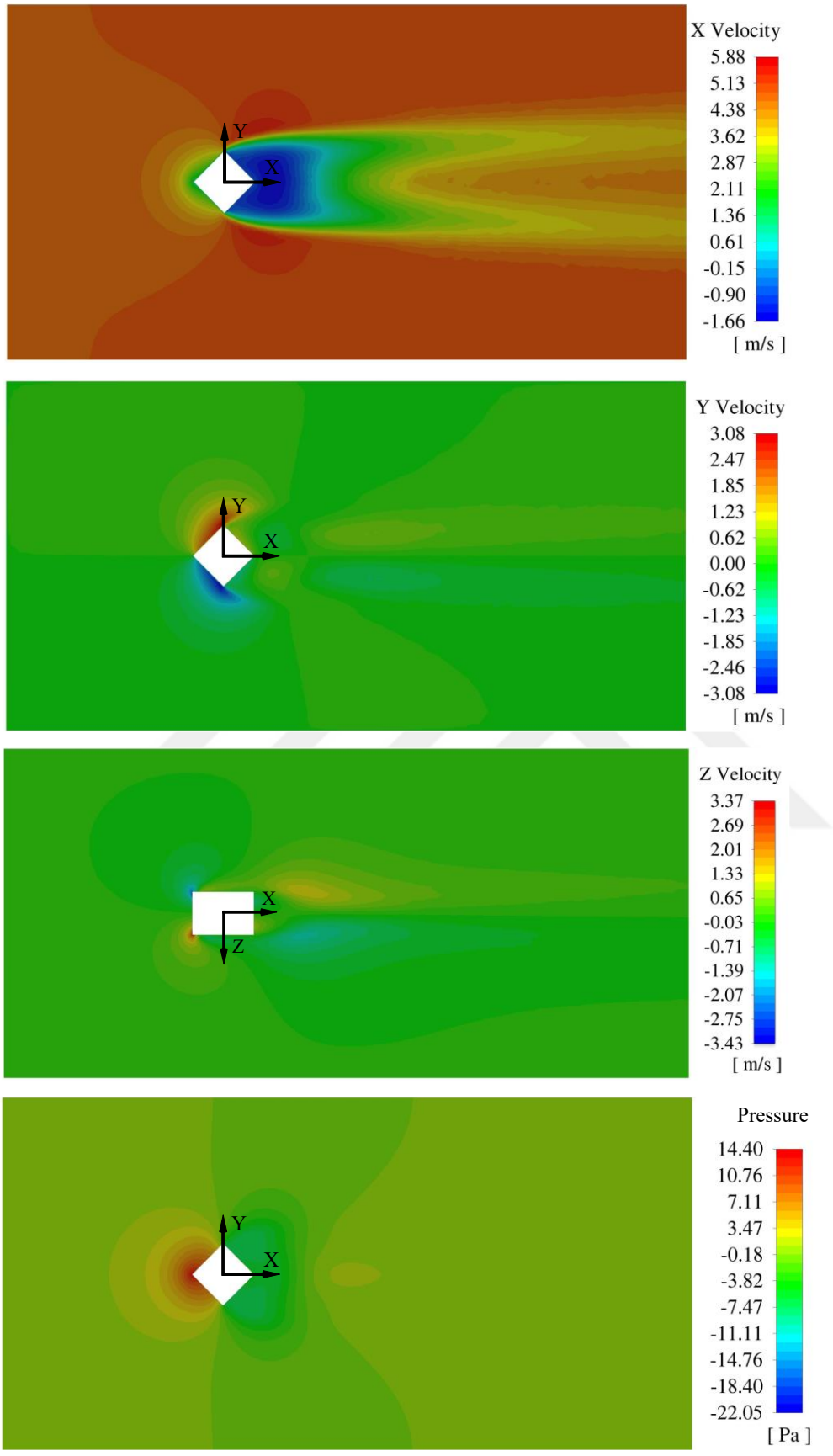


Figure 2.10 : Velocity and pressure contours obtained from numerical simulation. Contours of x and y velocities and pressure on the x-y plane, and z velocity on the x-z plane, intersecting at the geometry's center of mass.

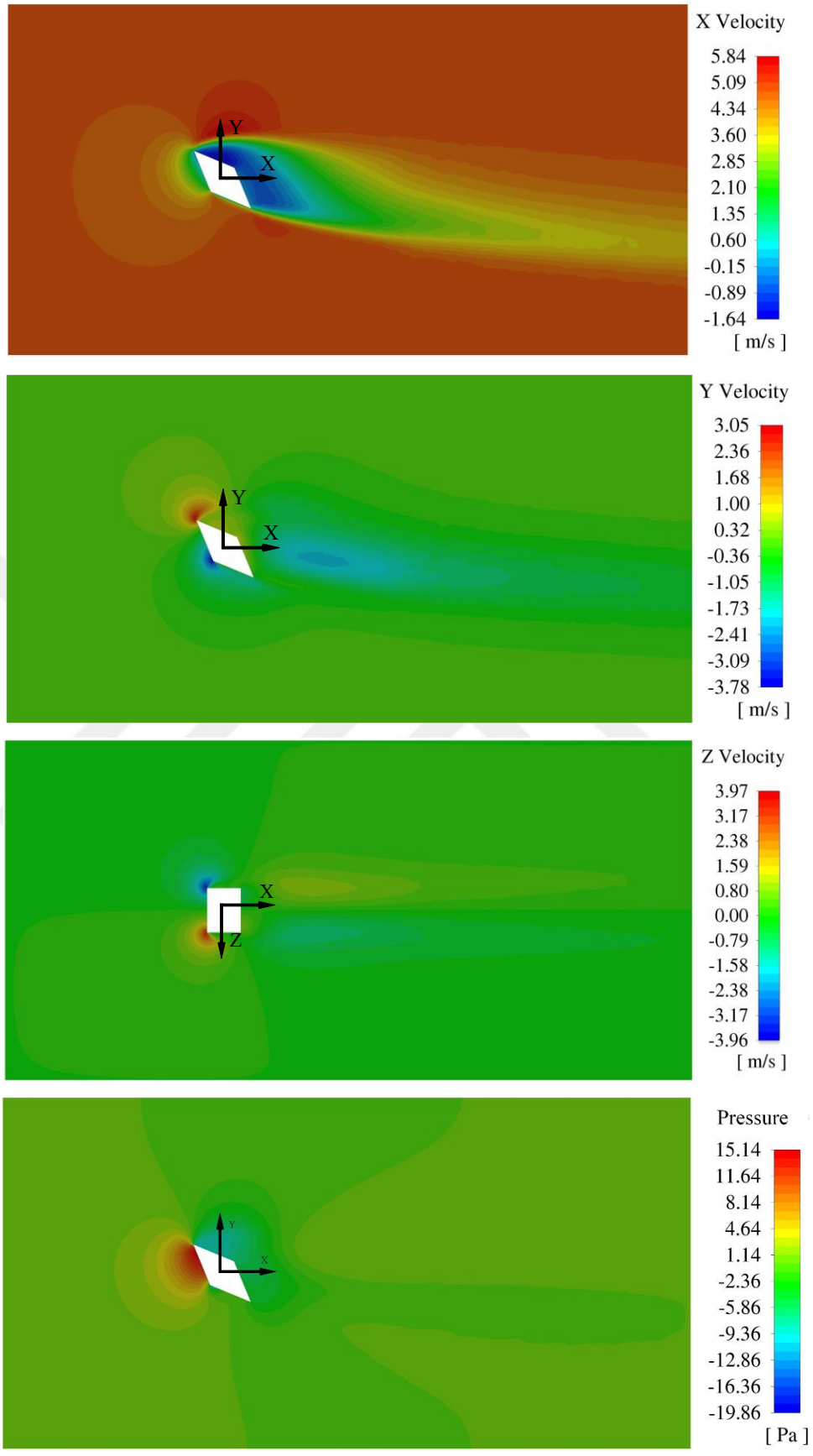


Figure 2.10 (continued): Velocity and pressure contours obtained from numerical simulation. Contours of x and y velocities and pressure on the x-y plane, and z velocity on the x-z plane, intersecting at the geometry's center of mass.

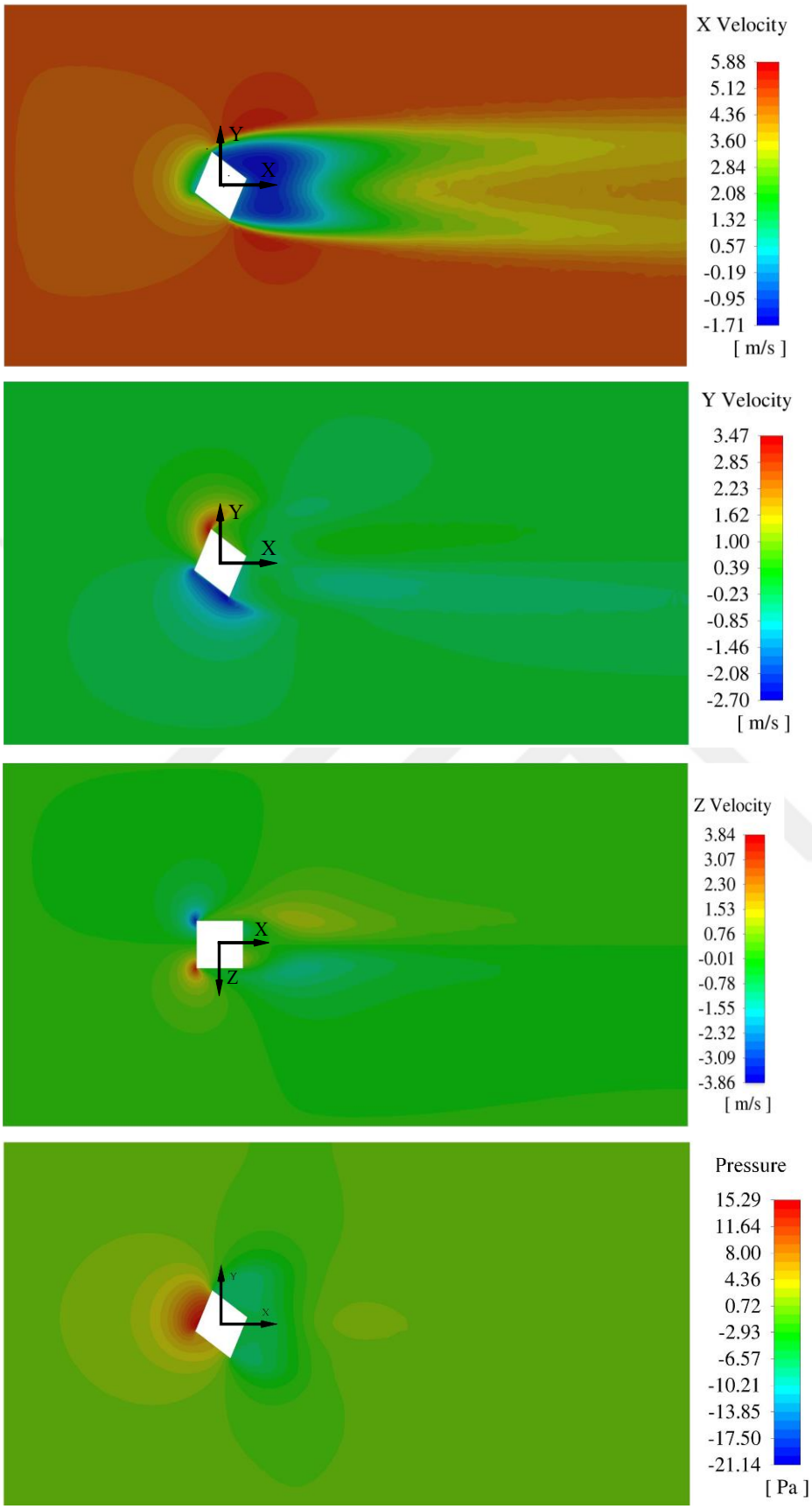


Figure 2.10 (continued): Velocity and pressure contours obtained from numerical simulation. Contours of x and y velocities and pressure on the x-y plane, and z velocity on the x-z plane, intersecting at the geometry's center of mass.

2.3 Data Generation and Preprocessing

2.3.1 Generating ASCII data files

Due to the large number of simulations required to produce the dataset, it is impractical to manually set up and solve each simulation in Fluent. Therefore, in this study, Fluent was coupled with MATLAB using the ANSYS_AAS 1.1.8 (Beta) toolbox and launched in "AAS" mode. To automate the simulation procedure, a MATLAB code and a Text User Interface (TUI) journal file were developed to provide Fluent settings for each case. The simulation process commences with reading the mesh file for each sample, followed by finding the numerical solution of the governing equations, and concludes with exporting and saving the solution results in an ASCII file format. The output file contains the 3D spatial coordinates of all nodes comprising the computational grid system, velocity components in the x, y, and z directions, and pressure values at each node. The goal was to obtain a separate ASCII file containing the CFD solution for each sample. Each file can be considered as a 2D array consisting of 140k to 150k rows and 7 columns. The rows represent vertices in the CFD domain, which constitute the nodes of the grid system, and the number of rows varies according to the unique geometry of each sample. The columns contained the relevant parameters associated with each vertex, as depicted in Table 2.2. The first three columns corresponded to the position of the vertex in the cartesian coordinate system, while the last four columns contained the velocity components and pressure values. To generate the dataset, it is necessary to concatenate all of the arrays into a three-dimensional array that represents the entire dataset. Each layer of the 3D array contains the CFD solution for a specific sample. However, prior to concatenation, it is necessary to carry out some manipulation and preprocessing of the data.

Table 2.2 : An example of Numerical solution data (ASCII file) for a selected sample.

Node number	x-coordinate	y-coordinate	z-coordinate	x-velocity	y-velocity	z-velocity	pressure
1	9.73E-02	6.19E-03	-7.99E-03	3.77E+00	1.59E-01	6.37E-01	8.59E-01
2	9.75E-02	8.97E-03	-9.48E-03	3.55E+00	1.36E-01	6.02E-01	6.55E-01
3	9.53E-02	8.27E-03	-6.30E-03	3.14E+00	2.97E-01	5.02E-01	8.39E-01
4	7.68E-02	-1.78E-02	1.18E-02	1.82E+00	5.91E-02	-3.14E-01	-2.86E-02
5	7.66E-02	-1.90E-02	1.47E-02	2.23E+00	2.34E-01	-3.02E-01	-9.99E-02
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
151035	9.45E-02	-2.08E-02	2.08E-03	1.61E+00	-4.78E-01	-2.12E-02	3.82E-01
151036	8.83E-02	-2.10E-02	-1.32E-02	2.32E+00	1.73E-01	1.19E-01	1.32E-01
151037	7.27E-02	2.16E-02	-5.90E-03	8.83E-01	1.24E-01	3.20E-02	-1.58E-01
151038	7.69E-02	1.17E-02	-1.70E-02	3.73E+00	-2.76E-01	7.38E-01	9.26E-02

The neural network employed in this study takes the spatial coordinate of vertices as input and predicts the corresponding quantities of velocity and pressure as output. Generally, it is not necessary to take the whole CFD domain into account and any area of interest can be taken to train the deep learning framework for that specific subdomain. Furthermore, the computational resources restrict the number of inputs of a neural network framework. Thus, a constraint of 10k was applied to the number of input vertices. The goal is to capture the properties of the flow field around the object and its wake area by considering only 10k of the closest vertices to the object. A cross-section of the CFD domain corresponding to one of the samples on the Z plane is shown in Figure 2.11. It shows a scatter plot of the X velocity throughout the entire CFD domain. In this figure, all of the points in the CFD domain were taken into account. Figure 2.12 illustrates the same sample but only 20k points were taken this time, and by taking 5k points in Figure 2.13, we see that only a tiny area around the object is covered. The distribution of data points near the object's walls is so dense due to the application of the inflation tool in grid generation. Therefore, to represent the desired area by only 10k points we need to drop out some of the points.

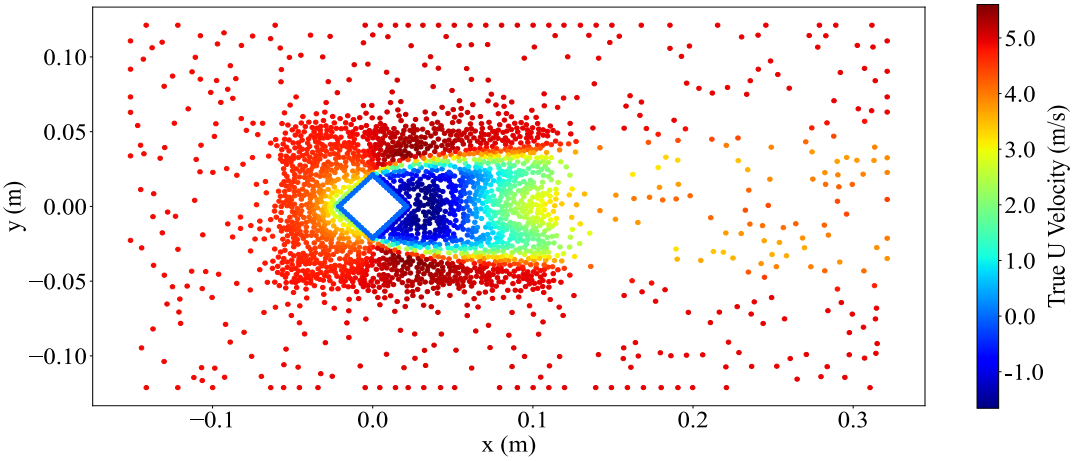


Figure 2.11 : A cross-section view of the entire CFD domain at z = 0 consisting of all the points.

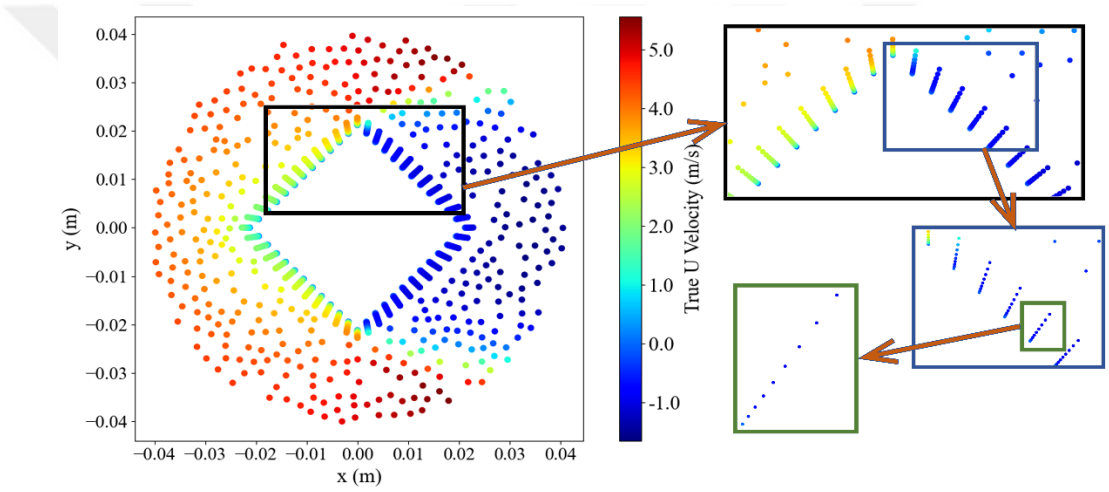


Figure 2.12 : Cross-section view of CFD domain at z = 0 with 20k points that are closer to the center of mass.

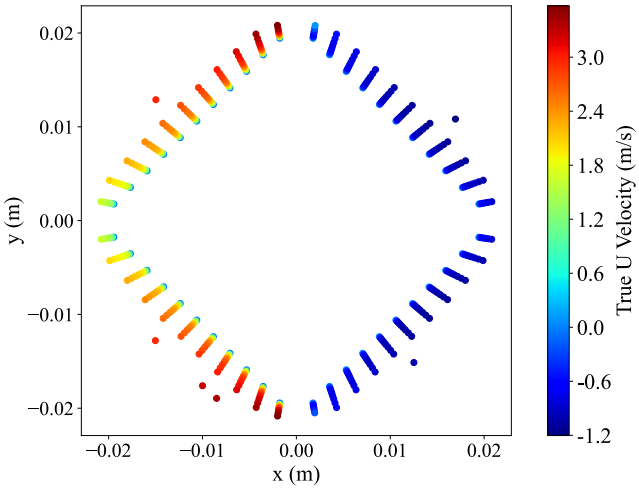


Figure 2.13 : Cross-section view of CFD domain at z = 0 with 5k points.

To achieve the desired outcome, a step-by-step plan of action is followed as outlined below:

1) Sort the rows of the data by x coordinate value in ascending order.
2) Delete odd rows (½ of the entire points of the array will be omitted).
3) Calculate R = $\sqrt{x^2 + y^2 + z^2}$, distance of each point from the center of mass.
4) Sort by R. This arranges the points in order of their distance from the center of the mass of the object.
5) Delete odd rows (½ of the remaining points of the array will be omitted).
6) Cut the extra points of the flow domain and keep only the points that fall within the desired neural network domain which have the following boundaries:

$$-0.07 \leq x \leq 0.14, \qquad -0.05 \leq y \leq 0.05, \qquad -0.05 \leq z \leq 0.05$$

7) Add surface points again to the datasheet (in case they have been omitted during the dropout).
8) Sort by R again.
9) Keep the desired number of closest points to the center of mass and omit the rest.

Figure 2.14 illustrates the same sample in Figure 2.13 after applying the abovementioned procedure with 10k points. Additionally, Figure 2.15 provides a 3D visualization of the same sample in a cross-sectional view.

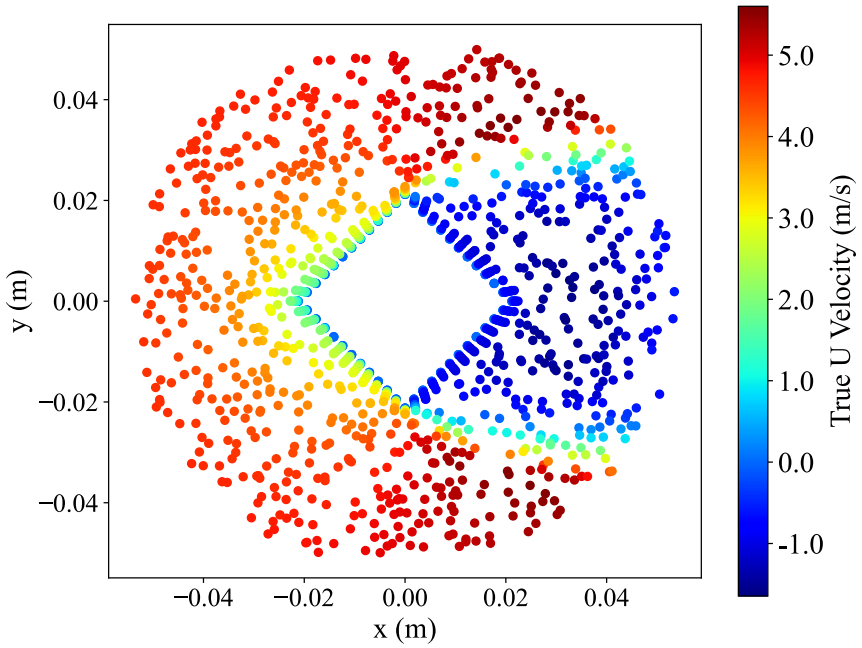


Figure 2.14 : Cross-section view of CFD domain at z = 0 with 10k points after dropping points.

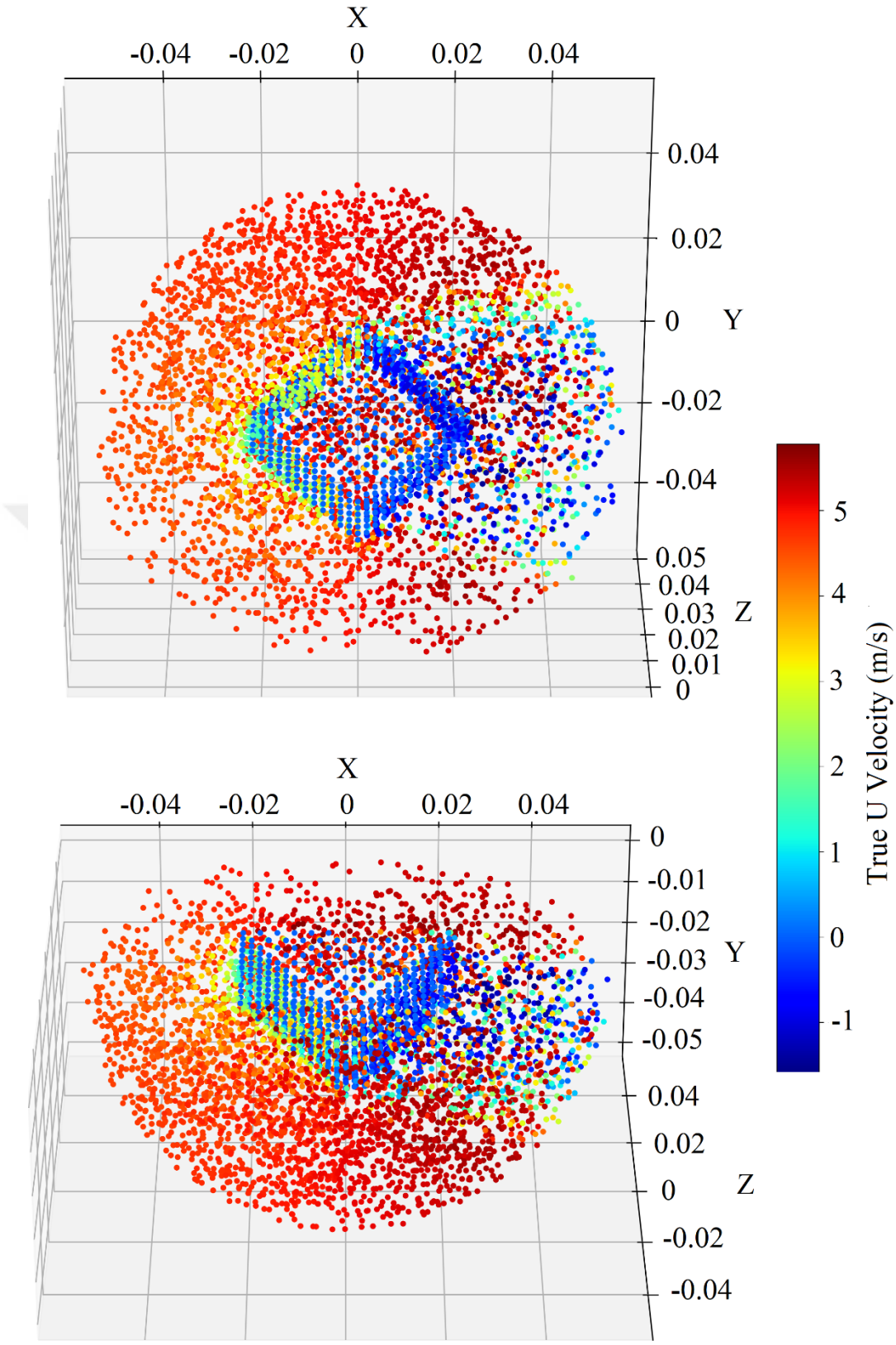


Figure 2.15 : 3D visualization of the CFD domain with 10k points: Cross-sections view at Z = 0 (top figure) and Y = 0 (bottom figure).

2.3.2 Non-dimensionalizing and normalizing the data

The presence of unscaled input variables can result in a sluggish or unstable training process, while unscaled target variables can hinder the convergence of the network in regression tasks. To address these issues, normalization is employed to standardize the features and bring them to a similar scale. This can improve the performance and stability of the model during training. In the current investigation, the output data containing $u, v, w,$ and p have been normalized to expedite the convergence of the training process and to ensure that the velocity and pressure variables contribute equally to the determination of the network parameters. Prior to normalization, the output variables (u, v, w, and p) were made dimensionless by utilizing the following equations:

$$u^* = \frac{u}{u_\infty} \tag{2.6}$$

$$v^* = \frac{v}{u_\infty} \tag{2.7}$$

$$w^* = \frac{w}{u_\infty} \tag{2.8}$$

$$p^* = \frac{p - p_0}{\rho u_\infty^2} \tag{2.9}$$

where u^*, v^*, w^*, and p^* denote the corresponding dimensionless variables, p_0 refers to the atmospheric pressure and u_∞ represents the upstream velocity.

Adjusted min-max method has been utilized to normalize the data, which has been found to result in a higher level of prediction accuracy [38]. In this method, all the output data are normalized between 0.1 and 0.9 using Equation 2.10, where $\emptyset_i'$ indicates normalized data, $\emptyset_i$ denotes the output variable, and $\max(\emptyset)$ and $\min(\emptyset)$, represent the maximum and minimum values of the output variable in the entire dataset, respectively.

$$\emptyset_i' = 0.8 \times \frac{\emptyset_i - \min(\emptyset)}{\max(\emptyset) - \min(\emptyset)} + 0.1 \tag{2.10}$$

In Equation 2.10 $\emptyset$ is replaced by each set of the variables u^*, v^*, w^*, and the p^* to obtain the corresponding set of normalized values u', v', w', and p'.

In this particular case, normalizing the input data (the coordinates) is not required as the data already have a similar scale and range between approximately -0.06 to +0.06. Since the input data already aligns well with the required parameters for training, any potential enhancement in the training process would be minimal. Consequently, the decision is made to retain the input data in their original physical domain, as mentioned in reference [28].

3. NETWORK ARCHITECTURE

Through this chapter, it is intended to provide a thorough understanding of the deep learning network utilized in this study and the steps taken to optimize its performance.

3.1 Network Architecture

In this investigation, a neural network architecture has been developed to establish a mapping from the coordinates of unstructured nodes in a 3D grid system, as input, to the corresponding values of velocity components and pressure at related nodes, resulting in the distribution of velocity and pressure in the flow field around the object.

This can be mathematically formulated as a regression problem of the form $\mathcal{F}(\mathcal{X}) = \mathcal{Y}$, where $\mathcal{X}$ represents a set of N=10,000 points denoted as $\mathcal{X} = \{x_i \in \mathbb{R}^d\}_{i=1}^N$. Here, d=3 denotes the dimension of the input data. The corresponding flow field values are represented by $= \{y_i \in \mathbb{R}^D\}_{i=1}^N$, where D=4 is the dimension of the output data. For a given input dataset $\mathcal{X}$ we aim to find output $\mathcal{Y}$, which is achieved using a machine-learning model, G, that acts as a nonlinear regression function approximating the function $\mathcal{F}$. The model can be represented as $G(\mathcal{X}, W) \approx \mathcal{Y}$, where W is a matrix of trainable weights that are optimized during the training phase through direct supervision of CFD data. The objective of this optimization process is to minimize the difference between the output of G and the ground truth $\mathcal{Y}$, aiming to achieve a high degree of similarity.

The deep learning model developed in this study is founded on the structure proposed by Kashefi et al [30], which is a modification of the PointNet architecture introduced by Qi et al. [39]. This architecture was initially presented in 2017 as a promising method for 3D computer vision applications, such as shape segmentation and classification from point clouds. PointNet has been specifically designed to handle unprocessed point cloud data and does not require any pre-processing steps like converting the data into regular grids or voxel representations. This structure is simple and easy to implement while maintaining a reasonable degree of accuracy on tasks involving shape segmentation and classification. Kashefi et al. [30] made

modifications to the segmentation architecture of PointNet and employed the mean squared error as the loss function to optimize the trainable parameters. They employed their network to acquire the distribution of pressure and velocity in a 2D external flow but the framework was potentially capable of handling 3D flows as well. Thus, their framework is modified in this study to predict 3D flow properties.

CNNs are commonly used for structured input data such as image grids or 3D voxels. However, processing point cloud data poses a challenge due to its inherent nature. In contrast to pixel arrays or voxel arrays, a point cloud is characterized by an unordered collection of points without any specific arrangement or structure. Moreover, neighboring points that interact with each other in the physical domain may not be considered neighbors in the input matrix, depending on how the matrix is sorted. To address these properties of point clouds, it is crucial for the network to generate output that remains unchanged regardless of the order in which the input points of the point cloud are presented. This is particularly important in CFD applications, where the network must acquire relevant information throughout the set of points to deduce the object's geometry and location, which ultimately identify the flow fields. Furthermore, it is important to note that the representation obtained from the set of points should not be affected by specific transformations. For instance, in object classification or segmentation, rotating or translating all points should not affect the point cloud's category or segmentation. Similarly, when using PointNet for fluid flow prediction, the acquired representation of the set of points should remain consistent despite specific transformations such as global translations and rotations of the input point cloud, since these transformations do not affect the underlying fluid flow physics. In other words, the velocity distribution at a particular point in the fluid domain should be the same regardless of the orientation or position of the point cloud. Figure 3.1 depicts the design of the original PointNet by Qi et al. [39], which comprises three primary modules: a max pooling layer, a concatenation structure, and two joint alignment networks. The functionality of each module is discussed below.

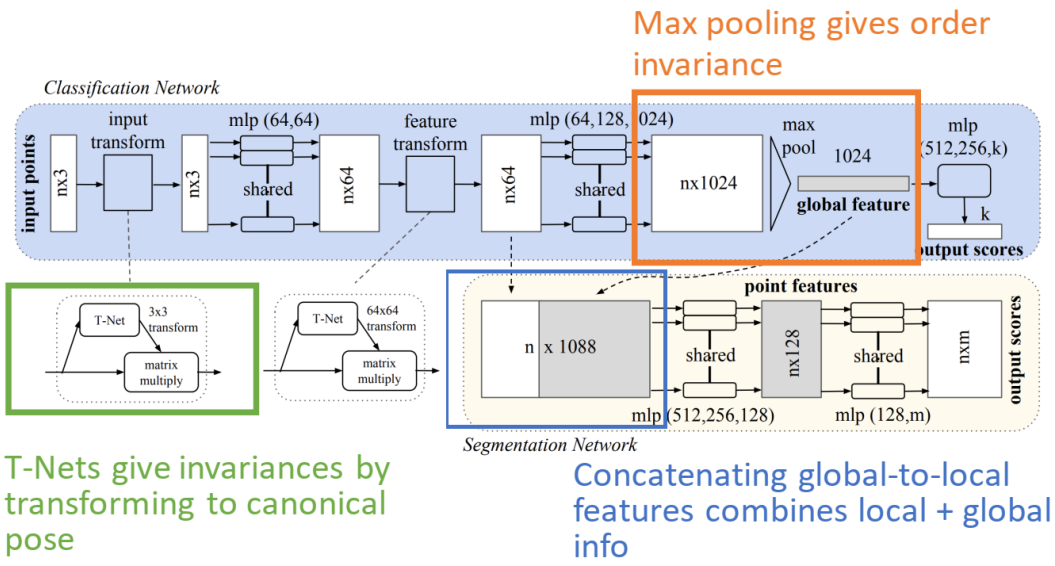


Figure 3.1 : Architecture of the original PointNet network [39].

3.1.1 Max pooling layer

To attain order invariance in point cloud processing, incorporating a max pooling layer proves advantageous as a fundamental symmetric function for gathering information from individual points. A symmetric function accepts n vectors as input and produces a new vector that remains invariant regardless of the input order. This can be expressed mathematically as follows:

$$g\{(x_1.,,,.x_n)\} \approx S(m(x_1).,,,.m(x_n)) \tag{3.1}$$

In PointNet, a symmetric function S is utilized to process the transformed elements within the point set and approximate the general function g. The function m is estimated using an MLP network, while g is represented as a combination of a single variable function and a max pooling function. This approach effectively addresses the challenge of permutation invariance within the model. The resulting output of the max pooling operation is a vector referred to as "global," which represents a latent description of the point cloud. The top part of the structure depicted in Figure 3.1 is designed to implement the above equation.

3.1.2 Concatenation structure

As previously stated, the output of the max pooling layer forms a vector which is a latent representation of the flow field. In other words, it is a compressed representation of the flow field data that captures the underlying structure or patterns in the data. This

is lower-dimensional latent space that has been obtained by encoding the flow field. To establish a bijection map between each input point x_i and its corresponding output properties y_i, a combination of local and global knowledge is required. After computing the global point cloud feature vector, it is concatenated with each point, creating a combined point feature. By exploiting the combined feature, the model is able to extract updated per-point features, enabling a comprehensive representation that incorporates both local and global information. This integration empowers the per-point features to capture not only the local geometry but also the global semantics or context. As a result, the network becomes capable of predicting per-point quantities that depend on both the immediate surroundings and the broader contextual understanding of the point cloud.

3.1.3 Joint alignment networks

As discussed earlier, achieving transformation invariance for the acquired representation of the set of points is crucial, especially when dealing with Euclidean or rigid transformations. To achieve this, Qi et al. [39] have exploited T-Nets in their invented architecture PointNet. T-Nets refer to transformation networks that are responsible for learning and applying an affine transformation to the input point cloud, which serves to align the point cloud to a standard orientation or canonical space. T-nets are mini PointNets that resemble the main network, comprising basic modules of the main network, such as fully connected layers (MLPs) and a max pooling layer. The first T-net in the PointNet, called Input Transform T-Net, takes a set of point cloud coordinates (i.e., the point cloud itself) as input and outputs a transformation matrix. This transformation matrix is then applied to the input point cloud using a matrix multiplication operation to align it to a standard orientation. T-nets can be trained end-to-end with the rest of the PointNet architecture, allowing the network to learn the best transformation for the given task.

The PointNet architecture employs a second T-Net, called the Feature Transform T-Net, which learns a linear transformation for each individual feature channel of the intermediate feature representation obtained by the PointNet encoder. This allows the network to capture the relative orientation of the features and be invariant to deformations and irregularities within the point cloud. The output of the Feature Transform T-Net is then applied to the intermediate feature representation to obtain a transformed feature representation that is invariant to small perturbations and

deformations of the input point cloud. The transformation matrix in the feature space possesses a greater dimensionality compared to the spatial transformation matrix, as it is a 64 by 64 matrix in the original PointNet architecture, whereas the input transformation matrix is only 3 by 3.

3.1.4 Modified network architecture

The architecture of our network is depicted in Figure 3.2. Similar to Kashefi et al. [30], the segmentation structure of PointNet was utilized. However, certain modifications were made to the structure, such as adjusting the number of layers and nodes in each layer, to accommodate a larger number of inputs required for the prediction of 3D flow properties.

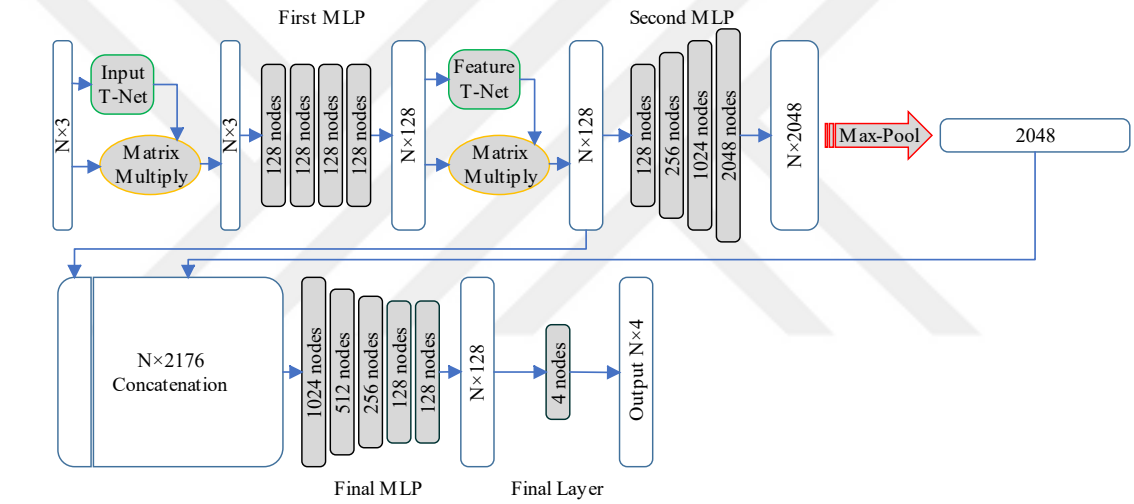


Figure 3.2 : Architecture of the main network.

The input to the PointNet architecture is an N×3 array representing each sample. Here, N is set to be equal to 10,000 which corresponds to the number of points representing the physical domain, and 3 denotes the coordinate of the point in the x, y, and z directions. The first step in the PointNet architecture is to learn the input transformation matrix that aligns the input point cloud with a canonical coordinate system, which is achieved using the input T-Net. The structures of the T-Nets will be described in the upcoming paragraphs, following the main structure explanation.

The input data array is multiplied by the transformation matrix, a 3 by 3 matrix, and the transformed point cloud is then passed through the first shared MLP. This MLP consists of four fully connected layers, each with 128 neurons. These layers are applied to each point in the transformed point cloud data independently, which results in a set

of point-wise features that capture local information about each point in the input point cloud.

The second T-Net is utilized to learn the feature transformation matrix, which is a 128 by 128 matrix in our architecture. The output of the first shared MLP, an N×128 array, is multiplied by the feature transformation matrix and passed through the second MLP. This MLP comprises four fully connected layers with 128, 256, 1024, and 2048 neurons, respectively.

Next, the point-wise features are aggregated using a max pooling operation that takes the maximum value across all points in the set, resulting in a single global feature vector that summarizes the entire input point cloud. A copy of the global feature vector obtained from the max pooling layer is concatenated with each point-wise feature. This results in an array of size N by 2176.

The combined feature is passed through a final shared MLP, resulting in the prediction of CFD quantities with dimensions N×P. In this study, P is equal to 4, including the velocity components in the x, y, and z directions and the pressure. The final MLP consists of five fully connected layers, followed by a final layer that outputs the results of the prediction. The number of neurons in each layer is 1024, 512, 128, 128, and P, respectively.

In the main structure, the ReLU activation function and batch normalization are employed in all layers except the final output layer. The final layer benefits from a sigmoid activation function and no normalization.

T-Nets, as discussed before, are mini PointNets. Both T-Nets share the same architecture, but their output is different due to the last layer's size. Figure 3.3 depicts the architecture of the T-Nets.

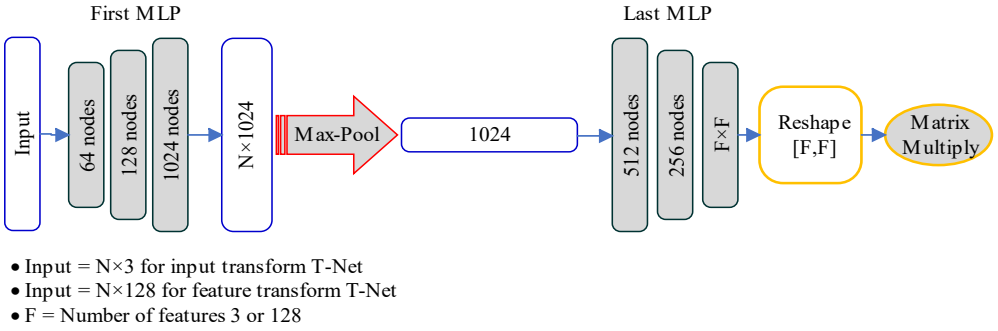


Figure 3.3 : Architecture of the T-Nets.

The input data passes through an MLP consisting of three shared layers, with 64, 128, and 1024 neurons in each layer, respectively, as shown in the diagram. A max pooling layer is then applied, followed by another MLP with two layers of sizes 512 and 256 to process the data further. Finally, the last layer of each transformation net has a different size, with 3^2 in the first T-Net and 128^2 in the second T-Net. To ensure that this transformation does not modify the input points at the beginning, the output matrix is initialized to an identity matrix. Furthermore, an L2 regularizer with a weight of 0.001 is added to the last layer. The activity regularizer adds a penalty term to the loss function to encourage the weights in the dense layer to be small, which helps prevent overfitting and improves generalization. Finally, as the output of the last layer is a vector, it is reshaped to a matrix before the multiply operation.

Similar to the main structure, the ReLu activation function and batch normalization are employed in all layers except for the last layer of the T-Nets, which has no activation function or normalization.

3.2 Training

There are 11,231,821 trainable parameters comprising weights and biases in the designed model. The training process involves optimizing these parameters to minimize the objective or loss function. In this study, Mean Squared Error (MSE) has been selected as the loss function, which is frequently employed in regression problems. It calculates the mean squared difference between the predicted and actual values, making it a suitable option when it is necessary to penalize greater errors more severely than minor errors. The formulation of the loss function for our problem is as follows:

$$\mathcal{L} = \frac{1}{4 \times N}\left(\sum_{i=1}^{N} [(u_i' - \tilde{u}_i')^2 + (v_i' - \tilde{v}_i')^2 + (w_i' - \tilde{w}_i')^2 + (p_i' - \tilde{p}_i')^2]\right) \quad (3.2)$$

where u', v', w', and p' represent normalized values of velocity and pressure computed by CFD solver and $\tilde{u}'$, $\tilde{v}'$, $\tilde{w}'$, and $\tilde{p}'$ are the predicted values obtained from the network.

Adaptive Moment Estimation (Adam) optimizer algorithm is chosen for model training. Adam is an enhanced version of stochastic gradient descent (SGD) that incorporates adaptive learning rates for individual parameters. The optimizer's parameters are set as follows: $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$, and $\hat{\epsilon} = 10^{-6}$, where β_1 and β_2 are the exponential decay rates for the first and second moment estimates, respectively. $\hat{\epsilon}$ is a very small number added to the denominator of the update rule to avoid division by zero during implementation.

As outlined in Section 2.1, two datasets were created, containing 961 and 3511 samples, respectively. After shuffling the samples, to prevent any order or pattern in the data from affecting the training process, we partitioned the data for both datasets into training, validation, and testing subsets with proportions of 80%, 10%, and 10% for the smaller dataset and 65.5%, 20.5%, and 14% for the larger dataset, respectively.

The implementation, training, and testing of the neural network were carried out using the TensorFlow framework and the Keras library in this study.

The training process was carried out utilizing the resources provided by UHem, the National Center for High-Performance Computing located at Istanbul Technical University. The employed machine for training was equipped with 128 cores of AMD EPYC™ 7742 with a clock speed of 2.25GHz and 256 GB of RAM. It's worth noting that a CPU-based training approach was employed in this study.

3.3 Hyperparameters Optimization

The batch size is a crucial hyperparameter in machine learning, as it dictates the number of training examples utilized in each iteration during the training process. Its value affects the training speed, stability, and quality of the resulting model. The choice of batch size has been determined based on careful consideration of computational limitations and training stability. Specifically, to avoid divergence

during the training phase, smaller batch sizes were deemed inadequate, while larger batch sizes were constrained by computational resources. Consequently, a batch size of 80 has been selected as an appropriate compromise.

To optimize the other hyperparameters of the deep learning model such as initial learning rate, number of layers in each MLP, and number of neurons in different layers, a sequential model-based approach known as Bayesian optimization was employed. The method initially selects a few hyperparameters at random and then, based on their performance, selects the next best set of hyperparameters. The approach utilizes information derived from previous evaluations to guide the search toward more promising regions. Keras library provides a package called Keras tuner, which includes the implementation of several optimization methods, including the Bayesian method.

To define the search area in the Bayesian method, certain constraints are applied to the number of layers and neurons in each MLP. In the first MLP, the number of layers is limited to a range of 2 to 4 layers, and the number of neurons per layer can be one of the following: 64, 128, 256, or 512. Moving on to the second MLP, the layers preceding the last layer (just before the max pooling) can vary between 2 and 4 layers, and the number of neurons per layer can be selected from the options of 64, 128, 256, 512, or 1024.

A distinct search area is assigned to the last layer in this MLP, as its neuron range begins with larger numbers: 1024, 2048, 3072, or 4048 neurons per layer. In the final MLP, a similar strategy is employed as in the second MLP. A separate search area is designated for the first layer (immediately following the concatenation) due to its neuron range starting from larger numbers: 512, 1024, 2048, or 3072 neurons per layer. For the subsequent layers, the number of layers is constrained to change between 3 and 6 layers, and the number of neurons per layer can be chosen from the options of 64, 128, 256, 512, or 1024. Additionally, constraints are imposed on the search area for initial learning rates, including values such as 5×10^{-4}, 1×10^{-4}, and 1×10^{-5}. A summary of the search areas for each MLP and the initial learning rate is provided in Table 3.1.

Table 3.1 : Search areas for each MLP and initial learning rate.

Item	Searching Area	
	Number of Layers	Number of neurons per Layer
First MLP	2 to 4	64, 128, 256, 512
Second MLP	2 to 4	64, 128, 256, 512, 1024
Second MLP (Last Layer)	1	1024, 2048, 3072, 4048
Final MLP (first layer)	1	512, 1024, 2048, 3072
Final MLP	3 to 6	64, 128, 256, 512, 1024
Initial Learning Rate	5×10^{-4}, 1×10^{-4}, and 1×10^{-5}	

Due to the extensive number of possible combinations for different layers and neuron numbers, limitation is imposed on the maximum number of trials, restricting it to 80 tests. Additionally, the maximum number of epochs was set to 50. The best combination is determined based on the achieved smallest validation loss value within the 50 epochs and was subsequently chosen as the network architecture.

Due to the limitation of 50 epochs per trial in the optimization step, the determined optimum learning rate was employed as the initial learning rate in the real training step. The training process began with a learning rate of 5×10^{-4} and continued until the validation loss started to diverge. At that point, the learning rate was reduced to 2×10^{-4} to facilitate the attainment of smaller validation loss values. The training process spanned 5000 epochs for smaller dataset (961 samples) and 1400 epochs for larger dataset (3511 samples). Figure 3.4 and Figure 3.5 depict the training history for small and large datasets, respectively. These figures illustrate the overall trends in the learning process. However, in order to offer a clearer insight into the learning process, learning curves are presented for specific checkpoints, specifically at the epochs where the minimum loss values occurred within every 200 epochs. This presentation can be observed in Figure 3.6 for the smaller dataset and in Figure 3.7 for the larger dataset. The larger dataset demonstrated a faster convergence speed in reaching the same loss values compared to the smaller dataset. However, due to the increased computational burden and the absence of further improvement in the convergence, it was unavoidable to terminate the training at this stage. The final training, validation, and test loss values are presented in Table 3.2.

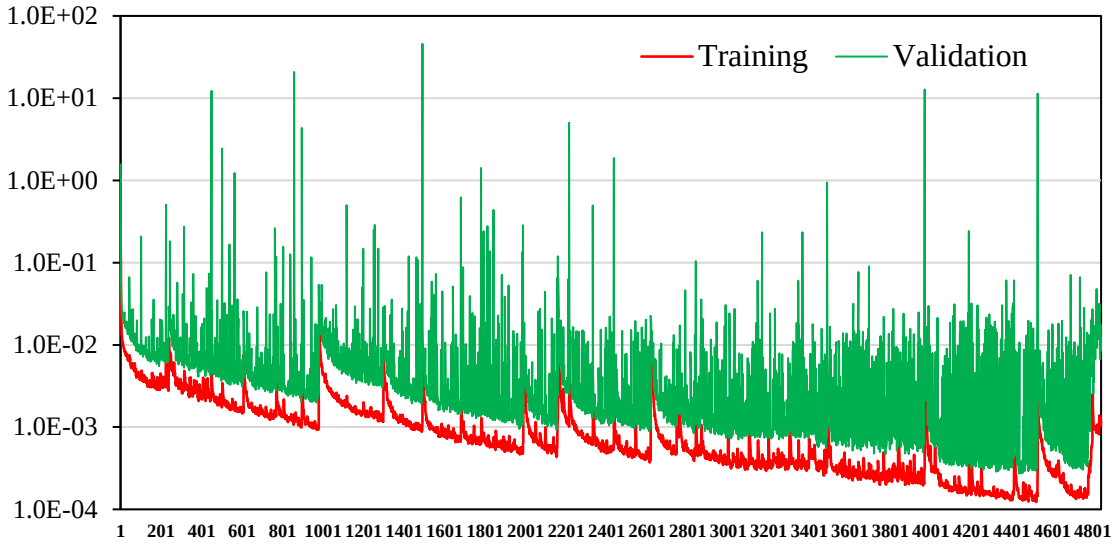


Figure 3.4 : Original training curve of the dataset with 961 cases.

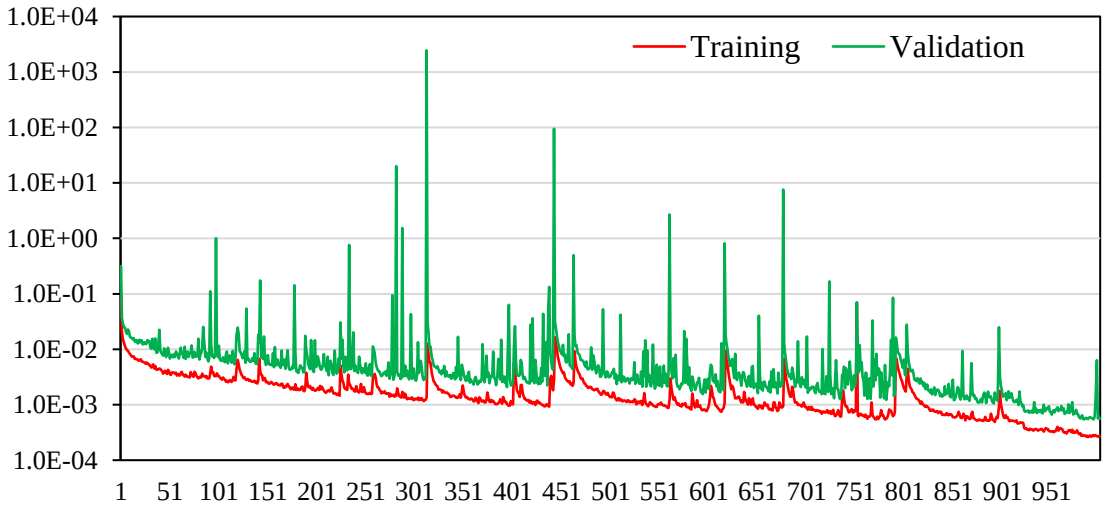


Figure 3.5 : Original training curve of the dataset with 3511 cases.

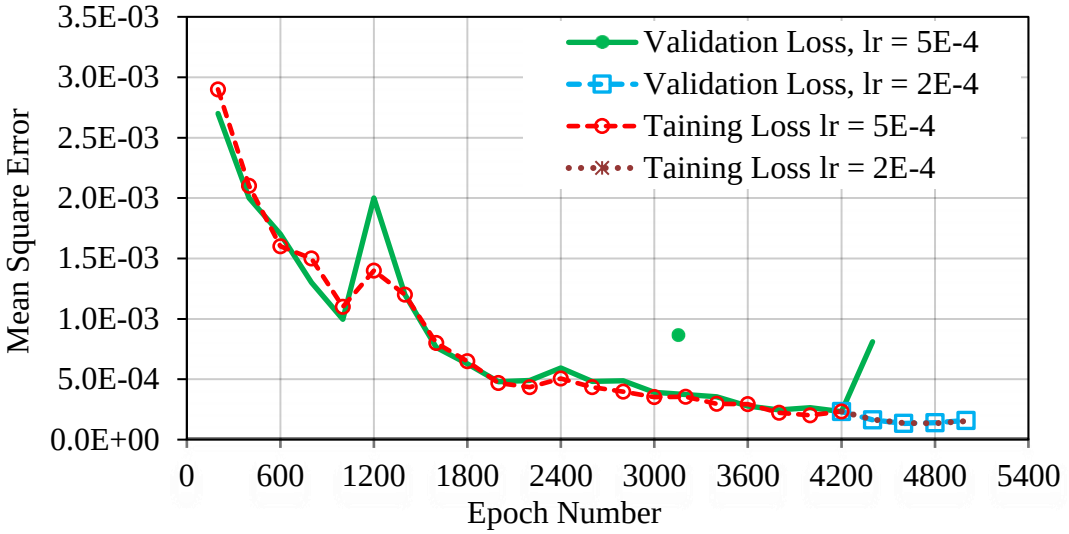


Figure 3.6 : Training curve of the dataset with 961 samples plotted at Checkpoints (lr refers to learning rate).

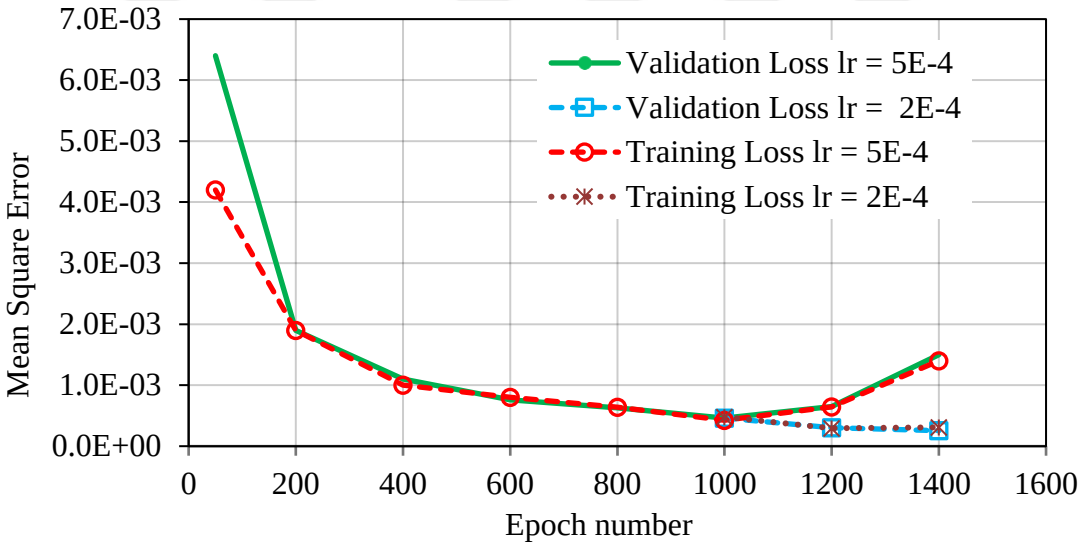


Figure 3.7 : Training curve of the dataset with 3511 samples plotted at Checkpoints (lr refers to learning rate).

Table 3.2 : The final value of training, validation, and test loss.

Parameters	dataset with 961 samples	dataset with 3511 samples
Training Loss	1.25×10^{-4}	3.08×10^{-4}
Validation Loss	1.34×10^{-4}	2.54×10^{-4}
Test Loss	1.38×10^{-4}	2.59×10^{-4}

4. RESULT AND DISCUSSIONS

In this section, an evaluation of error measurement is carried out. Subsequently, a comparison is made between CFD results and predicted results, encompassing distribution contours and error contours.

Note: All analyses presented in this chapter are conducted on the test subsets of both the small and large datasets.

4.1 Model Evaluation

To evaluate the performance of the deep learning model, the following error formulations are considered. Equation 4.1 calculates point-wise Relative Error Percentage (REP). In this equation ∅ is replaced by u, v, w, or p values from CFD data or predicted results to find the relative error percentage in each node.

$$\mathrm{REP}_{\emptyset_i} = \frac{\emptyset_{CFD_i} - \emptyset_{Pre_i}}{\emptyset_{CFD_i}} \times 100 \tag{4.1}$$

Equation 4.2 calculates the Average Relative Error Percentage ($\mathrm{AREP_s}$) that is average amount of relative error percentage for a parameter in one sample, where n is the number of points in a sample that equals 10,000.

$$\mathrm{AREP}_s = \frac{\sum_{i=1}^{n} \mathrm{REP}_{\emptyset_i}}{n} \tag{4.2}$$

Equation 4.3 calculates the Average Relative Error Percentage at the entire test set ($\mathrm{AREP_t}$).

$$\mathrm{AREP}_t = \frac{\sum_{j=1}^{m} \left(\sum_{i=1}^{n} \mathrm{REP}_{\emptyset_{ji}} \right)}{m \times n} \tag{4.3}$$

where n is the number of points in each sample and m is the number of samples in the test subset that is equal to 96 and 490 for the smaller and larger datasets respectively. Table 4.1 and Table 4.2 present these errors individually for four output parameters, corresponding to the smaller and larger datasets, respectively. The first and second rows represent the worst and best case in the test subset which have the largest and

smallest value of $AREP_s$. For example, as shown in Table 4.1, the average relative error for the U velocity ranges from 1.89% in the worst case to 1.07% in the best case. From these tables, it can be concluded that the amount of average error in the entire test subsets is close to the amount of error in the best cases. This means that most of the samples have a relative error value close to the minimum value.

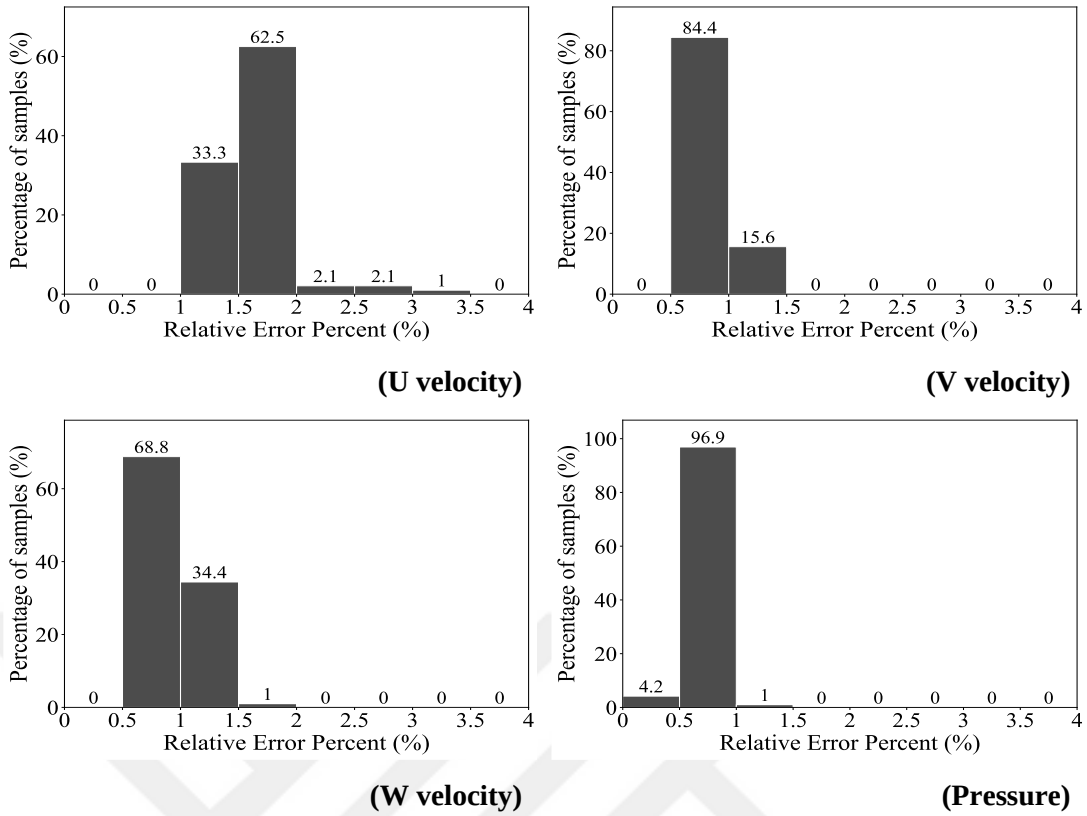
Table 4.1 : Relative error percentage in the test subset for smaller dataset.

Parameters	**u**	**v**	**w**	**P**
Max (worst case)	1.89 %	1.04 %	1.31 %	1 %
Min (best case)	1.07 %	0.63 %	0.64 %	0.5 %
Average (whole test set)	1.27 %	0.78 %	0.79 %	0.6 %

Table 4.2 : Relative error percentage in the test subset for larger dataset.

Parameters	**u**	**v**	**w**	**P**
Max (worst case)	4.58 %	2.92 %	3.08 %	3.04 %
Min (best case)	1.3 %	0.64 %	0.63 %	0.54 %
Average (whole test set)	1.81 %	1.11 %	1.09 %	0.82 %

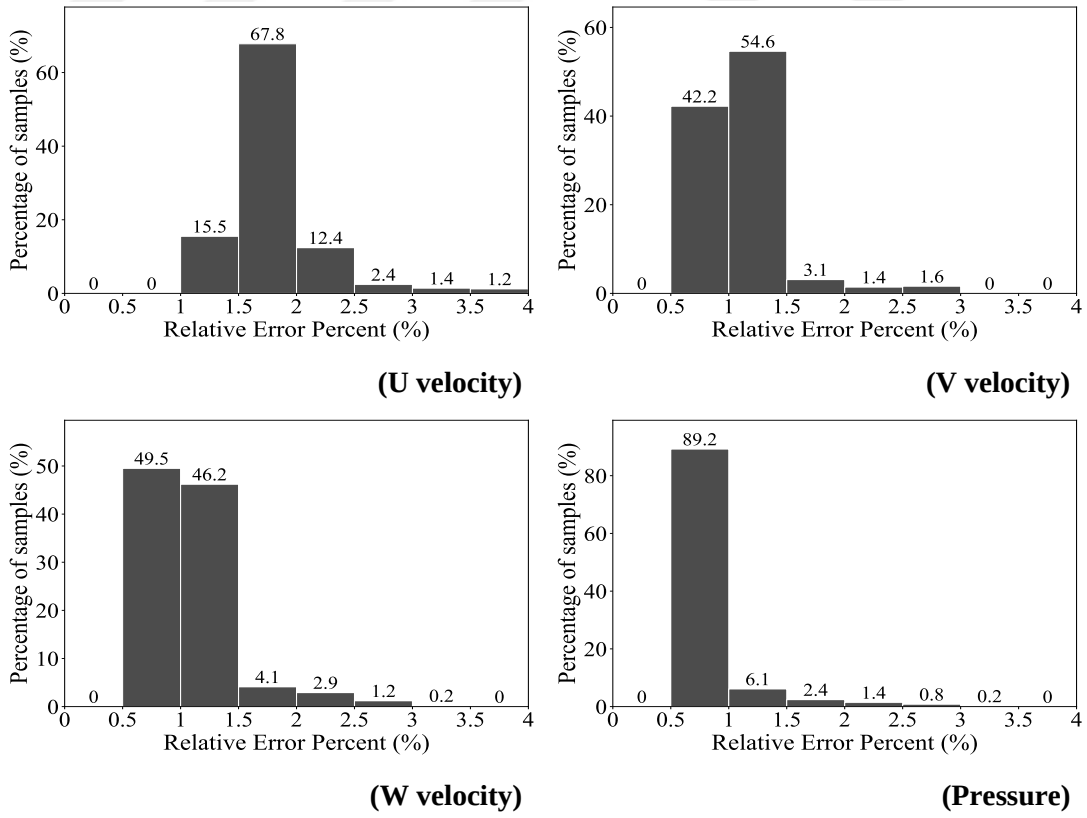
Figure 4.1 and Figure 4.2 illustrate the distribution of average relative error percentage in the test subset for small and large datasets respectively. For instance, as depicted in Figure 4.1 for the smaller dataset, nearly 33% of the samples in the test subset exhibit a relative error value between 1% and 1.5% for the U velocity, and also almost 62% of the samples have an error value between 1.5% and 2%. For V velocity, almost 84% of the samples in the test subset have a relative error value between 0.5% and 1%.



(U velocity) **(V velocity)**

(W velocity) **(Pressure)**

Figure 4.1 : Relative error distribution (in percentages) in the test subset for the smaller dataset.



(U velocity) **(V velocity)**

(W velocity) **(Pressure)**

Figure 4.2 : Relative error distribution (in percentages) in the test subset for the larger dataset.

4.2 Samples of Predicted Cases

This section will feature the presentation of three scenarios from the test subset, each highlighting the best, average, and worst predicted outcomes for the smaller dataset. Similarly, for the larger dataset, two scenarios will be presented, portraying the best and worst predicted cases.

Figure 4.3, Figure 4.4, and Figure 4.5 display the comparison between numerical simulation and prediction results generated by PointNet for U, V, and W velocities, as well as pressure specifically for the smaller dataset. Additionally, these figures present the coefficient of determination, R^2, for the best, average, and worst predicted cases. For both the best and average cases, most of the points are concentrated around y = x lines, implying that the network is highly accurate.

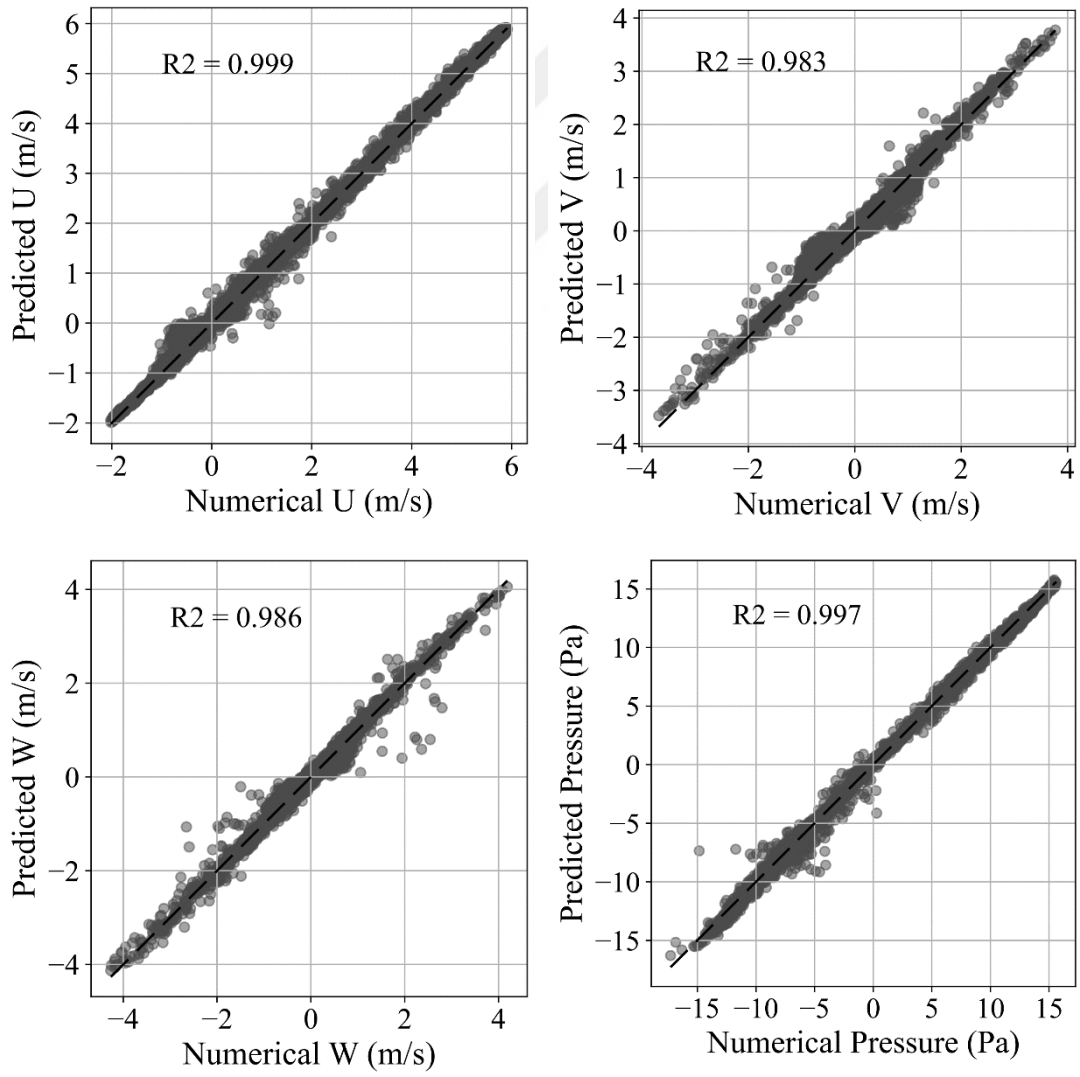


Figure 4.3 : Comparison of numerical simulation and PointNet prediction and R^2 for the sample number 718, best case (small dataset).

Moreover, there is not a very significant difference between the two cases. However, in the worst case, the accuracy is low. It is worth noting that there are always a few exceptional cases with higher errors.

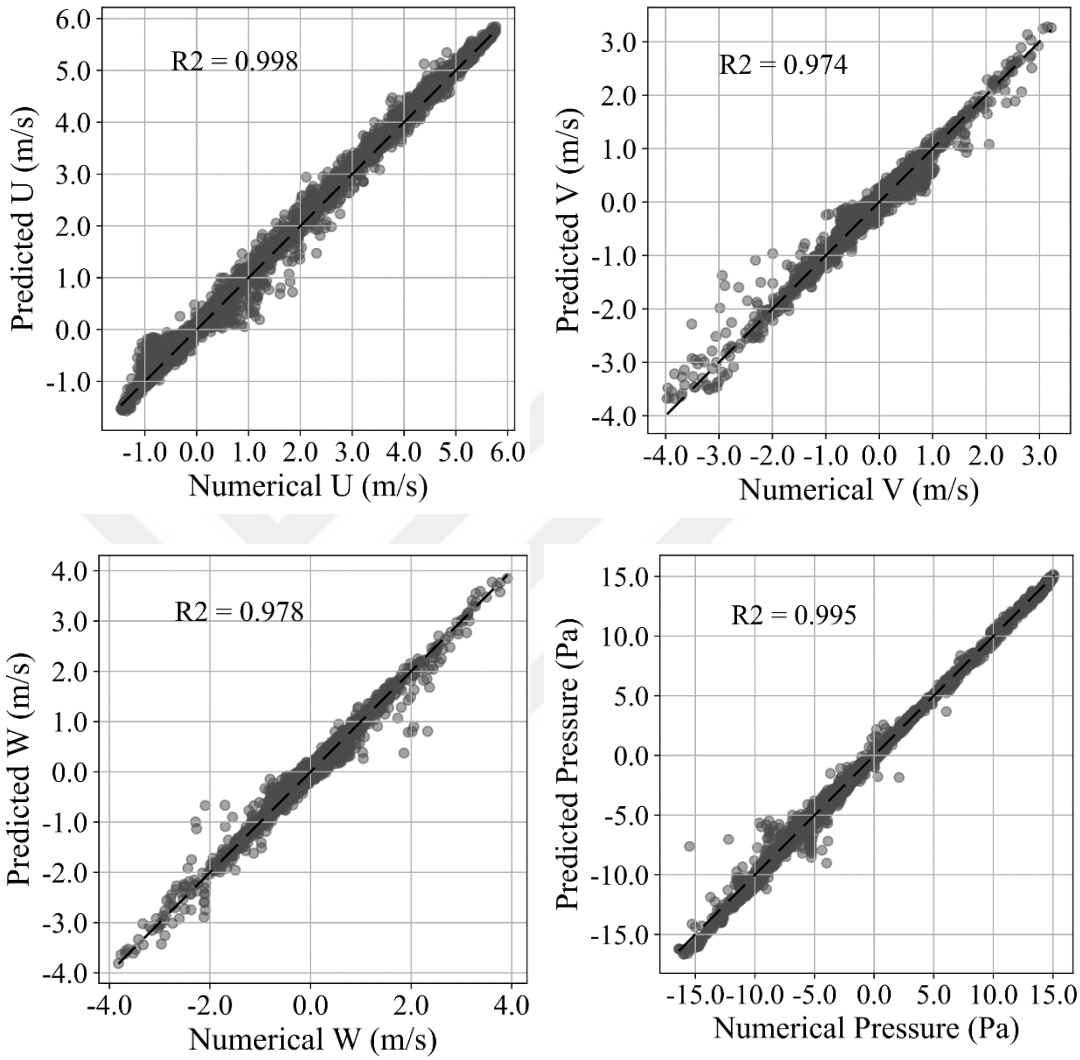


Figure 4.4 : Comparison of numerical simulation and PointNet prediction and R^2 for the sample number 318, average case (small dataset).

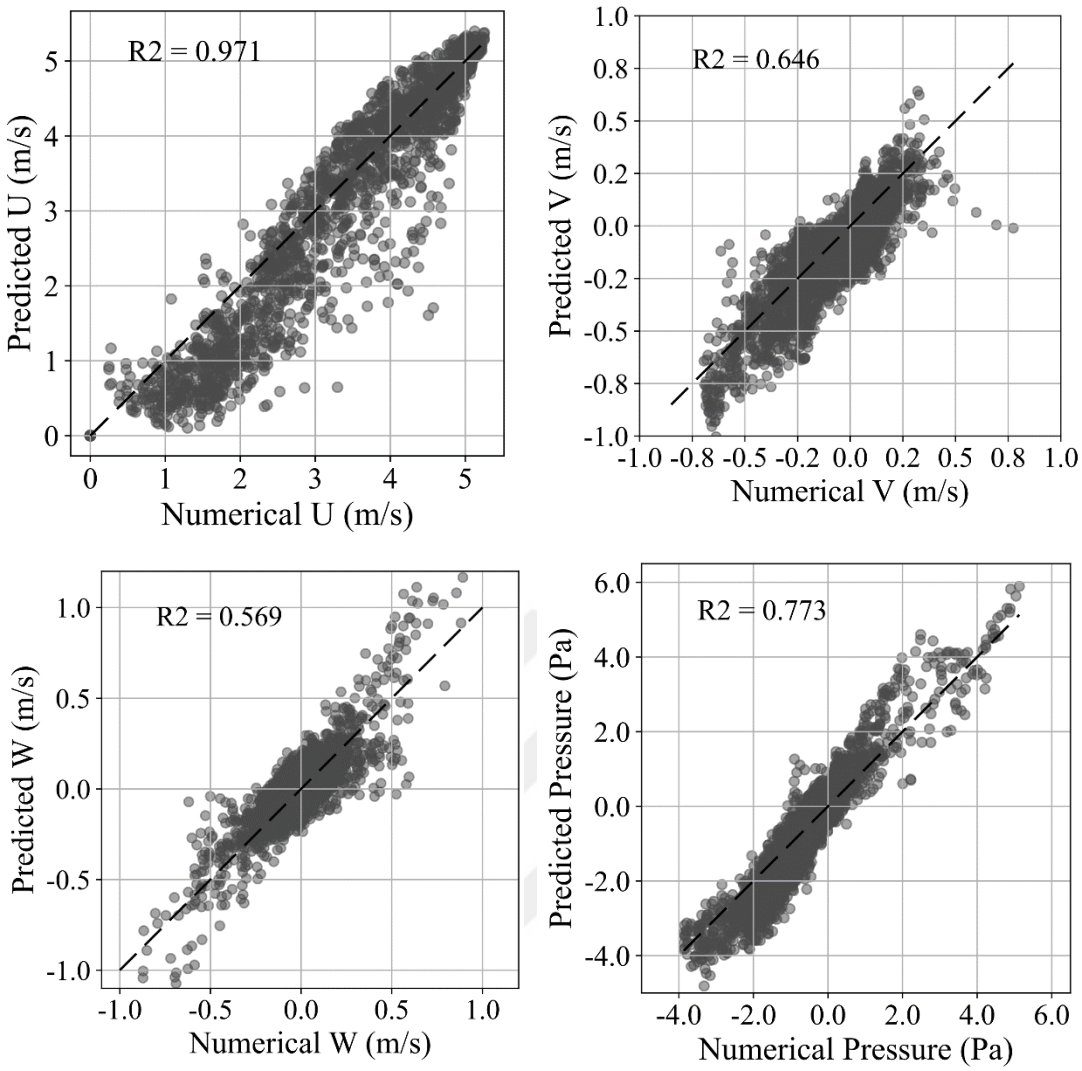


Figure 4.5 : Comparison of numerical simulation and PointNet prediction and R^2 for the sample number 902, worst case (small dataset).

Figure 4.6 and Figure 4.7 showcase a similar comparison for the best-predicted and worst-predicted cases within the larger dataset. Notably, the model demonstrates consistent results as observed in the case of the smaller dataset.

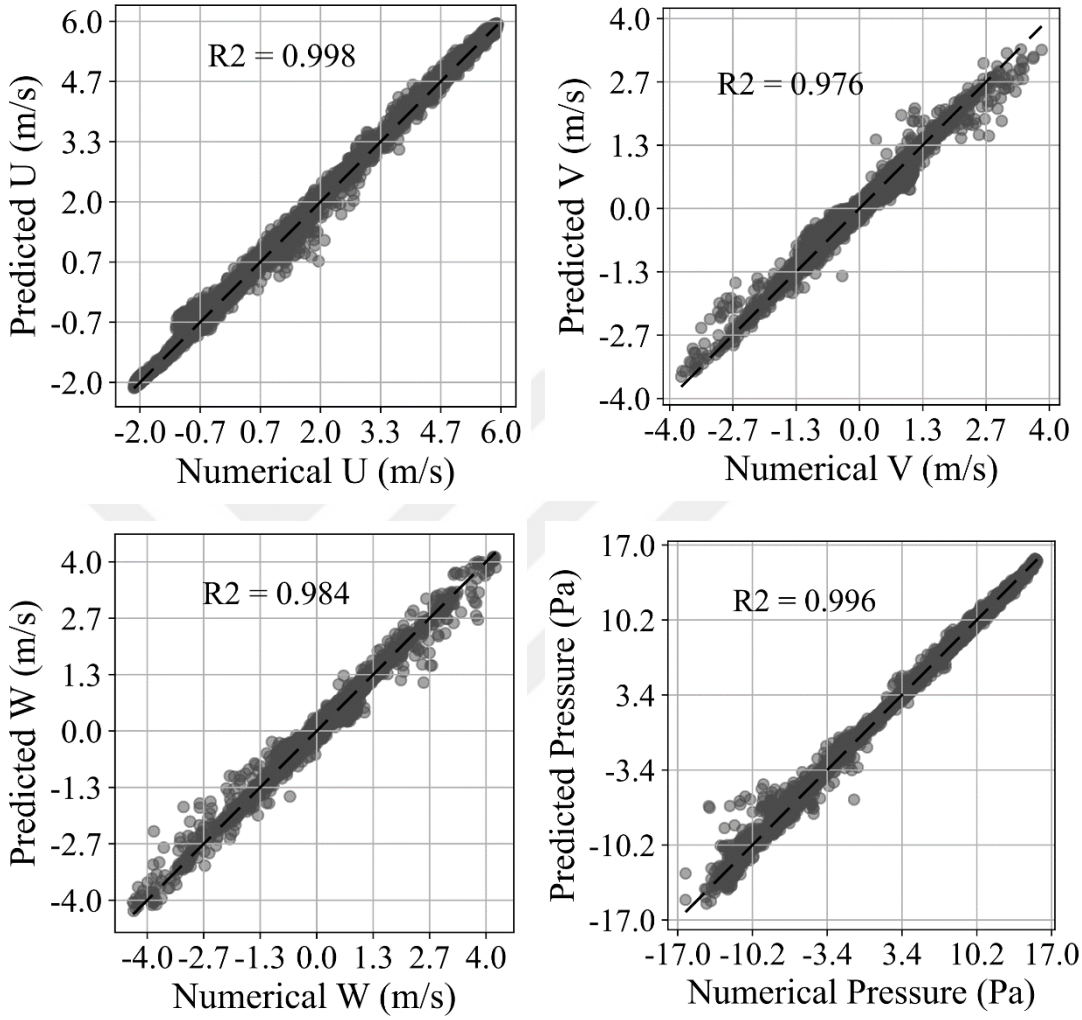


Figure 4.6 : Comparison of numerical simulation and PointNet prediction and R^2 for the sample number 751, best case (large dataset).

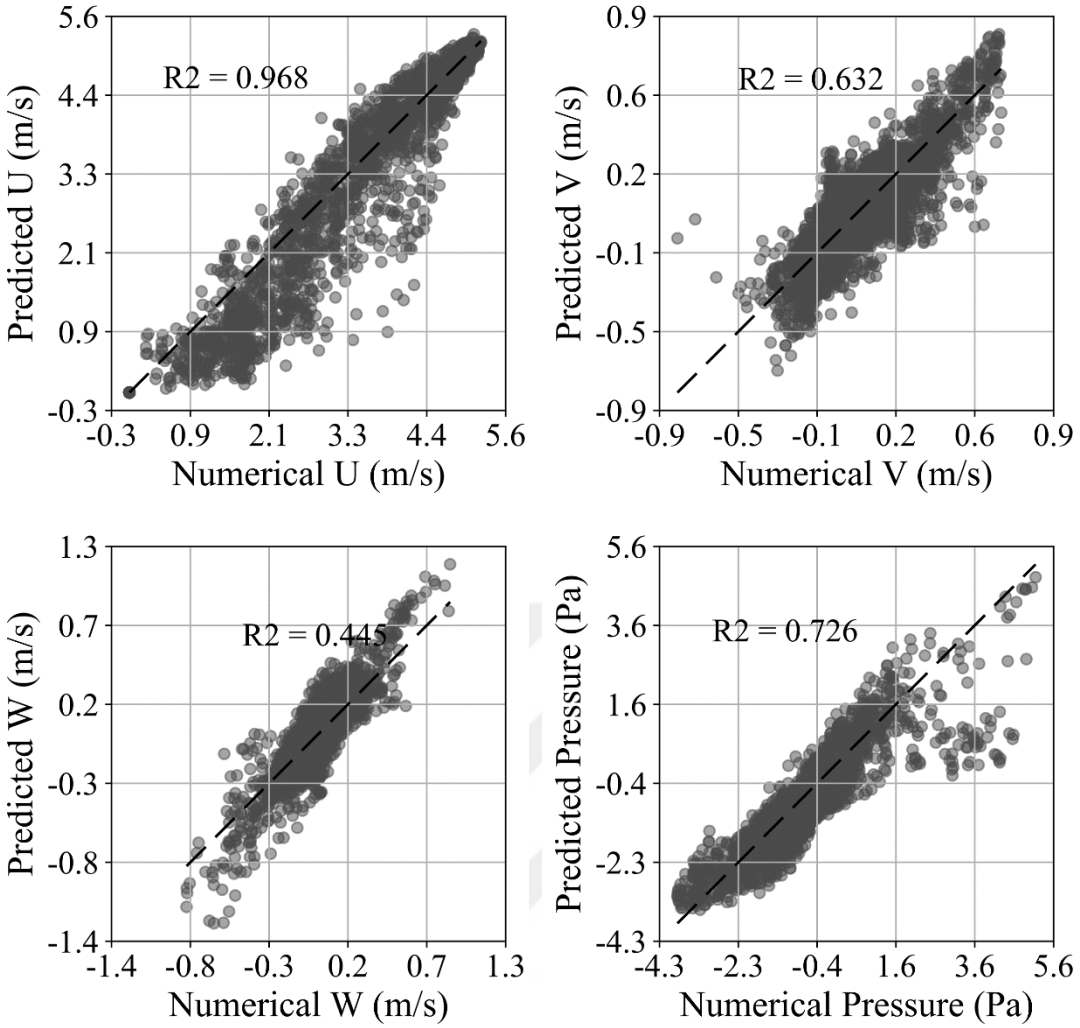


Figure 4.7 : Comparison of numerical simulation and PointNet prediction and R^2 for the sample number 932, worst case (large dataset).

The comparison of U, and V velocities as well as pressure distribution contours obtained from numerical simulation and PointNet prediction results for the smaller dataset are plotted in Figure 4.8, Figure 4.9, and Figure 4.10 respectively. Furthermore, the same comparisons are presented in Figure 4.11 and Figure 4.12, focusing on the best and worst predicted cases within the larger dataset. The network demonstrates a high level of accuracy in predicting flow behavior across crucial regions, including the stagnation and wake regions, for both the best and average cases. The predicted contours reveal consistent patterns, such as the formation of a recirculation region with negative velocities and negative pressure behind the object. Additionally, the recovery of the wake and the presence of a stagnation region in front of the object are observed. The velocities in the Y-direction exhibit asymmetry, showcasing both positive and negative flow deflection, which can be attributed to the angle of attack. However, it is

worth noting that the worst cases remain as anomalies in the results, deviating significantly from the expected trends.

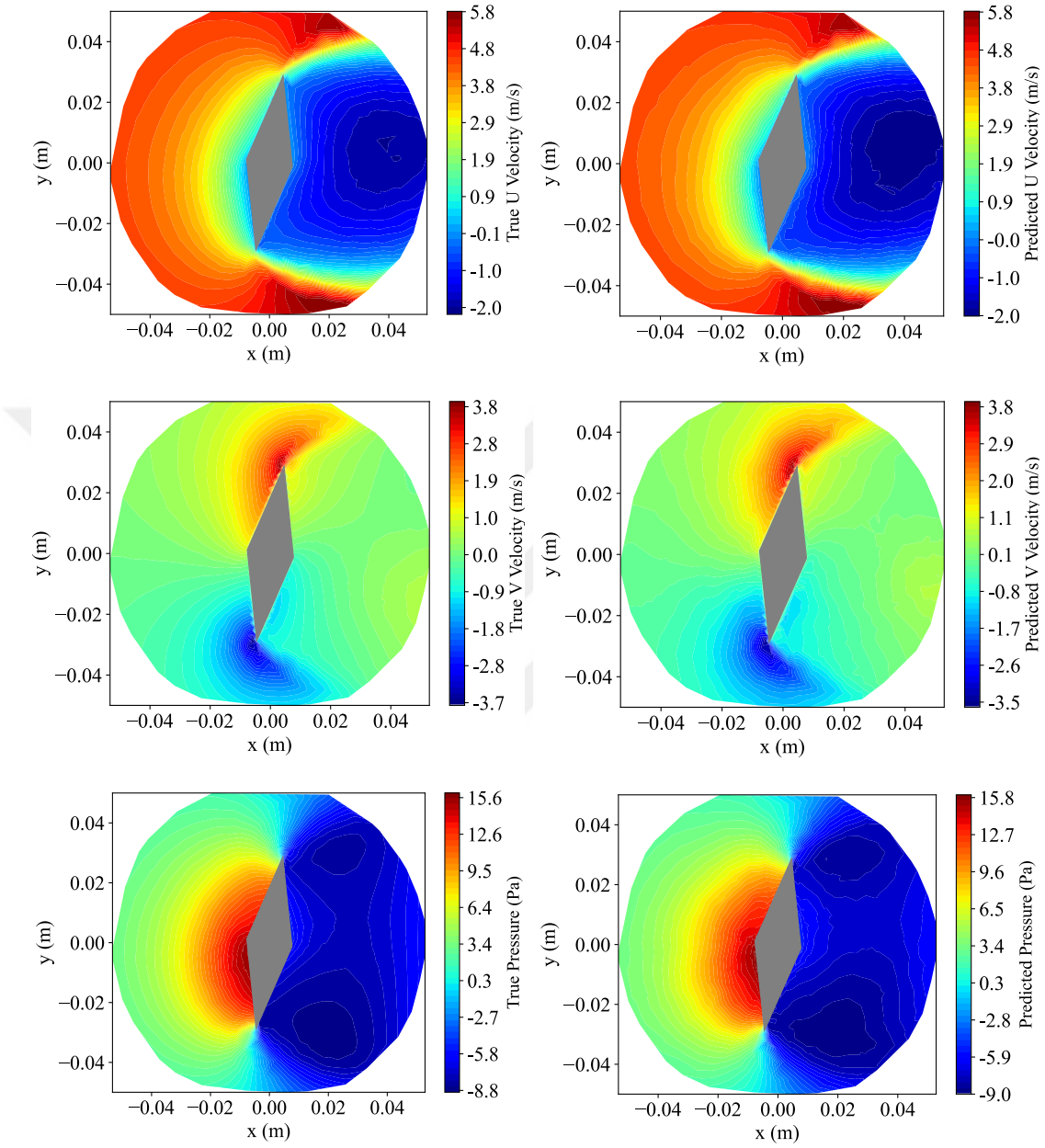


Figure 4.8 : Comparison of numerical and prediction results: U and V velocity and pressure distributions for sample number 718, best case (small dataset).

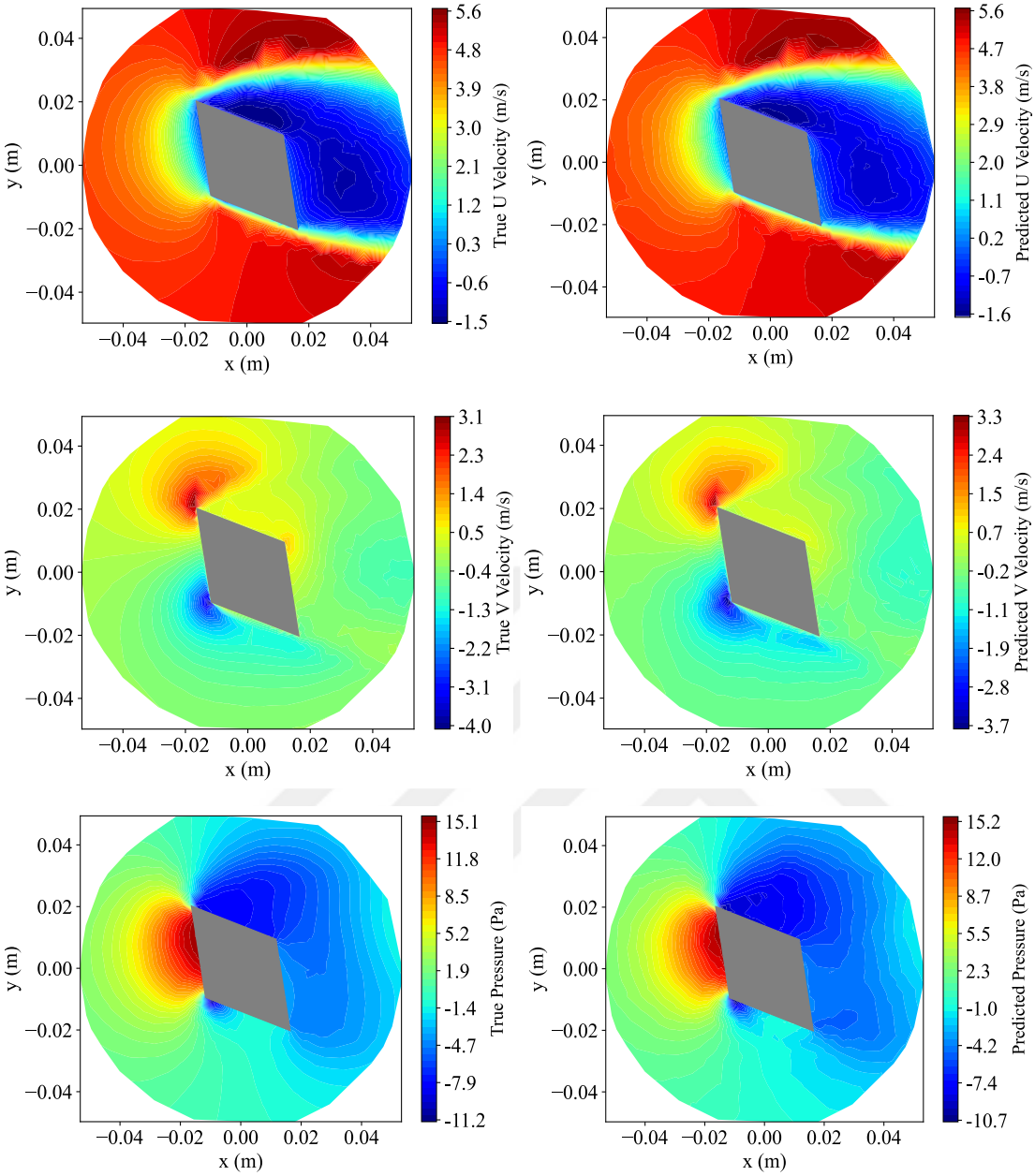


Figure 4.9 : Comparison of numerical and prediction results: U and V velocity and pressure distributions for sample number 318, average case (small dataset).

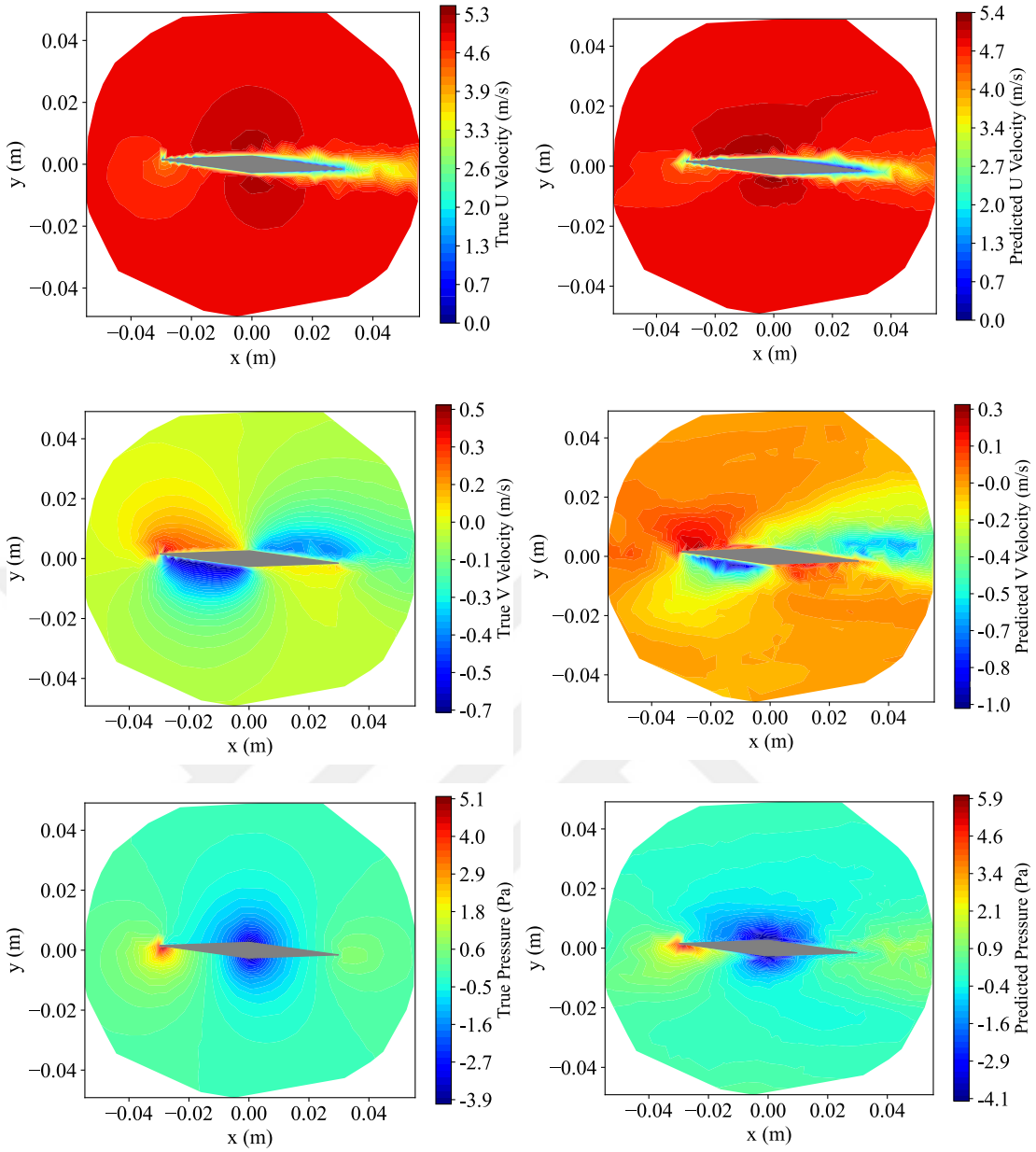


Figure 4.10 : Comparison of numerical and prediction results: U and V velocity and pressure distributions for sample number 902, worst case (small dataset).

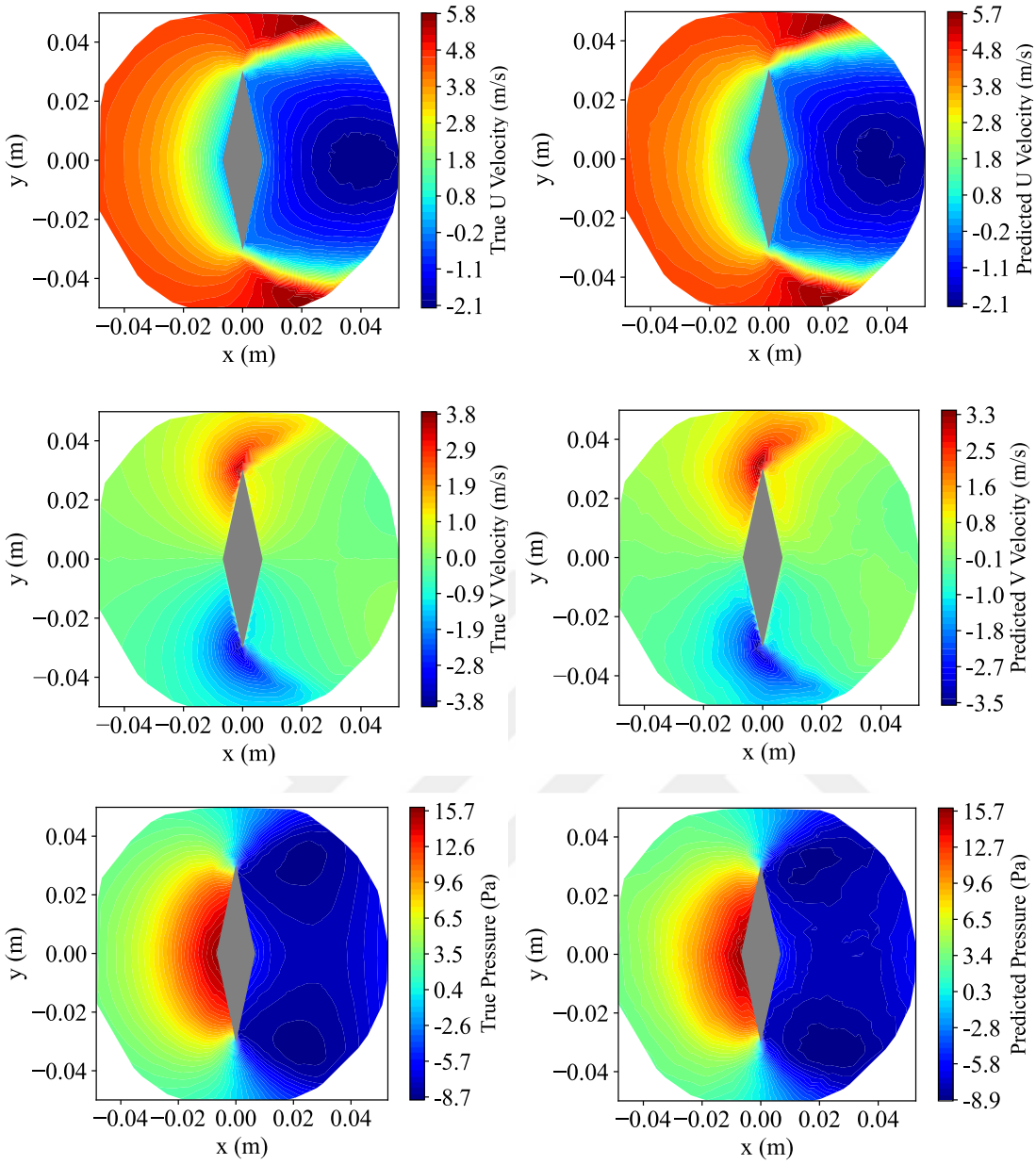


Figure 4.11: Comparison of numerical and prediction results: U and V velocity and pressure distributions for sample number 751, best case (large dataset).

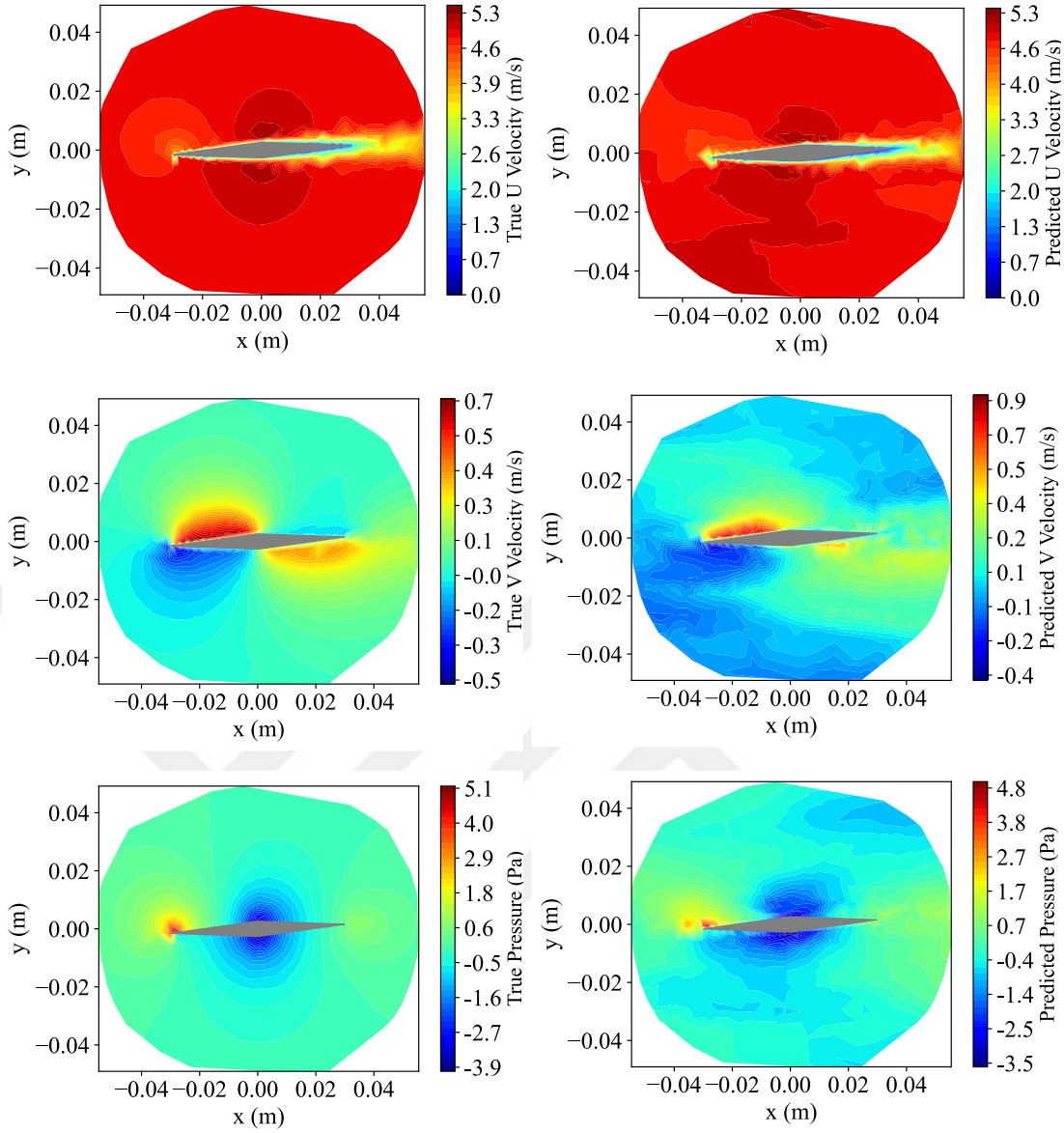


Figure 4.12 : Comparison of numerical and prediction results: U and V velocity and pressure distributions for sample number 932, worst case (large dataset).

Figure 4.13 displays the three-dimensional representation of the predicted flow for sample number 718. In this depiction it can be seen that the model is able to adeptly capture the flow behavior in both the stagnation and wake regions. The recirculation area marked by rotating vectors behind the object is effectively captured. This visual insight highlights the predictive capabilities of the machine learning model, effectively reproducing the intricate interplay between the solid body and the surrounding fluid environment.

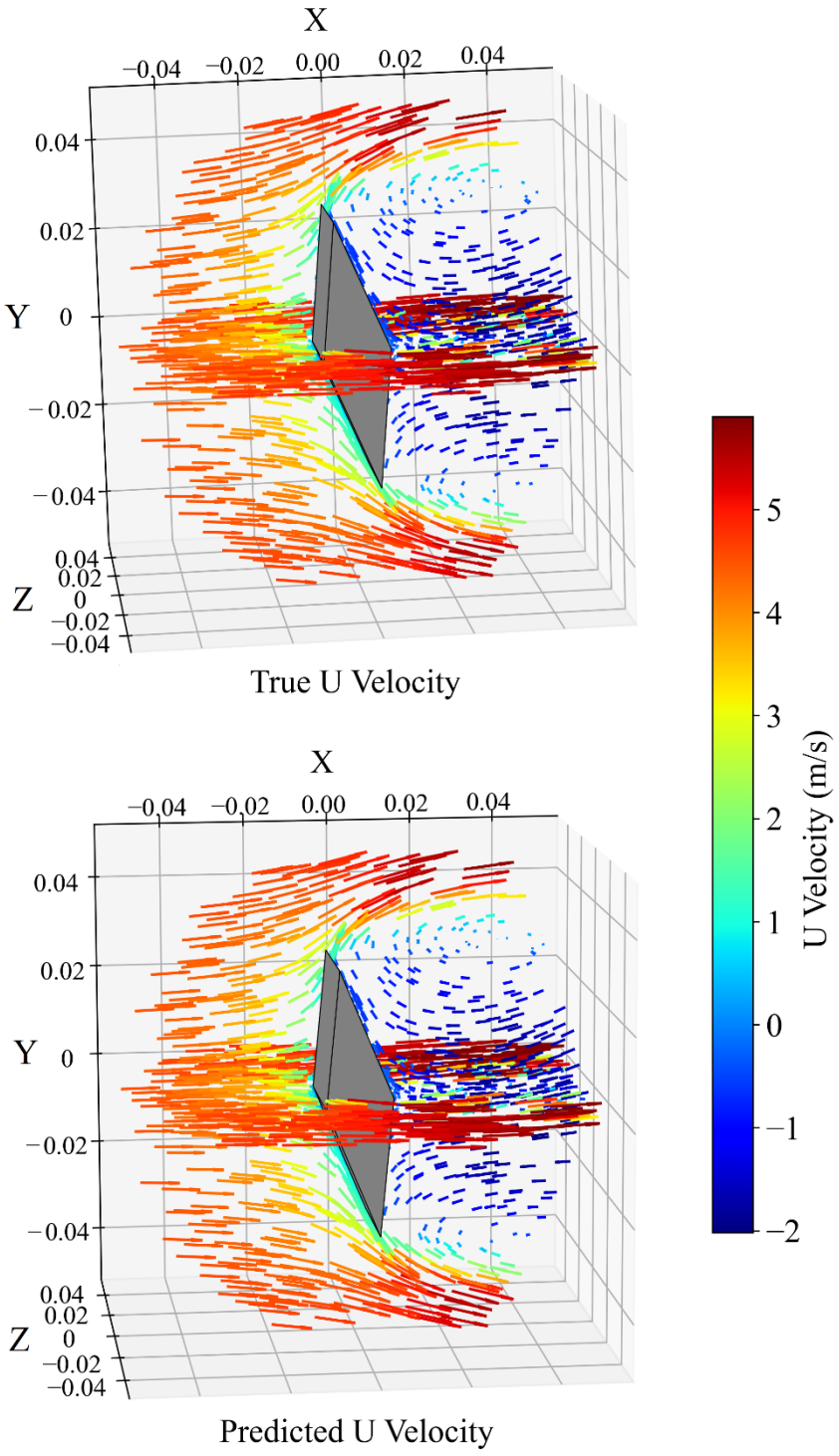


Figure 4.13 : 3D visualization of predicted flow around the sample number 718, two cross-sections at z = 0 and y = 0.

Figure 4.14, Figure 4.15, and Figure 4.16 showcase contour plots of relative error distributions for the best, average, and worst predicted cases within the smaller dataset, respectively. Additionally, Figure 4.17 and Figure 4.18 exhibit analogous plots for the best and worst predicted cases within the larger dataset. It seems that the majority of errors in the flow field prediction are concentrated in the immediate vicinity of the object surface. This occurrence is particularly pronounced in regions where the gradients of the velocity exhibit substantial magnitudes. These observations suggest that the accuracy of the flow field prediction is notably influenced by the complex flow dynamics and rapid velocity changes occurring in the close proximity of the bluff body.

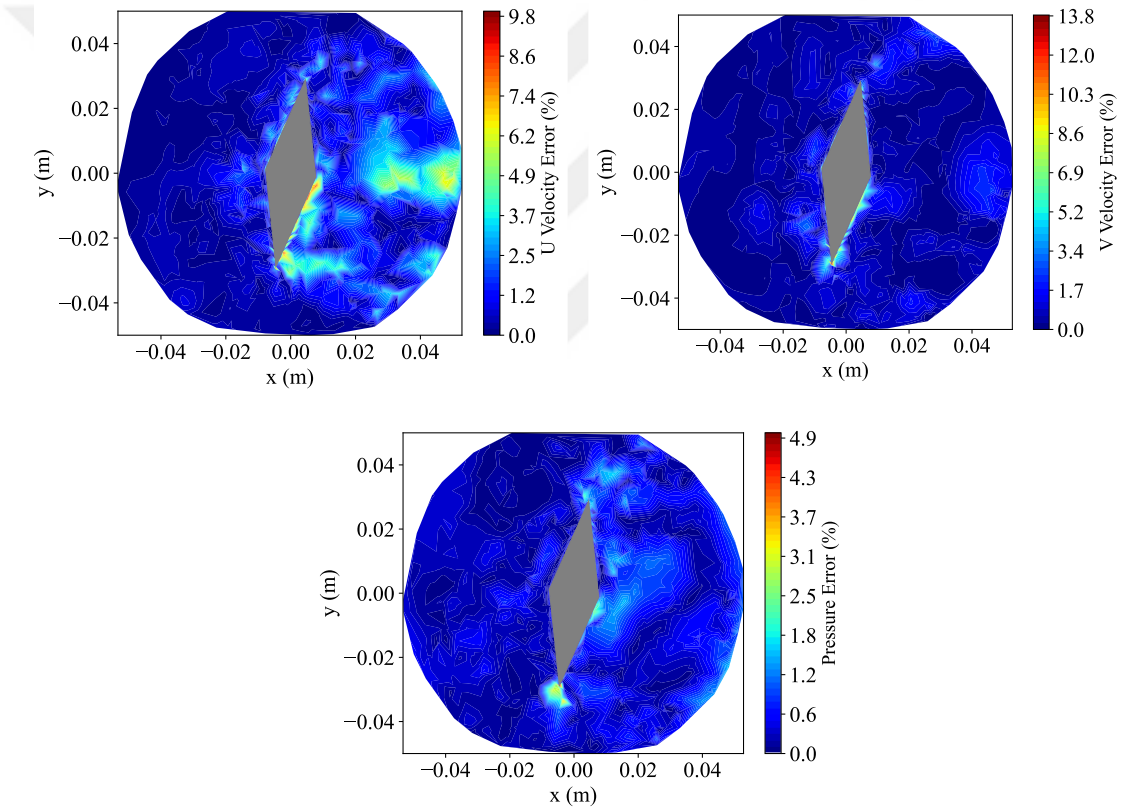


Figure 4.14 : Relative error distribution, sample number 718, best case (small dataset).

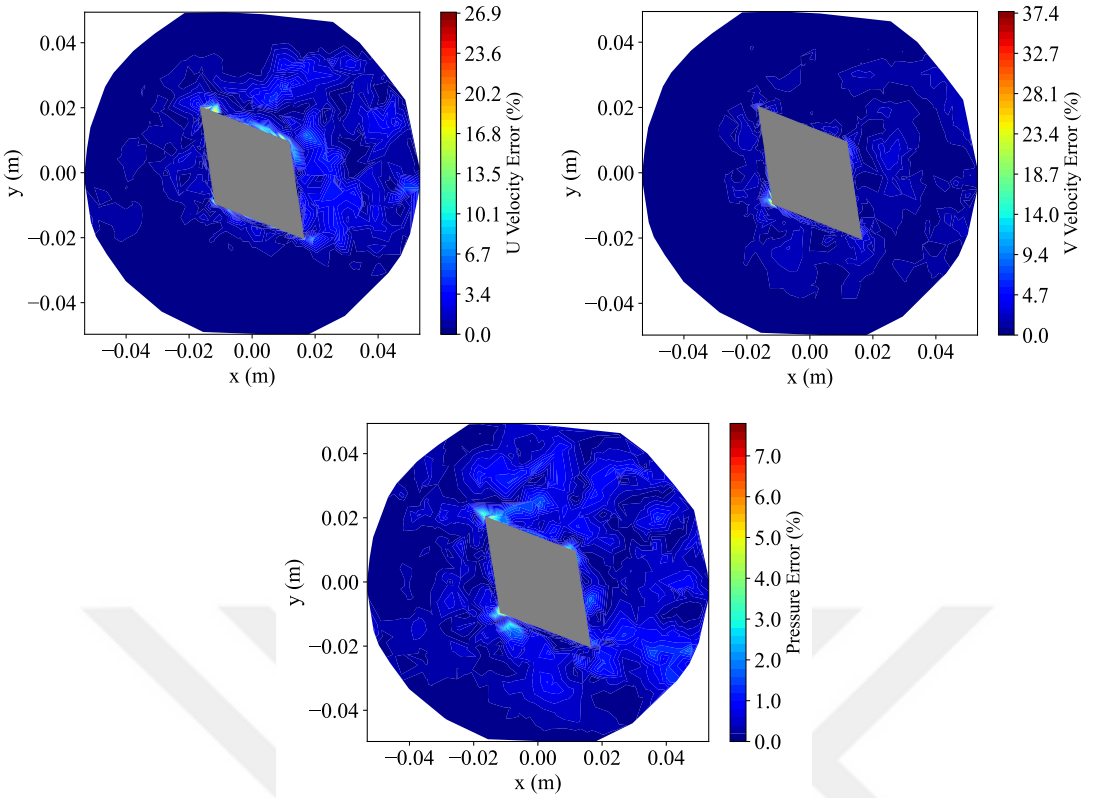


Figure 4.15: Relative error distribution, sample number 318, average case (small dataset).

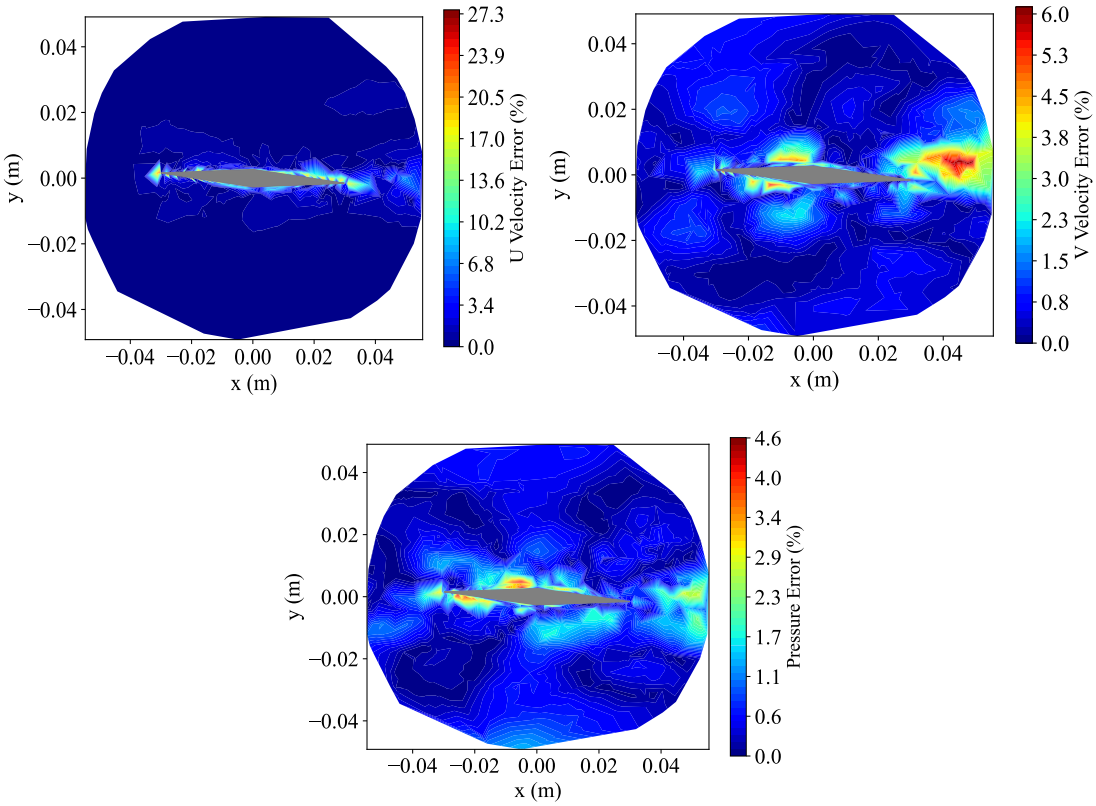


Figure 4.16 : Relative error distribution, sample number 902, worst case (small dataset).

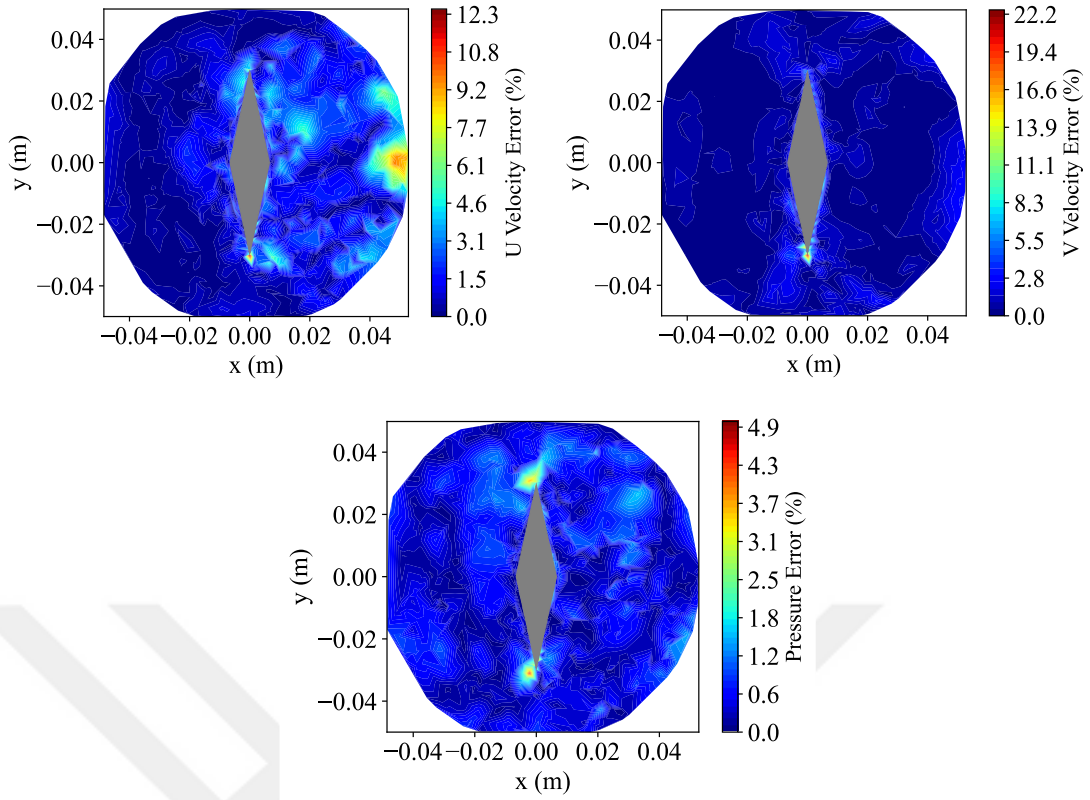


Figure 4.17 : Relative error distribution, sample number 751, best case (large dataset).

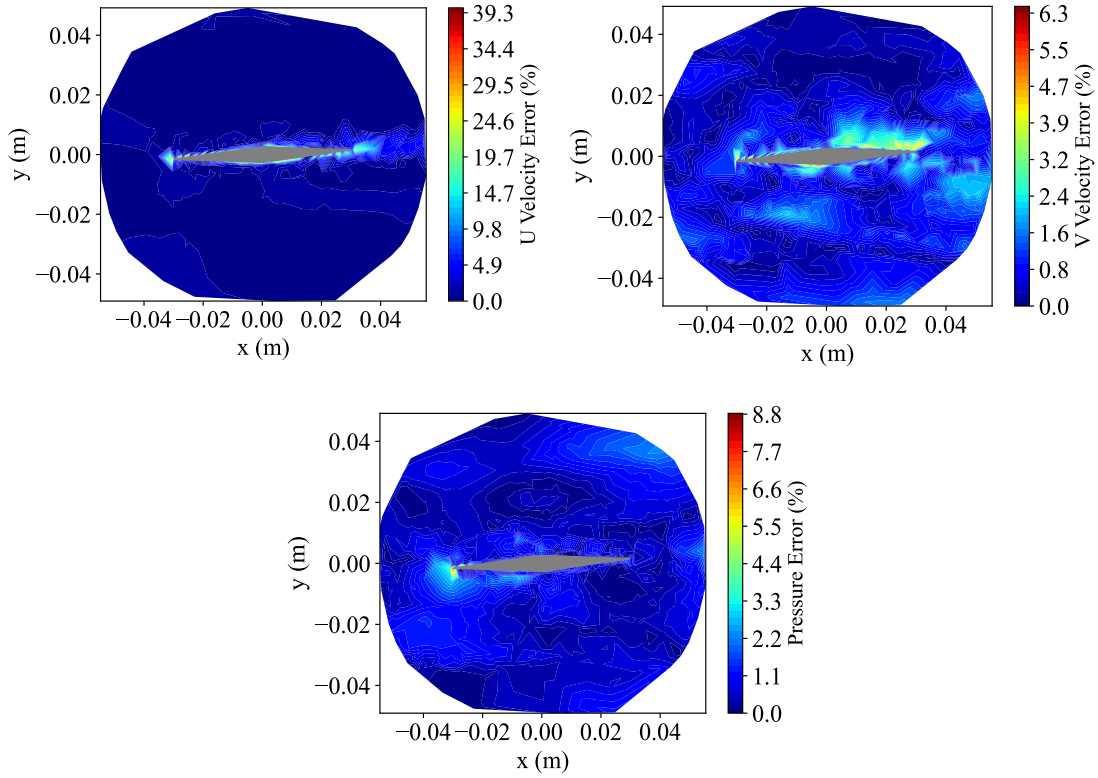


Figure 4.18 : Relative error distribution, sample number 932, worst case (large dataset).

Overall, these results highlight the effectiveness of the model in capturing global flow patterns while also pointing out areas where further improvements may be necessary to enhance accuracy near the object surface.

Table 4.3 displays the computational times required for both the CFD simulation and network prediction. This gap is expected to widen further when dealing with more complex flow fields, particularly when incorporating the energy equation. This is due to the fact that the prediction time remains relatively stable, while the time required for CFD simulations significantly increases for intricate simulations.

Table 4.3 : Comparison of average computation time for one sample.

Simulation type	CPU time (intel @ 3.3 GHz, 12 cores)
CFD simulation time per case	460 sec
PointNet prediction time per case	< 1 sec

5. CONCLUSIONS

Deep learning techniques have gained popularity among CFD practitioners for optimizing design processes. In realistic scenarios, the analysis of 3D models is essential, yet there is a scarcity of studies in this particular domain.

A deep learning framework has been proposed to predict characteristics of 3D flow fields, including velocities and pressure distributions. The architecture of this framework is designed based on the PoinNet architecture, which is capable of processing unordered point data as input. While originally developed for predicting 2D flow fields and having potential for 3D flow prediction, certain modifications were necessary to adapt it to the context of 3D flows. The study focuses on an external turbulent flow passing a 3D bluff body. The solid object generating the dataset is a right rhombic prism with four variable geometric parameters, altering its shape. Two datasets are generated, each considering a different number of geometric parameters. The first dataset comprises 961 diverse samples, while the larger dataset comprises 3511 samples. MATLAB and Fluent are coupled to automate the simulation procedure, yielding numerical solutions. Hyperparameter optimization for the deep learning model is performed using a Bayesian optimization approach.

The model's ability to generalize is assessed using test subsets after the completion of the training phase. Test loss values of 1.38×10^{-4} and 2.59×10^{-4} are obtained for the small and large datasets, respectively. These values are slightly higher than the corresponding validation losses, with no anomalies observed. Additionally, point-wise relative error percentages (REP) and average relative error percentages (AREPs) are used to evaluate the model's performance. The average error across the entire test subsets closely aligns with the minimum value, indicating that most samples in the test subset exhibit relative error values close to the minimum.

Furthermore, an in-depth model evaluation that involves comparing predicted data with numerical simulations is conducted using various metrics. These metrics include velocity and pressure distribution contours, error distribution contours, correlation charts, and determination coefficients for each output parameter. The deep learning

model's prediction results are presented using three samples from the test subset of the small dataset (representing best, average, and worst predictions) and two samples from the test subset of the large dataset (representing best and worst predictions). Considering all the correlation charts, flow field contours, and error distribution contours, the model demonstrates its capability to accurately predict crucial areas of 3D flow fields, such as separation, wake, and recirculation regions. However, the worst-case stands as an outlier in the results.

In conclusion, the study reveals that PoinNet has the potential to predict complex 3D flows with an acceptable to good level of accuracy. Moreover, employing the deep learning approach significantly reduces the computational time required for obtaining 3D flow field data, which is particularly advantageous for tasks involving extensive design space exploration, such as geometry optimization.

REFERENCES

[1] **François Chollet,** "Deep Learning with Python," Manning Publications, p. 4.

[2] "**What are Neural Networks**? | IBM." https://www.ibm.com/topics/neural-networks

[3] **I. Neutelings**, "Neural networks." https://tikz.net/neural_networks

[4] **J. Viquerat and E. Hachem,** "A supervised neural network for drag prediction of arbitrary 2D shapes in low Reynolds number flows." arXiv, Jul. 03,2020. Accessed: May 24, 2023. [Online]. Available: http://arxiv.org/abs/1907.05090

[5] **K. Team, "Keras documentation**: Adam." https://keras.io/api/optimizers/adam/

[6] **D. P. Kingma and J. Ba,** "Adam: A Method for Stochastic Optimization." arXiv, Jan. 29, 2017. Accessed: Mar. 24, 2023. [Online]. Available: http://arxiv.org/abs/1412.6980

[7] "**Explain Pooling layers**," *For Machine Learning*. https://androidkt.com/explain-pooling-layers-max-pooling-average-pooling-global-average-pooling-and-global-max-pooling/

[8] **Keras Team**, "Keras documentation: Layer weight regularizers." https://keras.io/api/layers/regularizers/

[9] **X. Guo, W. Li, and F. Iorio**, "Convolutional Neural Networks for Steady Flow Approximation," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, in KDD '16. New York, NY, USA: Association for Computing Machinery, Aug. 2016, pp. 481–490.

[10] **E. Yilmaz and B. German**, "A Convolutional Neural Network Approach to Training Predictors for Airfoil Performance," in *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Denver, Colorado: American Institute of Aeronautics and Astronautics, Jun. 2017.

[11] **A. B. Farimani, J. Gomes**, and V. S. Pande, "Deep Learning the Physics of Transport Phenomena." arXiv, Sep. 07, 2017. Accessed: May 23, 2023. [Online]. Available: http://arxiv.org/abs/1709.02432

[12] **H. Wu, X. Liu, W. An, S. Chen, and H. Lyu**, "A deep learning approach for efficiently and accurately evaluating the flow field of supercritical airfoils," *Computers & Fluids*, vol. 198, p. 104393, Feb. 2020.

[13] **H. R. Tamaddon-Jahromi, N. K. Chakshu, I. Sazonov, L. M. Evans, H. Thomas, and P. Nithiarasu**, "Data-driven inverse modelling through neural network (deep learning) and computational heat transfer," *Computer Methods in Applied Mechanics and Engineering*, vol. 369, p. 113217, Sep. 2020.

[14] **S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, and S. Kaushik**, "Prediction of aerodynamic flow fields using convolutional neural networks," *Comput Mech*, vol. 64, no. 2, pp. 525–545, Aug. 2019.

[15] **V. Sekar, Q. Jiang, C. Shu, and B. C. Khoo**, "Fast flow field prediction over airfoils using deep learning approach," *Physics of Fluids*, vol. 31, no. 5, p. 057103, May 2019.

[16] **J. Chen, J. Viquerat, and E. Hachem**, "U-net architectures for fast prediction of incompressible laminar flows." arXiv, Oct. 25, 2019. [Online]. Available: http://arxiv.org/abs/1910.13532

[17] **Y. Zhang, W.-J. Sung, and D. Mavris**, "Application of Convolutional Neural Network to Predict Airfoil Lift Coefficient." arXiv, Jan. 16, 2018.

[18] **V. Sekar, M. Zhang, C. Shu, and B. C. Khoo**, "Inverse Design of Airfoil Using a Deep Convolutional Neural Network," *AIAA Journal*, vol. 57, no. 3, pp. 993–1003, Mar. 2019.

[19] **G. Hajgató, B. Gyires-Tóth, and G. Paál**, "Predicting the flow field in a U-bend with deep neural networks." arXiv, Oct. 01, 2020. Accessed: May 23, 2023. [Online]. Available: http://arxiv.org/abs/2010.00258

[20] **K. Fukami, K. Fukagata, and K. Taira**, "Super-resolution reconstruction of turbulent flows with machine learning," *Journal of Fluid Mechanics*, vol. 870, pp. 106–120, Jul. 2019.

[21] **X. Hui, J. Bai, H. Wang, and Y. Zhang**, "Fast pressure distribution prediction of airfoils using deep learning," *Aerospace Science and Technology*, vol. 105, p. 105949, Oct. 2020.

[22] **N. Thuerey, K. Weissenow, L. Prantl, and X. Hu**, "Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows," *AIAA Journal*, vol. 58, no. 1, pp. 25–36, Jan. 2020.

[23] **M. Eichinger, A. Heinlein, and A. Klawonn**, "Surrogate Convolutional Neural Network Models for Steady Computational Fluid Dynamics Simulations," *Universität zu Köln*, Dec. 14, 2020. http://www.uni-koeln.de/ (accessed May 23, 2023).

[24] **S. Shahane, P. Kumar, and S. P. Vanka**, "Sensitivity Analysis of Lift and Drag Coefficients for Flow over Elliptical Cylinders of Arbitrary Aspect Ratio and Angle of Attack using Neural Network." arXiv, Oct. 28, 2021. Accessed: May 23, 2023. [Online]. Available: http://arxiv.org/abs/2012.10768

[25] **J. An, H. Wang, B. Liu, K. H. Luo, F. Qin, and G. Q. He**, "A deep learning framework for hydrogen-fueled turbulent combustion simulation," *International Journal of Hydrogen Energy*, vol. 45, no. 35, pp. 17992–18000, Jul. 2020.

[26] **M. D. Ribeiro, A. Rehman, S. Ahmed, and A. Dengel**, "DeepCFD: Efficient Steady-State Laminar Flow Approximation with Deep Convolutional Neural Networks." arXiv, Nov. 26, 2021. Accessed: May 23, 2023. [Online]. Available: http://arxiv.org/abs/2004.08826

[27] **L.-W. Chen, B. A. Cakal, X. Hu, and N. Thuerey**, "Numerical investigation of minimum drag profiles in laminar flow using deep learning surrogates," *J. Fluid Mech.*, vol. 919, p. A34, Jul. 2021.

[28] **M. A. Bouhlel, S. He, and J. R. R. A. Martins**, "Scalable gradient–enhanced artificial neural networks for airfoil shape design in the subsonic and transonic regimes," *Struct Multidisc Optim*, vol. 61, no. 4, pp. 1363–1376, Apr. 2020.

[29] **G. Waxenegger-Wilfing, K. Dresia, J. C. Deeken, and M. Oschwald**, "Heat Transfer Prediction for Methane in Regenerative Cooling Channels with Neural Networks," *Journal of Thermophysics and Heat Transfer*, vol. 34, no. 2, pp. 347–357, Apr. 2020.

[30] **A. Kashefi, D. Rempe, and L. J. Guibas,** "A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries," *Physics of Fluids*, vol. 33, no. 2, p. 027104, Feb. 2021.

[31] **P. Pant, R. Doshi, P. Bahl, and A. B. Farimani**, "Deep Learning for Reduced Order Modelling and Efficient Temporal Evolution of Fluid Simulations," *Physics of Fluids*, vol. 33, no. 10, p. 107101, Oct. 2021.

[32] **X. Du, P. He, and J. R. R. A. Martins,** "Rapid airfoil design optimization via neural networks-based parameterization and surrogate modeling," *Aerospace Science and Technology*, vol. 113, p. 106701, Jun. 2021.

[33] **Y. Yang and Y. Mesri**, "Learning by neural networks under physical constraints for simulation in fluid mechanics," *Computers & Fluids*, vol. 248, p. 105632, Nov. 2022, doi: 10.1016/j.compfluid.2022.105632.

[34] **J. Liu, R. Chen, J. Lou, Y. Hu, and Y. You**, "Deep-learning-based aerodynamic shape optimization of rotor airfoils to suppress dynamic stall," *Aerospace Science and Technology*, vol. 133, p. 108089, Feb. 2023.

[35] **K. Zuo, S. Bu, W. Zhang, J. Hu, Z. Ye, and X. Yuan**, "Fast sparse flow field prediction around airfoils via multi-head perceptron based deep learning architecture," *Aerospace Science and Technology*, vol. 130, p. 107942, Nov. 2022.

[36] ***ANSYS Fluent Theory Guide**, Release 2021 R1, Section 4.3.1*.

[37] **S. F. Hoerner**, *Fluid-dynamic drag: practical information on aerodynamic drag and hydrodynamic resistance*, [2d. ed. Midland Park, N. J.], 1958.

[38] **G. Aksu, C. O. Güzeller, and M. T. Eser**, "The Effect of the Normalization Method Used in Different Sample Sizes on the Success of Artificial Neural Network Model," *Int. J. Assess. Tools Educ.*, vol. 6, no. 2, Art. no. 2, Jul. 2019.

[39] **C. R. Qi, H. Su, K. Mo, and L. J. Guibas**, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." arXiv, Apr. 10, 2017.

CURRICULUM VITAE

Name Surname: Farhad NEMATI TAHER

EDUCATION:

- **B.Sc. :** 2012, Tabriz Branch of Azad University, Faculty of Mechanical Engineering**.**
- **M.Sc. :** 2023, Istanbul Technical University, Faculty of Mechanical Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2015-2019 Tabriz, IRAN, Mechanical engineer and Manager of Quality Test Laboratory. Imen Part Rafee. Manufacturing of Ball Joints and Spare Parts of Automobiles.
- 2012-2015 Tabriz, IRAN, Mechanical Engineer Azer Petro Farayand. Design and Manufactureing of Industrial Machines and Related Components.
- 2010-2011 Tabriz, IRAN, Project Designer Shimi Pajouhesh Sanat. Craft Paper Equipment.

PUBLICATIONS AND PRESENTATIONS:

- ***F. Nemati Taher***, S. Zeyninejad Movassag, K. Razmi, R. Tasouji Azar, ‘‘***Baffle Space Impact on the Performance of Helical Baffle Shell and Tube Heat Exchangers***’’. Applied Thermal Engineering 44 (2012) 143-149.
- S. Zeyninejad, ***F. Nemati Taher***, R. Tasouji Azar, and K. Razmi, “***Tube bundle replacement for segmental and helical shell and tube heat exchangers: performance comparison and fouling investigation on the shell side***”. Applied Thermal Engineering, 51 (2013) 1162-1169.
- R. TasoujiAzar, S. Zeyninejad, and ***F. Nemati***, “***Comparison of Heat Transfer Coefficient per Pressure Drop Ratio in Shell & Tube Heat Exchanger with Segmental and Helical Baffle***”, 7th International Conference on Heat Transfer, Fluid Mechanics, and Thermodynamics HEFAT2010, 19-21 July 2010, Antalya, Turkey.

- R. Tasouji Azar, … ,**7- *F. Nemati***. ‘‘***Design and manufacturing of first helical shell and tube heat exchanger in Tabriz Petroleum Company***’’. First international conference in the field of energy and heat exchanger, 2009. Tehran, Iran.