Double Cone Flow Field Reconstruction Between Mach 4 and 12 Using Machine Learning Techniques

THESIS

Trevor A. Toros, Second Lieutenant, USAF

AFIT-ENY-MS-22-M-313

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENY-MS-22-M-313

DOUBLE CONE FLOW FIELD RECONSTRUCTION BETWEEN MACH 4 AND 12 USING MACHINE LEARNING TECHNIQUES

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science

Trevor A. Toros, B.S.

Second Lieutenant, USAF

March 2022

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENY-MS-22-M-313

DOUBLE CONE FLOW FIELD RECONSTRUCTION BETWEEN MACH 4 AND 12 USING MACHINE LEARNING TECHNIQUES

Trevor A. Toros, B.S.
Second Lieutenant, USAF

Committee Membership:

Ramana V. Grandhi, PhD
Chairman

José A. Camberos, PhD, P.E.
Member

Mark F. Reeder, PhD
Member

ADEDJI B. BADIRU, PhD
Dean, Graduate School of Engineering and Management

Abstract

Analysis of hypersonic vehicle designs are costly due, in part, to the physical phenomena unique to the hypersonic flight regime. It is desirable to consider as many of these phenomena as possible early in vehicle design when computational resources are limited. Reduced order models can provide insight into these phenomena at a low cost by leveraging previous results. The utility of machine learning models for predicting the pressure field around a hypersonic weapon in flight is investigated. A parameterized double cone model is simulated at hypersonic speeds using a steady-state RANS solver, Kestrel. The resulting pressure fields are used to train two neural network (NN) models, the U-Net and the Multiscale Network, as well as two meta models, K-Nearest Neighbors and Regression Kriging. The NN models are designed to extract flow field relationships using distinct methodologies: the U-Net utilizes auto-encoding while the Multiscale Network utilizes a sequential refinement scheme. All models predict the pressure values on a uniform Cartesian grid of much smaller resolution than the unstructured mesh required for CFD simulation. The accuracy, computational complexity, and versatility of the NN are compared against the meta models. Additionally, the ability for each method to accurately predict shock interactions or impingement with downstream vehicle geometry is examined. Such closed-form ML models can provide advantages over traditional CFD solutions as they do not require any meshing of the computational domain and can quickly generate flow field predictions - on the order of seconds. The NN models were found have lacking yet robust performance on this dataset. Additionally, the NN models were shown to be effortlessly applicable to arbitrary geometries that cannot be described using the existing geometric parameterization.

Acknowledgements

First, I would like to thank my advisor Dr. Ramana Grandhi for making sure all of the resources I could possibly need were easily available and playing a major supportive role in my education and this thesis. Additionally, I would like to extend my deep gratitude to Dr. Benjamin Grier and Dr. Christopher (Corey) Fischer for their extensive contributions of time, experience, and advice in the pursuit of this research. The advice and direction from Dr. Logan Riley (AFRL) and Dr. Ryan Bond and Mr. Robert Nichols (CREATE-AV) on matters relating to computational tools and preparing and running simulations played a major enabling role in this research as well.

Trevor A. Toros

Table of Contents

Page

Abstract iv

Acknowledgements v

List of Figures viii

List of Tables x

List of Abbreviations xi

I. Introduction 1

1.1 Hypersonic Flight 1
1.2 Hypersonic Environment 3
1.3 Vehicle Design 5
1.4 Machine Learning 8
1.5 Document Organization 9

II. Machine Learning 10

2.1 K-Nearest Neighbors 11
2.2 Gaussian Process Regression 12
2.2.1 Previous Work 14
2.2.2 Regression Kriging 14
2.3 Neural Networks 16
2.4 Neural Network Training 22
2.5 Convolutional Neural Networks 25
2.5.1 Fully Convolutional Networks 35
2.6 U-Net Architecture 36
2.6.1 Previous Work 38
2.7 Multi-Scale Network Architecture 39
2.7.1 Loss Function 41
2.7.2 Masked Refinement 42

III. Model Implementation 46

3.1 K-Nearest Neighbors 46
3.2 Gaussian Process Regression 47
3.3 U-Net Architecture 48
3.4 Multi-Scale Network Architecture 49

Page

IV. Simulation and Data Collection 52

4.1 Double Cone 52
4.2 Design Space Sampling 52
4.2.1 Design Matrix Generation 53
4.2.1.1 Undesirable Latent Distributions 55
4.2.1.2 Design Space Bounds 57
4.2.2 Training, Validation, and Test Splits 61
4.3 Solver Configuration 62
4.4 Verification Case 65
4.5 Computational Mesh 67
4.5.1 Domain Definition 68
4.5.2 Mesh Type Selection 68
4.5.3 Boundary Condition Considerations 73
4.5.4 Procedural Generation 80
4.6 Grid Independence Study 83
4.7 Data Processing 84
4.7.1 Image Generation 85
4.7.2 Data Preparation 88
4.7.3 Insertion and Extraction of Surface Values 90
4.7.4 Interpolation Error Quantification 93

V. Results 102

5.1 Model Tuning 102
5.1.1 k-Nearest Neighbors 104
5.1.2 Gaussian Process Regression 106
5.1.3 U-Network Architecture 107
5.1.4 Multiscale Network Architecture 108
5.1.5 Model Convergence 111
5.2 Surface Distributions 113
5.2.1 Model Performance 118
5.2.2 Categorical Metrics 127
5.3 Prediction Samples 127
5.4 Model Comparison 135
5.4.1 Extended Applicability 136

VI. Discussion 139

6.1 Future Work 145

List of Figures

Figure		Page
1	Overview of data operations in the perceptron	17
3	Neural network with one hidden layer	21
4	Standard CNN architecture	27
5	Demonstration of kernel operation in a CNN	28
6	Convolution operation numerical example	31
7	CNN FoV demonstration	34
8	Convolution operation with multiple input channels	36
9	Fully convolutional network architecture	37
10	U-Net architecture	38
11	Overview of data operations in MS-Net	41
12	Masked-Refinement motivation	45
13	Multiscale Network architecture	51
14	Double-Cone geometry and schematic	53
15	Design space sampling projections and marginal distributions	56
16	Latent distributions from design space sampling	58
17	Splitting dataset into train/test data	62
19	Independent simulations and experimental results	67
20	Definition of the computational domain	69
21	Representative solution on a structured grid	70
23	Recreated reference results	74
28	Grid convergence study results	84
31	Identification of image pixel types	91

Figure Page

32 Insertion of target wall values 92

33 Extraction of predicted wall values 93

34 Comparison of unstructured and Cartesian meshes 95

39 Sample of error introduced by interpolation to the Cartesian grid 98

47 Neural network model training losses 112

57 Model performances in predicting wall value distributions 126

58 Cases where models exhibited their worst wall prediction performance 131

61 Cases where models exhibited their best wall prediction performance 134

63 Example of NN model extended applicability 137

List of Tables

Table		Page
1	Design variable limits	61
2	Verification case flow field and geometric parameters	66
3	Confusion matrices for predictions for shock impingement classifications for all models on the entire dataset	128
4	Confusion matrices of predictions for shock impingement classification for all models on the test dataset	129
5	Accuracy and precision of the model classifications	129
6	Time required to predict the pressure field for both one query configuration and a batch of 64 queries.	136

List of Abbreviations

Abbreviation		Page
ML	Machine Learning	1
USAF	United States Air Force	1
ML	Machine Learning	11
K-NN	K-Nearest Neighbors	11
GP	Gaussian Process	12
GPR	Gaussian Process Regression	12
OK	Ordinary Kriging	14
RK	Regression Kriging	14
MSE	Mean Squared Error	23
RELU	Rectified Linear Unit	24
CNN	Convolutional Neural Networks	25
FoV	Field of Vision	32
RANS	Reynolds Averaged Navier Stokes	39
MS-Net	Multiscale Neural Network	39
RBF	Radial Basis Function	47
CELU	Continuously Differentiable Exponential Linear Units	49
RANS	Reynolds-Averaged Navier-Stokes	63
SA	Spalart-Allmaras	63
CFL	Courant-Friedrichs-Lewy condition	63
CUBRC	Calspan University of Buffalo Research Center	65

DOUBLE CONE FLOW FIELD RECONSTRUCTION BETWEEN MACH 4 AND 12 USING MACHINE LEARNING TECHNIQUES

I. Introduction

This research implements multiple Machine Learning (ML) techniques to predict the steady-state pressure field generated by a hypersonic double-cone body. Flow simulations of the parameterized hypersonic body are generated for a wide range of freestream conditions and geometry configurations. This chapter will discuss the interest in hypersonic vehicles, characteristics of the hypersonic flow regime, and provide examples of previous work in related fields.

1.1 Hypersonic Flight

The hypersonic flight regime has become a topic of interest for organizations around the world. Systems that can travel at and sustain hypersonic speeds present both offensive and defensive advantages for militaries. The United States Air Force (USAF) considers the development of capable hypersonic weapons to be a "game-changing" advancement and anticipates that the 2030 operational environment will utilize these weapons [1]. The trail which has culminated in the design and use of hypersonic systems can be traced to the need for aircraft to reach faster speeds and higher altitudes starting with the Wright brothers' Wright flyer. The initial flight of the Wright flyer marked mankind's ability for powered travel in the atmosphere and ushered in the era of airpower. The development of aircraft built on the achievements of the Wright brothers and the speeds and operating altitudes of subsequent aircraft began to increase exponentially. By the second World War (about 40 years after the

Wright flyer's maiden flight), the construction methods and components of aircraft had changed dramatically, and these aircraft had top speeds of 400 miles per hour (mph), over an order of magnitude larger than that the of first powered aircraft and could reach altitudes of 30,000 feet. By the decades of the Space Race, the 1960's and 1970's, aircraft were able to reach transonic speeds of 1200 mph and had doubled their operational altitudes to 60,000 feet. During this time, the experimental X-15 hypersonic aircraft was able to reach Mach 7 and an altitude of over 350,000 feet [2]. These advances in aircraft capabilities are especially useful in the context of military airpower. Vehicles that can travel and maneuver faster and attain higher altitudes than adversary vehicles have an advantage in air-to-air combat. These attributes also aided in avoiding attacks from the ground as fast moving targets can be difficult to hit with kinetic weapons and such weapons are completely useless if the aircraft flies above the maximum attainable altitude by the munitions. These high-speed and high-altitude capabilities can also enable Intelligence, Surveillance, and Reconnaissance (ISR) missions. Much research had been performed in the controls [3], [4], propulsion [5], and optimization [6]–[8] of such hypersonic vehicles.

Recent interest in hypersonic vehicles has focused on unmanned weapons and munitions. The development of the Inter-Continental Ballistic Missile (ICBM) enhanced the global reach of military munitions by accelerating payloads high above the atmosphere using a rocket. Once the rocket booster runs out of fuel, the munitions fall back to Earth in a minimally-guided ballistic trajectory. The munitions exchange their gravitational potential energy for kinetic energy and can reach very large, hypersonic velocities during their reentry into the atmosphere. These large velocities during descent make targeting and neutralizing the munitions a difficult task, however the ballistic nature of the ICBM flight path makes early detection and predicting the flight path from early trajectory data feasible. Knowing the position of the munition

with relatively small uncertainty during the slowest portions of its flight path make them vulnerable to interception. Air-breathing or gliding hypersonic weapons can reduce these vulnerabilities. These types of hypersonic weapons are able to remain much closer to the surface of the Earth than their ICBM counterparts while retaining their large flight speeds. This combination allows these weapons to delay detection by an adversary, compressing their response time, while also remaining difficult to intercept with traditional munitions [9]. The continued development of hypersonic weapons will enable the USAF to produce more effective offensive weapons as well as bolster the capabilities to better defend against both similar and legacy weapons deployed by adversaries.

1.2 Hypersonic Environment

The hypersonic flight environment introduces several unique fluid flow properties which make this an adverse operating environment. Anderson provides excellent commentary on this topic in his books and publications and some of the main points are summarized here. Key features of hypersonic flows which delineate it from lower-Mach supersonic flow are: thin shock layers, entropy layers, viscous interactions, and high-temperature, low-density flows. [2]

The shock layer is most easily explained for flow over a 2D wedge with a shock wave attached at the leading edge. From the θ-β-M relation, it can be shown that for a prescribed turning angle, increasing the Mach number will reduce the shock angle such that in the limit of Mach number approaching infinity, the shock angle approaches the deflection angle. The corridor of flow between the wall and the shock wave is the shock layer. As stated, in the limit of large Mach numbers, the thickness of this shock layer shrinks and approaches zero. All of the flow passing through the shock wave must travel downstream in the shock layer. Because this layer shrinks with

increasing Mach number, the density of the flow in the shock layer must increase with increasing Mach number. By the Ideal Gas Law and isentropic relations, this increase in density causes an increase in pressure and temperature behind the shock wave as well (these relations can break down in the hypersonic flow regime as discussed later, but are still useful for illustrative purposes). If viscous effects are also considered, a boundary layer will form along the surface. At sufficient Reynolds and Mach numbers, the boundary layer can become a sizeable portion of the shock layer or the shock can merge with the boundary layer, an interaction which can cause large aerothermal loads [2], [10].

Flow which passes through a shock wave generates entropy which is proportional to the strength of the shock wave [2]. The strength of a shock wave is proportional to the incident angle with the freestream where the strongest shock wave is a normal shock which has a 90 degree incident angle with the flow and the strength falls off as the incidence angle decreases. If the shock wave around a hypersonic body is not attached at the leading edge and instead has some positive standoff distance, it will take the form of a bow shock. The scale and standoff distance of a bow shock are influenced by the shape of the hypersonic body and the freestream conditions. The portion of the bow shock near the stagnation point on the body will have an incident angle which is nearly perpendicular to the freestream with the incident angle falling off as the bow shock moves away from the stagnation point. This change in angle changes the strength of local patches of the bow shock which in turn generate a non-uniform distribution of entropy in the flow which passes through the bow shock. The resulting entropy gradient generates vorticity, following Crocco's theorem, and the strongest vorticity is generated in the flow which passed through the bow shock near the stagnation point. This high vorticity flow is concentrated near the boundary layer on the body and can introduce additional energy into the boundary layer. Note that

bow shocks can occur in any supersonic flow, however the strength of bow shocks in low-Mach flows are not sufficient to produce significant amounts of vorticity. [2]

Viscous interactions in the fluid facilitate the transfer of hypersonic flows' kinetic energy into thermal energy. This becomes especially prominent in the boundary layer as the flow must come to a complete stop (with respect to the vehicle) in order to satisfy the no-slip condition at the wall. These extreme temperatures can cause the wall to heat up to temperatures at which the constituent materials can soften and fail (if a metal) if proper thermal management has not been implemented [2]. If a shock wave impinges or interacts with the boundary layer, the resulting aerothermal loads can become even larger [10]. Large, localized aerothermal loads are undesirable, especially in locations where they are not expected or where the local surface has not been designed to handle these loads. Shock impingements in such locations can cause failure of the surface, compromising the aerodynamic and structural integrity of vehicle components. One example of this occurred during a test flight of the X-15 aircraft where the lower vertical stabilizer sustained major damage due to the heating caused by impinging shocks [11]. This incident did not result in the loss of the vehicle or pilot, however compromised external structure hypersonic flows can cause intrusion of high-temperature gases which can expose internal components to the conditions for which they were not designed. This intrusion of gases into the internal structure of the vehicle was determined to be the cause of the destruction of the Columbia space shuttle which resulted in the loss of both the vehicle and the crew on February 1, 2003 [12].

1.3 Vehicle Design

In the present and recent past, there have been relatively few full-scale hypersonic vehicles tested. These tests have been expensive and have had moderate success rates.

Additionally, hypersonic test facilities which can accurately simulate the freestream conditions of hypersonic vehicles for an appreciable percentage of a mission profile do not yet exist. These factors relegate a majority of hypersonic vehicle design to be completed using computational tools. These hypersonic fluid simulations can be complex and extremely computationally expensive requiring on the order of days or weeks to compute. In addition to the large time frames required for computation, there are other preparatory measures such as generating a computational mesh which add additional complexity and time required to produce simulation results for a single configuration. Such long time frames for a single configuration make these types of simulations non-ideal for exploring possible design spaces. Therefore, there is strong incentive to utilize reduced order models or methods which can produce lower-fidelity results at much faster rates in order to explore the design space. Low-fidelity results are those which may not be exactly representative of the true results or may not incorporate information about the physics relevant to the problem. Low-fidelity solutions are generally an approximation of the true solutions where either resolution or physical accuracy is sacrificed for computational expediency.

The process of designing a novel vehicle begins with identifying the space of possible configurations and corresponding constraints on vehicle geometry or conditions which occur in flight. A search is then conducted inside of this space to find the most desirable configuration as defined by one or multiple objective functions. The performance of previous designs can be used to focus or direct the next area of interest in the search. In general, many configurations must be evaluated before finding an optimal or acceptable configuration and the search is stopped. In the case of hypersonic vehicles, the evaluation of a design will often require extensive, expensive CFD simulation or wind tunnel testing to confirm its performance. To mitigate the number of expensive tests required to find an acceptable configuration solution, the

search may be broken into rounds where a computationally inexpensive test which lacks in physical accuracy can be performed on a large group of candidate configurations to identify promising candidates [13]. A certain number of these candidates can then be selected for a more expensive and more physically accurate testing. This process of applying progressively more expensive, more accurate evaluation techniques to a progressively smaller pool of candidates can greatly reduce the total time and computational resources required to find a solution in the design space [14].

Previous works [15]–[19] have developed methods to bring enhanced physical accuracy to cheap evaluation methods to be used early in the design process. Introducing enhanced physical accuracy earlier into the design process can help identify possible undesirable performance characteristics earlier in the search process, focusing the selected designs on more promising configurations earlier. One example of a computationally cheap evaluation method is the Modified Newton Sine Squared method which is a panel-based method to predict the pressure distribution on a hypersonic surface. While this method can produce relatively accurate results in the limit of infinite Mach number, it assigns zero pressure coefficient to panels which are not directly exposed to the freestream. Importantly, this method is not able to make predictions which require knowledge of the evolution of the flowfield such as phenomena which result from shock waves and boundary layers. Using this method, any shock impingements on a vehicle configuration would not be detected and this may lead to further resources spent on investigating the configuration before ultimately discarding it from the pool of candidates. Bringing a Reduced Order Method with similar computational expense to this stage of design exploration has the opportunity to identify undesirable configurations earlier in the design process, saving time and computational resources. This research is focused on predicting the flowfield and

resulting surface pressures on a hypersonic system configuration in a regime where shock impingements are possible.

1.4 Machine Learning

Machine Learning (ML) is a field of numerical modeling which seeks to make sense of or discover underlying distributions in a dataset by minimizing or optimizing some desirability metric in an algorithmic or automated manner. Machine Learning models are self organizing in the sense that their structure or internal parameters can be adjusted to best fit training data. There are two types of learning: unsupervised and supervised. Unsupervised learning utilizes only the input data of a dataset and is generally used in clustering or dimensionality reduction problems. Unsupervised learning is not utilized in this research. Supervised learning utilizes both input and target responses. Models are trained to produce predicted responses which minimize some loss function with respect to the the known, target responses. Such models include simple algorithms such as Ordinary Least Squares (OLS) regression as well as most types of neural networks. Models such as OLS regression have unique solutions for the internal parameters which produce the best performance on a given dataset for a given loss metric. For other models such as neural networks, there are no analytical solutions and the model performance must be improved by iteratively updating the internal parameters. In the case of NN models, there are generally a large number of parameters which can be tuned and updating all of these using guess-and-check methods would be intractable. If the model is specified appropriately, then the direction and magnitude of the updates to the internal parameters can be computed using gradient descent methods.

A variety of machine learning models have been used in the field of hypersonics with applications from surface loading predictions to aeroelastic response modeling to

flight control systems. Kriging models have been used to both predict surface pressure and heat flux values on hypersonic vehicles as well as for hypersonic vehicle control [20]–[23]. Neural network models have been used for many applications including structural displacement predictions [19], mode prediction for aeroelastic effects, and convolutional neural networks have been used in accelerating CFD convergence as well. The U-Net architecture was used to model subsonic flows over airfoils [24] and the Multiscale network was used to model incompressible flow through porous mediums [25].

1.5 Document Organization

This document is organized into six chapters. Chapter 2 builds the background of the machine learning models used in this work. The papers which developed the two neural network models are discussed. Chapter 3 outlines the implementations of these models for this work. The generation of the input data from simulation outputs is also discussed. Chapter 4 discusses the sampling methodology used to generate the design points in the dataset. The utility of the double cone is discussed in the context of hypersonic testing and for this research. The simulations require both settings in the solver to be determined as well as a mesh to perform computations on. The process of generating the mesh and conducting the grid convergence study are also discussed in this chapter. Chapter 5 reviews the results of the model training in the context of their usefulness for preliminary hypersonic design. Finally, Chapter 6 provides some discussion on these results as well as future, follow-on work.

II. Machine Learning

Machine learning is the use of structured algorithms to generate a model of a data set. These models are useful for situations in which the underlying functional relationship in the dataset is unknown. Machine learning algorithms fall into one of two main categories - unsupervised and supervised. Unsupervised algorithms are generally used for dimensionality reduction or clustering of a dataset and only utilize the features of a dataset and are not utilized in this research. Features are the independent variables of a dataset. The labels or responses of the dataset are the dependent variables - there is some real-world or computational function which is applied to the features to produce the responses. Supervised algorithms utilize both the features and responses of a dataset and serve to find some mapping from the features to the responses.

In the development of a ML model, there is generally some degree of optimization which occurs. This optimization can be baked into the relevant algorithm, modifying internal parameters to produce better results, or the optimization can be of hyper-parameters which govern the construction of the model itself. As an example, consider least squares regression. The internal coefficients are found by a simple optimization problem which minimizes the Mean Squared Error between the model predictions and the actual responses. Because this optimization problem has a unique solution, the maximum performance of this model is dictated by its construction. By modifying the construction of the model (by utilizing hyper-parameters), the performance of the model can be increased. Consider a dataset with a strong second-order relationship between the features and responses. A linear regression will produce poor results, and that performance is the best achievable by that model. By contrast, a second-order regression model will outperform a linear model on this dataset.

In the Ordinary Least Squares model, there exists an analytical solution for the internal coefficients/parameters which yields optimal results in terms of minimizing the MSE loss function. In other ML models, there does not exist an analytic solution for the optimal parameters. In order to find parameters that optimize the respective loss function, an iterative or gradient-based method must be used.

2.1 K-Nearest Neighbors

An example of an unsupervised Machine Learning (ML) algorithm which can be applied to the current problem is K-Nearest Neighbors (K-NN) regression. This algorithm uses a distance metric to rank all samples in the training data by distance to some requested prediction configuration. A prediction for the response at the requested configuration is generated using some function of the K closest samples from the training data. The number, K, nearest neighbors to use is an user-selected input for the algorithm. There are infinitely many distance metrics which can be utilize for the ranking of training samples and the best choice varies by application. One of the simplest distance metrics is Manhattan distance, defined in Equation (1). This measures the absolute distance along all dimensions independently of the others.

$$d_i = \sum_{j}^{N} \left| x_j^{(i)} - x_j^q \right| \tag{1}$$

Where $x^{(i)}$ is the i^{th} training sample, x^q is the query point and j is the dimension index.

Another popular distance metric is Euclidean distance. Defined in Equation (2), this metric computes the straight line distance between two points.

$$d_i = \left(\sum_{j}^{N} \left(x_j^{(i)} - x_j^q \right)^2 \right)^{1/2} \tag{2}$$

The Minkowski metric is a generalization of the Euler distance, introducing a variable parameter for the exponent. Equation (3)

$$d_i = \left(\sum_j^N \left| x_j^{(i)} - x_j^q \right|^p \right)^{1/p} \tag{3}$$

where p is constrained to be at least one. Note that the Minkowski metric becomes the Manhattan distance when $p = 1$, and the Euclidean distance when $p = 2$.

2.2 Gaussian Process Regression

A Gaussian Process (GP) is a generalization of the Gaussian distribution where the GP instead of governing the properties of random variables, governs the properties of random distributions. Gaussian Process Regression (GPR) utilizes a mean function and a correlation matrix in generating a regression of the data:

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f_*} \end{bmatrix} \sim \mathcal{N} \left(0, \quad \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix} \right) \tag{4}$$

where $\boldsymbol{f}$ are the response values of the training points, $\boldsymbol{f_*}$ are the predicted responses at the test (query) points, X is the matrix of training points, and X_* is the matrix of query points. The operator $K(\cdot, \cdot)$ is the correlation operator or kernel such that $K(X, X_*)$ produces a matrix of size n (number of training points) by n_* (number of query points). The resulting correlation matrix provides the covariance structure of the GPR model. This distribution can be manipulated into a form which can be used to predict the response at query points:

$$\begin{aligned} \boldsymbol{f_*} | X_*, X, \boldsymbol{f} \sim \mathcal{N} \Big(& K(X_*,X)K(X,X)^{-1}\boldsymbol{f}, \\ & K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*) \Big) \end{aligned} \tag{5}$$

Equation (5) defines the distribution of possible ranges of responses at any query point. The correlation function can be considered as a distance function as the result of the kernel function is a scalar value which indicates the correlation between the response at a pair of points. If a pair of points have a high positive correlation, the response function is likely to follow the same trend in both locations in the design space. This formulation is useful as the correlation is computed purely as a function of the design variable values, X and X_*. Thus allowing for a covariance structure to be built which includes query points at which the true function value is unknown. The resulting correlation structure informs the distribution of possible response values at the query points (Equation (5)). The mean value and standard deviation of this generated distribution can be used to predict the function response at the query point. The choice of kernel function used to generate the covariance structure is arbitrary and allows for some hyperparameter optimization to be utilized to tune the model. Note that in this formulation, one correlation function is used to calculate the correlation between all pairs of points. [26]

The kriging model is a special application of the Gaussian Process (GP) model which was developed in the field of geostatistics [26]. This model generalizes the calculation of the correlation between pairs of points by allowing unique kernel hyperparameters for each pair. In the standard GPR, a single set of kernel function hyperparameters are utilized for all pairs. Consider the Matérn covariance function:

$$k_{Matern}(r;\nu,l) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{r\sqrt{2\nu}}{l}\right)^{\nu} K_{\nu}\left(\frac{r\sqrt{2\nu}}{l}\right) \tag{6}$$

where ν and l are independent hyperparameters, $K_{\nu}(\cdot)$ is a modified Bessel function, and $\Gamma(\cdot)$ is the gamma function [26]. This correlation function has 2 separate hyperparameters - ν and l. In the case of standard GPR, these two hyperparameters would have the same value in the calculation of the correlation between all pairs of

training points as well as the pairs between training and query points. In the kriging formulation, these hyperparameters would be optimized for each pairwise correlation. There are a wide variety of kriging models which all expand on the standard GPR model in different methods. A common distinction is for kriging models to model the function response as two components - a mean function value and a perturbation component. In ordinary kriging (OK), the mean function response inside the design space is assumed to be the mean of the response at all training points. The residuals of the training points from this mean value are then modeled using the above GPR-inspired method.

2.2.1 Previous Work.

Kriging models have successfully been used to model the flow around hypersonic systems and produce control algorithms for the same [20]–[23]. This series of papers focused on predicting the surface pressure and wall heat flux distributions on a finned missile. The authors constructed an ensemble of kriging models to predict the surface pressure and wall heat flux distributions which resulted at both supersonic and hypersonic speeds and the results were compared to other reduced order modeling techniques such as shock-expansion and Euler solution methods.

2.2.2 Regression Kriging.

One possible pitfall with the Ordinary Kriging method arrises due to the implicit assumption that all observations lie around a mean value and the variation between true values and the sample mean are modeled. In some applications, assuming a constant mean value response across the design space is not valid or useful. A simple and straightforward extension to OK, originally developed in 1994 by Odeh *et al.* [27] is Regression Kriging (RK) [28]. In RK (Equation (7)), instead of an implicitly

assumed constant mean response, a general regression function predicts the local mean response:

$$\hat{z}(s_0) = \hat{m}(s_0) + \hat{e}(s_0) \tag{7}$$

where s_0 is the design point to be queried, $\hat{m}$ is the "drift" or response component modeled with a regression technique, and $\hat{e}$ is the residual of the regression technique at the query point which will be modeled using OK [28]. Assuming the chosen model is an Ordinary Least Squares (OLS) regression model, Equation (7) can be expanded into the form:

$$\hat{z}(s_0) = \sum_{i=0}^{N} \hat{\beta}_i \cdot q_i(s_0) + \sum_{j=0}^{M} w_j(s_0) \cdot e(s_j) \tag{8}$$

where $\hat{\beta}_i$ are the regression coefficients, $q_i(s_0)$ is the ith explanatory variable, $w_i(s_0)$ are the kriging weights obtained from the covariance relations (discussed previously), and $e(s_j)$ are the residuals from the regression function [28]. The N explanatory variables are defined by the user. As an example, if the design space consisted of two dimensions, $s = (x_1, x_2)$, and a first order linear regression model without interaction terms was used, the explanatory variables would be:

$$q_0 = 1$$

$$q_1 = x_1$$

$$q_2 = x_2$$

The user is not limited to using OLS as the regression function and Generalized Least Squares (GLS) is usually preferred [28]–[31]. In fact, there is no underlying assumptions for the regression function, so any methods can be used including Generalized Linear Models, Random Forests, Tree-based methods, Support Vector Machines, etc. [32].

2.3 Neural Networks

Neural networks are a broad category of mathematical models which have their roots in the work done by Frank Rosenblatt in the 1950's and 1960's. Rosenblatt [33] developed the perceptron as a simple computational graph which was meant to approximate the structure of the human vision system. In simplified terms, the vision system receives its initial inputs from discrete cells in the retina. These inputs are passed along channels of neurons until they reach the brain where the signals are dispersed to the first group of computing neurons. Clusters of neurons fire depending on the content of the visual input, sending their signals to the next group of computing neurons, and the process is repeated until the visual input is processed and incorporated by the rest of the brain. In this idealized model, the connections between neuron clusters are initially random and the coordinated firing or activation of clusters can either excite or inhibit a response in the next layer of computational neurons [33].

Figure 1 presents a typical construction of a perceptron model. It is useful to understand the construction of this model and how it operates on data as it forms the base off of which the subsequently discussed models are built. The presented perceptron model has three main components: the inputs, weights, and internal processing of the neuron. There can be an arbitrary number of inputs, each with an associated weight with the addition of a single bias term. The internal processing of the neuron begins with calculating the weighted sum of the inputs (each weighted by their corresponding weight) and adding the bias term to the total to achieve an intermediate value. This intermediate value is subsequently passed through a non-linear activation function. The result from the activation function is the perceptron's predicted response for the input. In Rosenblatt's original idealized model, neurons

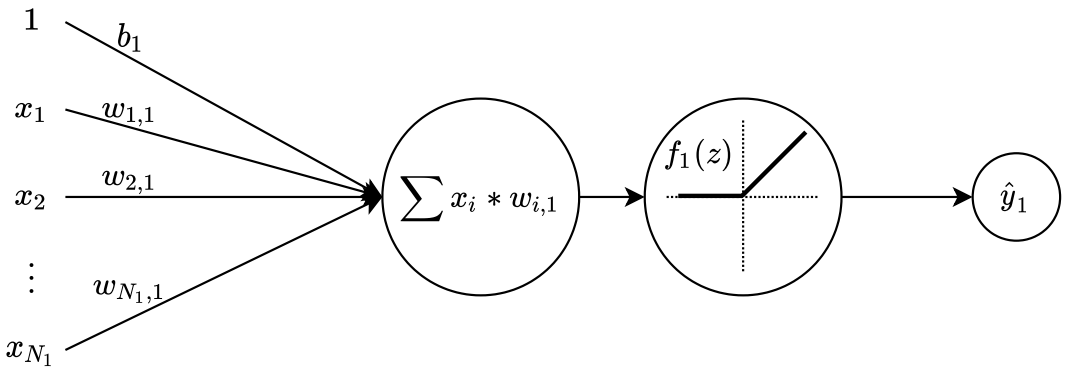


Figure 1. Overview of data operations in the perceptron

could only be in one of two states: fully activated, or completely dormant [33]. This behavior can be emulated in the model using the activation function:

$$sign(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases} \tag{9}$$

It is important for the activation function to be non-linear as a linear transformation of a linear combination of inputs can only produce a linear model. In order to learn non-linear functions, there must exist some non-linearity in the computational graph and the activation functions provide this non-linearity. By utilizing non-linear activation functions, the perceptron and more complex neural networks are able to model non-linear relationships.

The final important note for the perceptron is to examine the equation which it encodes. Computing the value of each node in order is possible, but is computationally inefficient. There is a better, more efficient formulation of a neural network which utilizes linear algebra. For a single neuron perceptron, there exists one weight for each input and these are multiplied pairwise to generate the weighted sum as the first part of the neuron's internal process. If the inputs are aggregated and represented as a vector and the weights are represented as a matrix (where one dimension is of

length unity), the internal weighted sum can be rewritten as a dot product of a vector and a matrix. The bias can be added to this result and then the entire vector passed through the activation function yielding the following equation:

$$\phi(\vec{x} * \mathbf{W} + \vec{b}) = \hat{y} \tag{10}$$

where $\vec{x}$ is the vector form of the inputs, $\mathbf{W}$ is the matrix of weights, $\vec{b}$ is the bias vector, and $\hat{y}$ is the perceptron prediction.

In the case of the perceptron, there is only one layer with one neuron which is densely (or fully) connected to all of the nodes in the previous layer (the inputs). All of the inputs are aggregated at a single neuron and the internal operations of the neuron produce a single output. Figure 2 visually describes the computational graph which emerges when multiple perceptrons are added in parallel. Here, all of the model inputs are available to all of the perceptron neurons in the layer and additional notation is introduced in order to effectively identify which weights correspond to which pairwise connections. The connections between the inputs and the layer of neurons are differentiated using different line styles (solid, dashed, and dotted) with the identifier for specific weights printed over their corresponding connection. The notation used in this research is $w_{i,j}$ where i is the index of the input/neuron where the connection originates and j is the index of the neuron which will receive the weighted value. As a reminder, the two-step operation of the neuron is shown in the first neuron - calculating the weighted sum of the inputs (plus the bias) and passing the results through a non-linear activation function. This operation is identical in all neurons, however the graphic is not repeated for clarity. The values produced by the neurons in the first layer are taken directly as the model outputs, in this case, there are N_2 outputs. In general, this behavior is not preferred as each additional

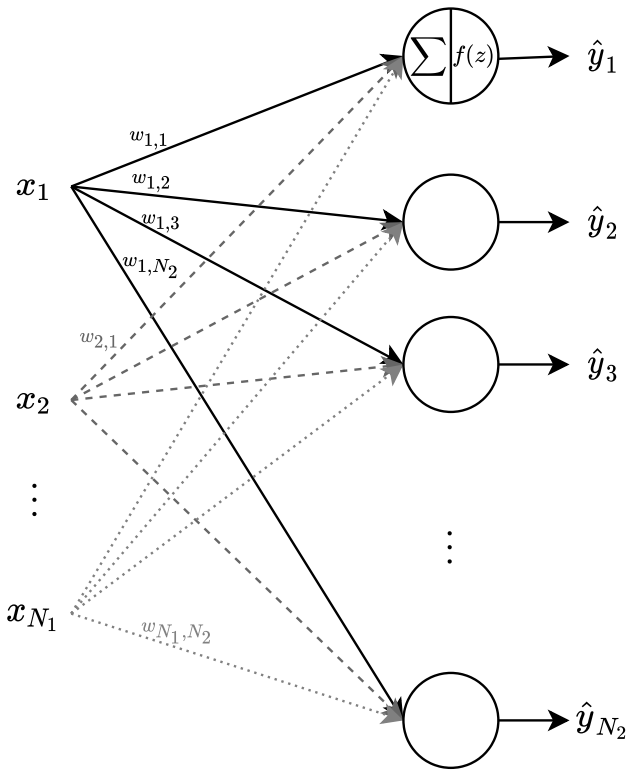


Figure 2. An arbitrary number of perceptron neurons arranged into a layer

computational node added produces an additional model output. This model can be further expanded by adding additional layers.

Figure 3 provides a visual overview of the computational graph of a canonical NN model with one hidden layer. A hidden layer is a layer with which the user does not interact (put values into, or receive values from). As before, the arbitrary number of input nodes are shown in the leftmost layer. The key difference in this model is the addition of an output layer with two neurons which are fully connected to the neurons of the previous layer (in general, an arbitrary number of outputs can be modeled). Similar to the hidden layer, whose inputs are a linear combination of the model inputs (plus a bias), the output layer inputs are the results from the hidden layer neurons (plus a bias). In this architecture, additional computational units/neurons can be added to the hidden layer without affecting the number of outputs. This allows for increasingly complex functional transformations to be learned by simply

adding additional neurons. To this end, Cybenko showed that a NN with one hidden layer and sigmoid activation function can approximate any continuous function with arbitrary precision (is a universal approximator) given enough neurons in the hidden layer [34]. The proven mathematical statement was:

$$|G(x) - f(x)| < \epsilon \tag{11}$$

for any $\epsilon > 0$ and x within the n-dimensional hypercube. The function $f(x)$ was defined to be an arbitrary continuous function of x and $G(x)$ was defined as a finite sum of the sigmoidal activation function outputs - i.e. the output of the three-layer NN [34]. This result holds for any point x within the unit n-hypercube, and is not restricted to regions near training points. Hornik later expanded the mathematical proof and showed that NN's are universal approximators of any function [35]. This property of NN's makes them an attractive choice when attempting to model an unknown function.

In order to accommodate multiple layers, the weight naming convection is augmented once again by adding a superscript to indicate to which layer the relevant weight belongs (i.e. delivers information to). Thus, this entire computational model can be expressed as Equation (12):

$$\vec{\hat{y}} = \phi_2 \left(w^{(2)} * \phi_1 \left(w^{(1)} * \vec{x} \right) \right) \tag{12}$$

where $\vec{\hat{y}}$ is the vector out outputs, $\vec{x}$ is the vector of model inputs, and the terms $w^{(\cdot)}$ are matrices of connection weights with dimensions next-layer-nodes by origin-layer-nodes. The non-linear activation functions of each layer are denoted by ϕ with a subscript according to the layer index. The same activation function is generally applied in all nodes of a given layer, but do not have to be the same for all layers.

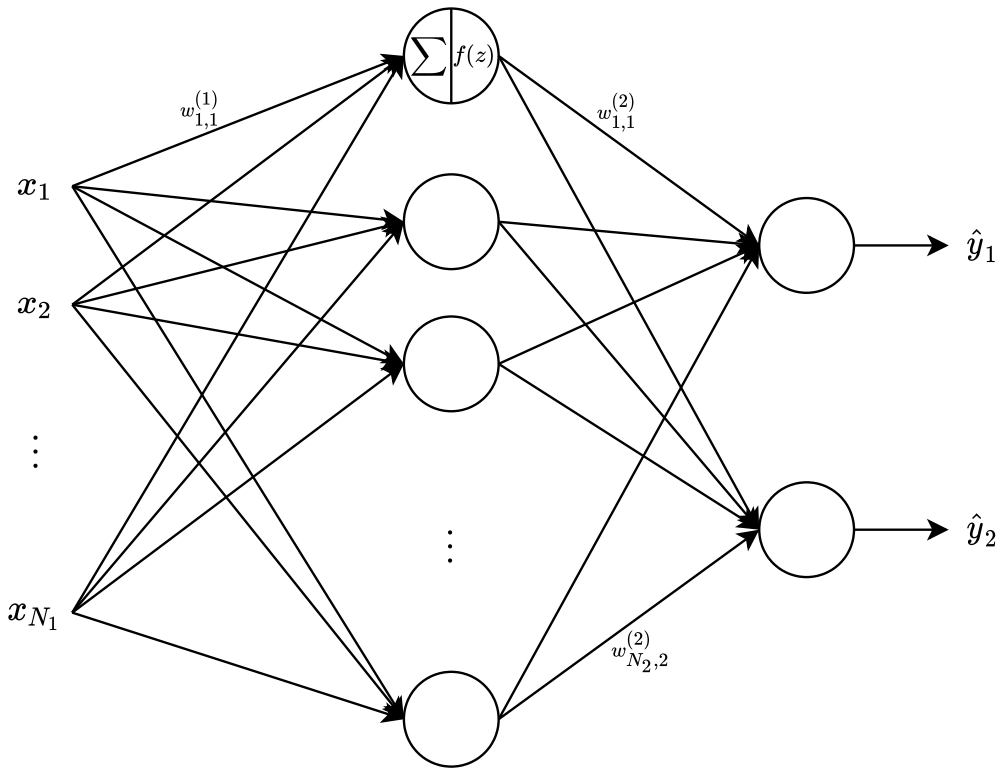


Figure 3. Two layers of neurons with an arbitrary number of neurons in the hidden layer

From this formulation, additional layers can be added as needed and produce Deep Neural Networks. In the general case, NN can have an arbitrary number of inputs, hidden layers, neurons in each hidden layer, and outputs.

In summary, basic neural network models utilize multiple steps of parallel application of non-linear activation functions on linear combinations of layer inputs. All input information is available at all downstream computational nodes and is able to influence the model prediction. This is a powerful model that this able to approximate any continuous function to an arbitrary precision. The process which allows the NN model to extract/learn information and relations from a given dataset is described in Section 2.4.

2.4 Neural Network Training

This section describes the basic gradient-based optimization technique used to train neural network models. Gradient-based optimization seeks to find the minimum of the relevant loss function by moving the current point in the direction of steepest descent. This steepest descent direction can be found by computing the gradient of the objective function (loss function). The computed direction is subsequently used to perturb the input towards values which have an expected loss value which is better (less than) the current guess:

$$x^{n+1} = x^n + \nabla L \tag{13}$$

where ∇L is the gradient of the loss function, x^n is the current query point, x^{n+1} is the next query point which should have a more favorable loss value. A training algorithm is required to tune the internal parameters of a NN model as the weights are generally initialized to be random noise. This training procedure updates the weights and biases in a manner which is expected to yield the most reduction in the loss function.

In the case of neural network models, the independent variables for this optimization problem are the weights and biases of the model. By modifying the weights and biases, the performance of the network can be optimized by minimizing the loss, $L(y, \hat{y})$, between the model predictions, $\hat{y}$, and the targets, y, in the training data. Recall from Section 2.3 that NN are fundamentally built using relatively simple operations - weighted sums and activation functions - arranged sequentially such that the output of one layer is the input of the next. Both of these operations as well as a properly implemented loss function are differentiable. Additionally, the output of the loss function is a single scalar value. This allows for the computation of the partial derivative of the training loss with respect to every weight and bias in the network.

A NN model with three total layers can be written as a closed form equation:

$$\vec{\hat{y}} = \phi_2 \left(w^{(2)} * \phi_1 \left(w^{(1)} * \vec{x} + \vec{b}_1 \right) + \vec{b}_2 \right) \tag{14}$$

where the notation is the same as that in Equation (12) with the addition of the biases for each node denoted $\vec{b}$ subscripted with their corresponding layer number. Consider the Mean Squared Error (MSE) loss function:

$$L = \frac{1}{N} \sum_{i=0}^{N} (\hat{y}_i - y_i)^2 \tag{15}$$

where N is the total number of samples, $\hat{y}_i$ is the NN model's prediction for the ith sample, and y_i is the target response value of the ith sample. The squared error between all of the NN prediction and the target responses are summed for all training samples and normalized by the number of training samples, yielding a single scalar value which indicates one measure of how well the NN model has internalized the training data. The objective of training a NN model is to drive the value of this loss function as low as possible. In order to do so, gradient descent methods are utilized. The gradient of the loss function can be found with respect to all weights and biases of the NN model by utilizing the chain rule of calculus. For simplicity, this process will be demonstrated on a loss value calculated using only one prediction from the NN.

$$L = (\hat{y} - y)^2 \tag{16}$$

First, the direction of steepest descent will be calculated for the weights and biases of the last layer (labeled two in Equation (14)) - $\frac{\partial L}{\partial w^{(2)}}$. For clarity, additional notation will be utilized to represent the output of a hidden layer and the weighted sum which

is the input for the activation function of a layer:

$$\sigma_i = \phi_i\left(z_i\right) \tag{17}$$

$$z_i = w^{(i)} * \sigma_{i-1} + \vec{b}_i \tag{18}$$

where σ_{i-1} is the output of the previous layer. Note that σ_0 is the input vector, $\vec{x}$. The partial derivative of the loss function with respect to the final layer weights can be calculated:

$$\frac{\partial L}{\partial w^{(2)}} = \frac{\partial L}{\partial \sigma_2}\frac{\partial \sigma_2}{\partial w^{(2)}} = \frac{\partial L}{\partial \sigma_2}\frac{\partial \sigma_2}{\partial z_2}\frac{\partial z_2}{\partial w^{(2)}} \tag{19}$$

From the expanded form, values can be substituted for the partial derivatives. Note that the output of layer two (the output layer in this case) is model output in this definition, thus σ_2 can be substituted by $\hat{y}$. The partial derivative of the output of the layer with respect to the input of the layer, $\frac{\partial \sigma_2}{\partial z_2}$, must be found by evaluating the derivative of the activation function, ϕ_2. In this case, assume that the Rectified Linear Unit (RELU) activation function was used such that:

$$\phi(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{20}$$

$$\frac{\partial}{\partial x}\phi(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{21}$$

thus Equation (19) becomes:

$$\frac{\partial L}{\partial w^{(2)}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma_2} \cdot \frac{\partial \sigma_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w^{(2)}} = \left(\frac{1}{2}\left(\hat{y} - y\right)\right) 1 \left(\frac{\partial \phi_2}{\partial x}(z_2)\right)\left(\sigma_1\right) \tag{22}$$

In this formulation, the derivative of the weights for the final layer are in terms of values which have been previously calculated ($\sigma_1,\ z_2,\ \hat{y}$), or values which are simple to calculate. A similar process can be performed to find the partial derivative with respect to the biases:

$$\frac{\partial L}{\partial \vec{b_2}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma_2} \cdot \frac{\partial \sigma_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial \vec{b_2}} = \left(\frac{1}{2}\left(\hat{y} - y\right)\right) 1 \left(\frac{\partial \phi_2}{\partial x}(z_2)\right) 1 \tag{23}$$

This same expansion procedure can be completed for the previous layer:

$$\frac{\partial L}{\partial w^{(1)}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sigma_2} \cdot \frac{\partial \sigma_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial \sigma_1} \cdot \frac{\partial \sigma_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w^{(1)}} \tag{24}$$

$$= \left(\frac{1}{2}\left(\hat{y} - y\right)\right) 1 \left(\frac{\partial \phi_2}{\partial x}(z_2)\right) w^{(2)} \left(\frac{\partial \phi_1}{\partial x}(z_1)\right) \sigma_0 \tag{25}$$

The terms in this expression again are either simple to calculate (as the closed form of the loss function, activation function, and their derivatives are known) or have been calculated previously, during the forward pass (i.e. prediction calculation). Note that the first three terms in this expression are identical to those in Equations (19) and (23). Therefore, by calculating the directions of steepest descent for the weights and biases of the layers in reverse order, the gradient can be accumulated using multiplication and then applied to the current layer by multiplying by one additional term.

2.5 Convolutional Neural Networks

Convolutional Neural Network (CNN) models process information which is in a grid pattern (e.g. images) and were inspired by the structure of the visual cortex of the brain [36]. Hubel and Wiesel [37] found that primate brains contained hierarchies of visual processing cells which responded to specific patterns in visual input. "Simple cells" responded strongly to simple patterns where only parallel lines of "on" and "off"

were important. "Complex cells" required more complex visual patterns to elicit a strong response such as movement directions and orientation. The next hierarchy of cells, "hypercomplex" cells, required additional conditions such as shape length, size, and position to also be satisfied in order to elicit a strong response [37]. Fukushima and Sei [38] proposed a neural network model in 1980 which incorporated a similar hierarchical architecture where neurons in the first layer would experience stronger activation if certain simple patterns were identified in the input. Subsequent layers would then identify patterns of patterns in the previous layers, sequentially identifying more complex patterns from the original input data [38].

Modern CNN models have the ability to learn spatial relations in a sequential hierarchy from low to high-level patterns with each layer of pattern recognition operating on the previous layer of patterns where the relations are not dependent on the absolute location in the input. A standard construction of a CNN, is presented in Figure 4, utilizes convolution, pooling, and fully connected layers where the convolution and pooling layers are responsible for identifying patterns and the fully connected layers are responsible for mapping the extracted patterns into a classification. In general, image data is stored in 2D arrays. Figure 5 illustrates how the convolution operation produces an output image from an input image. The CNN shown has an input image of size 5×5 and one kernel of size 3×3. No padding is utilized, thus this CNN produces an output of size 3×3. The input image and the kernel are both arrays of numerical values, however these values have been omitted in this example for clarity. One value in the output image (or array) is generated by computing a linear combination of values in the input using the weights contained in the kernel. Each possible set of 3×3 pixels in the input image correspond to one pixel in the output image. In the input image of size 5×5, there are nine possible 3×3 groups of pixels, thus there are a total of nine pixels in the corresponding output image. Note that

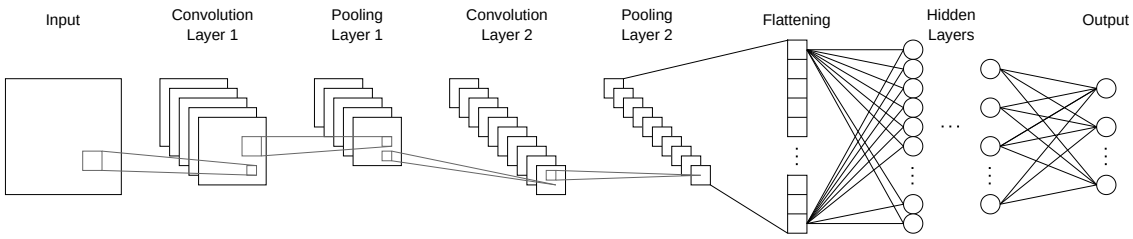
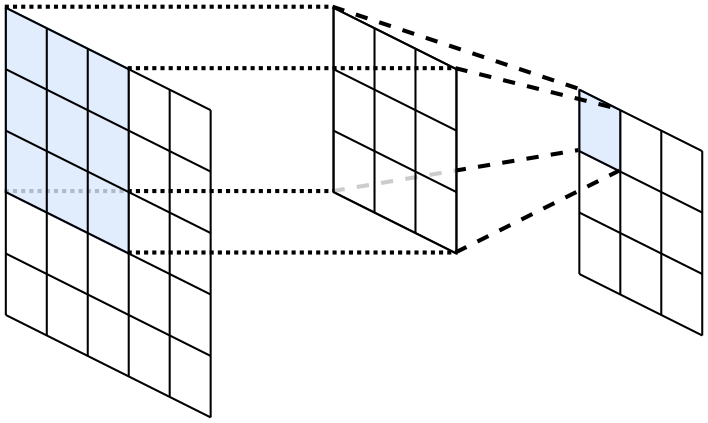


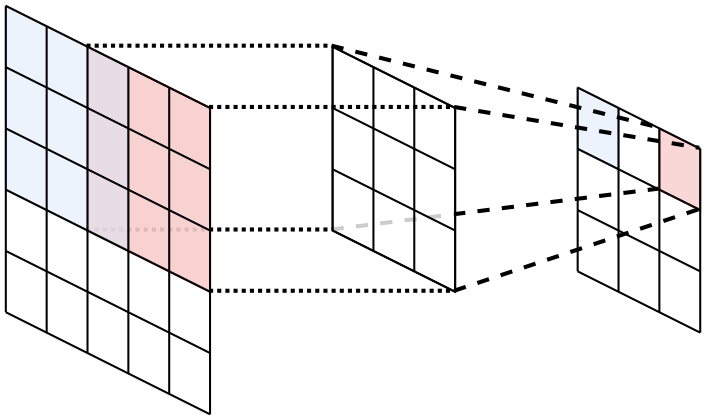
Figure 4. Standard CNN architecture with convolutional layers and pooling layers followed by a flattening operation and fully connected neural network. The final output of the model is a vector of values, generally used to generate classification predictions.

the groups of pixels are allowed to overlap in this construction. Figure 5 highlights three convolutional operations to illustrate how the information in the input image is condensed into an output of smaller size. Note that the same convolution operation with the same kernel is utilized to generate every pixel in the output image. Starting in the top-left (Figure 5a), the 3×3 kernel is matched up to a 3×3 subset of the input image. This subset is subsequently convolved with the 3×3 kernel (the elementwise multiplication is denoted by the dotted lines) and the elements are summed into a scalar value which is inserted into the output array (denoted by the dashed lines). In this manner, the strength of the spatial signal described by the kernel in that specific subset of the input image is encoded into the output image. This is repeated with the kernel (whose value remains unchanged) for every available subset in the input image until the full output image is generated. In Figure 5 only three of the nine total positions are shown. The three selected pixels in the output image have the same color as the input image pixels which were used to generate their values. Note that the subsets which generate the output pixels are unique, but do exhibit overlap with neighboring subsets.

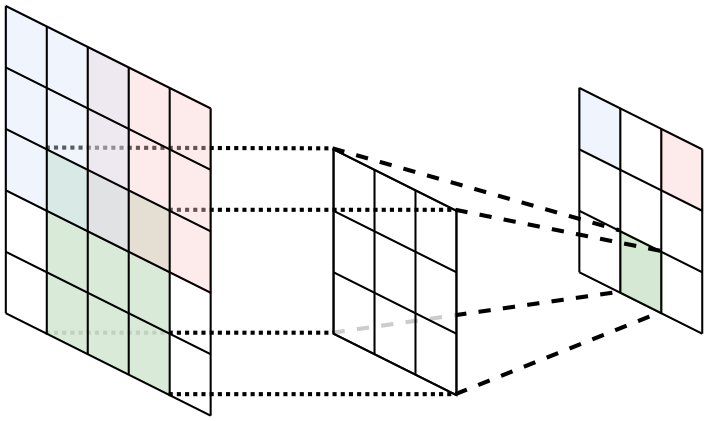
A numerical example of a convolution operation over an image with one kernel is presented in Figure 6. The set up for this example is the same as the previous example: a 5×5 input image is convolved with a 3×3 kernel which produces a 3×3 output image. The kernel in this example has been hand-crafted to match a



(a)



(b)



(c)

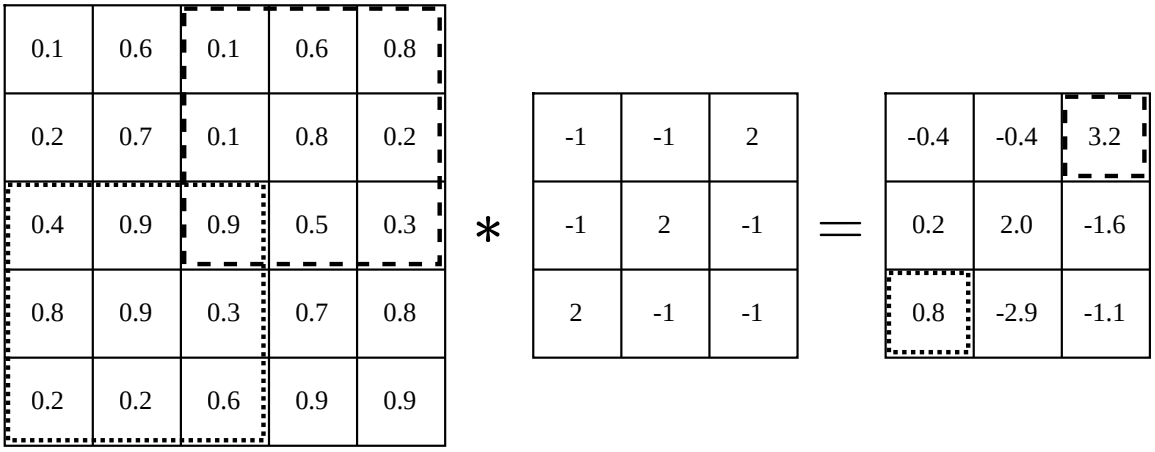
Figure 5. Demonstration of a convolution operation performed on an input image of size 5×5. The kernel is positioned over a set of pixels in the input image and an elementwise multiplication is performed (represented by the dotted lines), subsequently the resulting weighted sum (represented by the dashed lines) is inserted into the output image. This is repeated for every set of input pixels (3×3 in this case) to generate the output image. For clarity, only three sets of input pixels are shown.

specific spatial feature, namely a defined diagonal line pointed from the lower left to the upper right. If a subset of the image strongly matches this spatial feature, then the convolution with the kernel will result in a larger positive pixel value for the corresponding location in the output image. The construction of the output image follows the process described previously and shown in Figure 5. The output image is populated by starting with the kernel aligned with one corner of the input image. The pixel values covered by the kernel are then used to compute a weighted sum where the weights are the corresponding values in the kernel. This weighted sum is subsequently inserted into the output image and the kernel is moved over by one position at a time and the resulting weighted sums are inserted into the corresponding locations in the output image until all subsets of the input have been convolved and the output image is fully populated. For reference, Figure 6a identifies two subsets of the input image pixels used to generate the corresponding pixel values in the output image. Figure 6b overlays a colormap over the input image, kernel, and output image for a visual interpretation of the results. As stated previously, the kernel used was hand-crafted to have a strong positive response to diagonal lines pointing up and to the right. Visually, there is a relatively well defined line in the input image along the top half of the minor diagonal. There is still some noise in the region, however the aforementioned line is more defined or distinct from the surroundings in the upper right than in the center of the image as the pattern in the center has large values in the 3×3 area which would not correspond to positive values in the kernel. Therefore, we would expect a strong positive response from this kernel for the upper right region and a less strong response in the center. There are no other regions of the input image which contain the distinct diagonal pattern of interest defined in the kernel, so it is expected that the rest of the output image would not exhibit a strong positive response. The bottom middle 3×3 section of the input image contains a pattern which

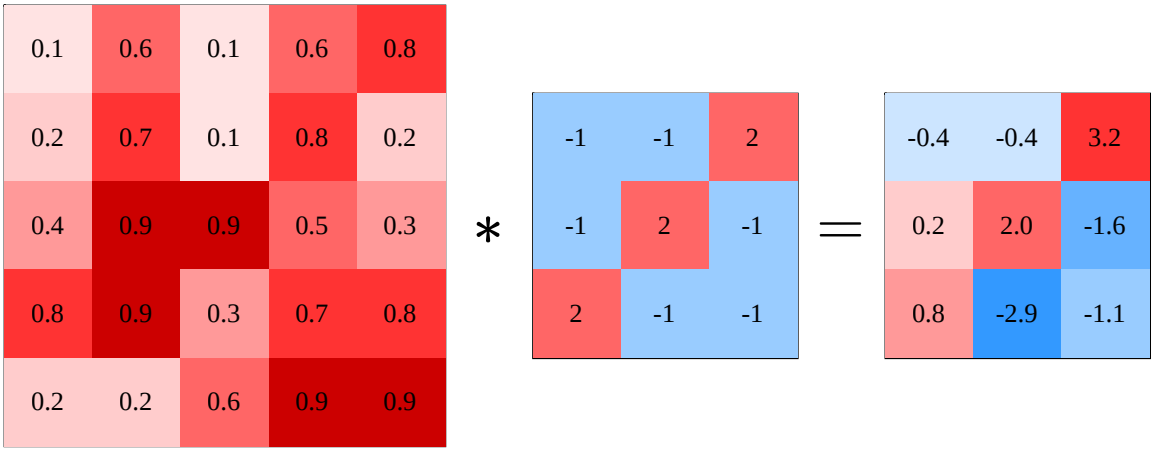
is more similar to the additive inverse of the kernel as there are small values along the diagonal of interest and large values in the off-diagonal positions. The corresponding kernel response to this region is expected to be strongly negative. These expectations are shown in the output of Figure 6b. There is a strong positive response in the top-right corner, a weaker positive response in the center, and a strong negative response in the bottom-middle. The rest of the output shows relatively weak responses which indicate that the pattern (or anti-pattern) did not match the other sections of the input image well.

This numerical example shows that a convolution operation can identify the spatial pattern defined by the kernel in the local spaces of the input image. Additionally, the response of the kernel can identify not only the degree to which the pattern matches the local area of the input image, but also if the additive inverse of the pattern is more prominent. In the case where the anti-pattern is more pronounced in the input, the corresponding output of the kernel will be produce a stronger signal in the opposite direction (less than zero in the case of the presented example). Convolutional layers are typically followed by non-linear activation function such that, after the previous convolution procedure is applied, the resulting image/array of values is passed through the activation function (in the same manner as described previously).

Two important hyperparameters of a convolutional layer are the kernel size and number. The shape of the kernel is generally square with side lengths of three, however kernels of size 5x5 or 7x7 may also be used. The number of kernels in a layer indicates how many patterns can be matched in the input tensor (because each kernel is one learnable pattern). Because all elements in the kernel must be matched to a value in the input tensor, the size of the output of a standard convolution is smaller than the input. The tensor size can be retained by padding the input perimeter with zeros or other values derived from the input. [36]



(a) The kernel (center) is convolved with the input image (left) to produce the output image (right). In the convolution process, the weighted sum of a subset of the input image is computed and inserted into the output image with the weights being the values in the kernel. As an example, the subset of input pixels used to generate the top right output pixel are identified with a dashed box and the pixels which define the bottom left output pixel are boxed in a dotted line.



(b) This example is repeated from above with the addition of a colormap to better visually illustrate the operation. The kernel is designed to produce a large positive response where the input has a line of three large value pixels in a diagonal pointing up and right and the surrounding pixels have a small value. After the convolution operation, the output image is a measure of how strongly the kernel responded to each spacial region of the input image.

Figure 6. Example convolution operation with (a) only the values within the input image and kernel and (b) with the same operation with the values mapped to a colormap.

An important feature of CNN's is that the kernels perform the same operation regardless of their relative position over the input. This sharing of the weights with all positions introduces some key advantages over the fully connected neural networks discussed above. First, local patterns can be detected anywhere in the image regardless of absolute location. Second, reduction in the number of parameters required to learn patterns leads to an increase in computational efficiency. Without padding (or with stride greater than unity), the size of the working tensor is decreased through the CNN model, requiring fewer and fewer parameters. [36]

Pooling layers offer a method of reducing the size of the working tensor and/or distilling the output of a convolutional layer. Common methods include max pooling and average pooling [36]. These layers also have a kernel size hyperparameter which determines how the output is calculated. For max pooling, the maximum value in the window is propagated to the output, and average pooling calculates the mean of the values in the window.

Finally, after all convolutional and pooling layers are complete, the final output tensor is flattened into a 1D array and passed into one or more fully connected layers. The operation of these layers is discussed above and transform the distilled patterns generated by the convolutional kernels into a prediction for the relevant problem.

CNN models build intermediate results from local information in the previous layer, therefore the concept of the Field of Vision (FoV) of an arbitrary layer output can be developed. The FoV of a result describes how much of the original input image was considered in the calculation of that result. The FoV for a simple CNN model with padding and no pooling (such that the intermediate results and the final result have the same dimensions as the input) is given by:

$$\text{FoV} = L \cdot (k_{size} - 1) + 1 \tag{26}$$

where L is the number of convolutional layers and k_{size} is the side length of the kernels (all kernels are assumed to be square and of the same size). For the trivial case of zero convolutional layers (i.e. the output is identically the input), the FoV is unity. Thus only input values within a radius of one were considered in generating each output value - this is a one-to-one transformation. Figure 7 illustrates the dependency structure through two convolutional layers with kernels of size $k_{size} = 3$. The analytical result from Equation (26) is $\text{FoV} = 2(3-1)+1 = 5$, thus every value in the output is affected by input values within a radius of five. The same conclusion is obtained in Figure 7 by examining the dependence of the center value through the previous layers. Note that the the image size does not change through the CNN, therefore there is some padding applied. Working backward from the selected output value, one and only one position of the 3×3 layer 2 kernel influences the value at that location. Similarly, the value of each of the nine pixels identified in layer 2 are determined by a unique 3×3 subset of pixels in layer 1 (each layer 2 pixel is the output of one 3×3 kernel operating on the first layer). These nine kernel positions in layer 1 do not correspond to disjoint sets of layer 1 pixels and their overlaps can be visualized in the form of a heatmap, presented in the graphic. The value of the single pixel in the final layer is a function of the values of nine (square of side length three) pixels in layer 2 and those nine pixel values are a function of only 25 (square of side length five) of the pixels in layer 1. Therefore, the value of the single pixel in the final layer is a function of a set of pixels within the square of side length $\text{FoV} = 5$ in the first layer and no other pixels. In effect, this pixel in the final layer only has knowledge of or can see a certain subset of pixels in the input image. This pixel is not aware of and cannot incorporate any information from the input which does not reside within its FoV. The calculation of the information availability can be changed by the addition of pooling layers which condense an intermediate image

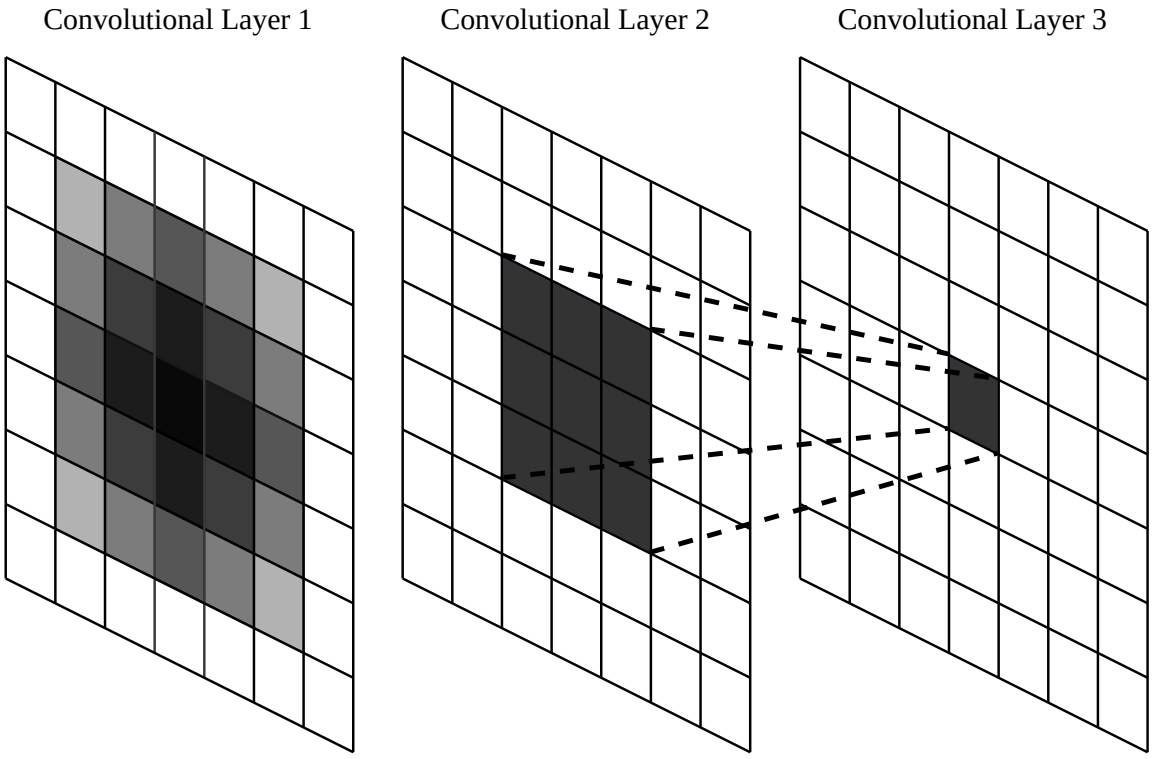


Figure 7. The FoV of a CNN is related to the kernel size and total number of layers. In this case, 3x3 kernels are used in all layers. Starting from the final layer and using the preceding kernel, the FoV can be determined by iteratively tracing backwards through the layers and determining which pixels from the previous layer influence a pixel in the current layer. In this case, the CNN's FoV is five, therefore the value of a pixel in the final layer is affected by the values of pixels within a five-by-five location in the first layer.

into a smaller image, increasing the FoV in a similar manner to a convolutional layer. This localization of available information marks a major distinction between CNN models and the standard NN model discussed previously. The standard NN make available all of the input information to all computational nodes (either directly as in the first hidden layer or indirectly through the outputs of the hidden layers) allowing any computational node to use information from any part of the input. This is not the case in the CNN model, the localization of information forces computational nodes to operate only on a spatially localized subset of the input information.

The construction of the kernels in convolutional layers can be modified such that the inputs and outputs can have different and arbitrary number of channels. Each output channel has one kernel which is used to generate it and the kernels must have the same number of channels as the input image. The previous examples, Figures 5-7 all illustrate an input image of one channel being transformed into an output image with one channel. If additional output channels were desired, the process previously described would be repeated n times where n is the desired number of output channels. The kernels which generate these output channels would all be the same size and operate in the same manner, only differing in their constituent weights. Figure 8 presents the case where a layer input has multiple channels. If the number of input channels is greater than one, the utilized kernel would have the same number of channels as the input (in the examples, each 3×3 kernel has one channel). If the input image had two channels, then the kernels would be of size $3 \times 3 \times 2$. The same sliding, convolution operation would be used to generate a single output channel where a subset of pixels from the now 3D input would be elementwise multiplied with the 3D kernel. The sum of the result would be added to the kernel's scalar bias term to produce a final output value which is subsequently inserted into the output image. Similarly to the neural networks described in Section 2.3, the weights inherent to the kernels are learned or optimized using a gradient-descent based algorithm as described in Section 2.4. The gradient of the loss function can be computed with respect to all weights and biases in the network and updated utilizing a gradient-descent algorithm such that the loss of the network decreases.

2.5.1 Fully Convolutional Networks.

The typical use case of CNN's has been for image classification problems where the input is a multidimensional array of values representing the image and the cor-

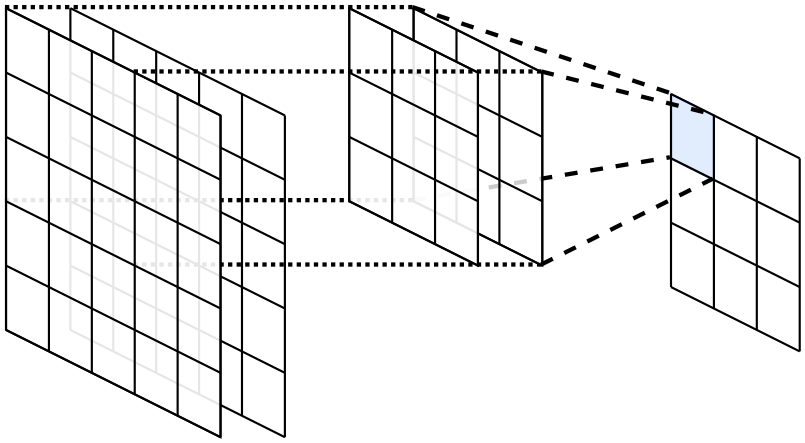


Figure 8. Convolution of multiple input channels utilize a kernel with the same number of channels as the input. Similar to the single-channel case, the weighted sum of corresponding pixels results in a single value and are inserted into output.

responding output is a scalar value (or vector encoding of one) which prescribes a response - i.e. many pixels to one value [39]. However, there are certain classes of problems where producing a single valued prediction is not the ideal solution strategy. One such application is in the identification of discrete objects in an image, also called segmentation. For these applications, the objective of the CNN model would be to flag the pixels which belong to the identified object(s), distinguishing them from the rest of the image [39], [40]. In order to accommodate this class of problems and utilize the spatial pattern recognition of CNN models, the manner in which the standard CNN (Figure 4) generates predictions must be modified. By replacing the fully connected, hidden layers from the standard model construction with additional convolutional layers. The outputs from the final convolutional layer can subsequently be upsampled to the original input dimensions. The manner of upsampling utilized can be determined by the user [40].

2.6 U-Net Architecture

The U-Net is a CNN model which builds off of the idea of the fully convolutional network described in Section 2.5.1 and was developed by Ronneberger *et al.* [39] in

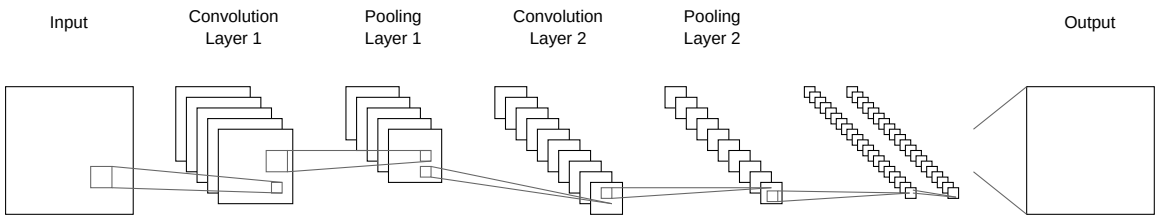


Figure 9. Fully convolutional network architecture where the densely connected layers from the standard CNN (Figure 4) are replaced by additional convolutional layers. The final layer of which is then used to construct a pixel-wise prediction for every pixel in the input image.

2014 to perform biomedical image segmentation. This CNN architecture builds on the previous fully convolutional network [40] work which was able to produce per-pixel predictions on a given input image instead of only produce a vector of classification predictions. In the medical imaging field the standard image classification is not preferred (identifying if certain objects are in an image), instead it is advantageous to localize certain classifications to a set of pixels such as classifying/segmenting a likely tumor from healthy tissue (where a certain object is in an image).

The template U-Net architecture is presented in Figure 10. The U-Net builds on the previous fully convolutional network architecture by adding an expanding path to the end of the contracting path in order to generate high resolution labels (the same size of the input). The contracting path encodes the information from the input into progressively smaller and smaller arrays of values where each value is a representation of more and more complex patterns and relations in the input. In the expanding path, those distilled patterns are upsampled and further transformed using additional convolutional layers. The inputs to each convolutional layer on the expanding path also has skip connections to the corresponding layer on the contracting path where the output from the contracting path layer is concatenated to the upsampled output from the previous expanding path layer. The reintroduction of information from previous layers is meant to provide context for the patterns generated through the expanding layer. [39]

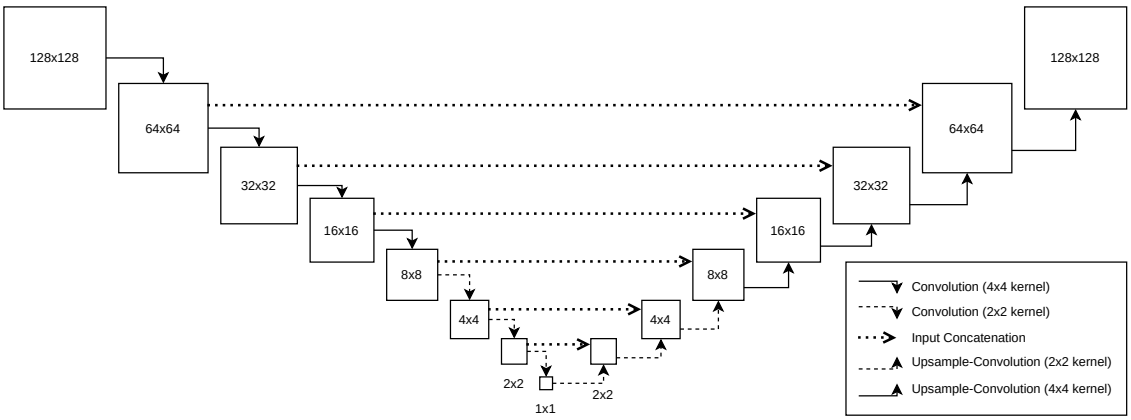


Figure 10. U-Net architecture consists of a condensing leg and a expanding leg. The resolution of the intermediate results is reduced through the condensing leg and increased through the expanding leg. The inputs for the layers in the expanding leg include both the upsampled results from the previous layer concatenated channel-wise with the corresponding outputs from the condensing leg. This specific architecture is utilized by [24].

Utilizing this architecture, the authors found that they were able to perform this per-pixel classification (segmentation) more accurately than other competing models utilizing few training samples and augmenting their dataset with realistic deformations of existing samples [39]. This research demonstrated that this network architecture can perform well on image-to-image transformation problems where location-agnostic patterns must be identified and transformed in-place into a different distribution.

2.6.1 Previous Work.

The U-Net architecture has been adapted for use in the field of fluid dynamics. Uses have included reconstruction and prediction of steady state, incompressible flows [24], [41] as well as forward prediction in time-accurate turbulent flows [42], [43]. Most applicable to this work is the work by Thuerey *et al.* [24] where the flow field around airfoils in high Reynolds number flows were reconstructed using a standard U-Net architecture (Figure 10). The authors generated Reynolds Averaged Navier Stokes

(RANS) solutions for over 1505 airfoils from the UIUC database [44]. The freestream conditions utilized had Reynolds numbers in the range from $[5 \times 10^5, 5 \times 10^6]$ and angle of attack between [-22.5, 22.5] degrees. The authors used the U-Net architecture presented in Figure 10 to reconstruct the x-velocity, y-velocity, and pressure fields (three channels) utilizing the freestream x and y velocity as well as a mask denoting the geometry of the considered airfoil configuration (three channels). Overall, the optimized U-Net could reconstruct the flow fields with accuracies around 4% (for sample sizes consistent with those used in this research).

2.7 Multi-Scale Network Architecture

Santos *et al.* [25] developed the Multiscale Neural Network (MS-Net) architecture to predict the velocity of fluid flow in porous mediums. In order to accurately model the fluid flow through a porous medium, not only must the complex relationship between the local structures (micro-structure) and fluid velocity be learned, but the distributed structures must also be considered. Porous mediums may contain large, spatially varying features such as fractures which can greatly impact the route of bulk flow. Therefore, for this class of fluid flow problems, it is important not only to understand and model how macroscopic features affect bulk flow, but also how the microscopic features add additional complexity to local fluid flow.

Recall from the discussion of FoV (Section 2.5) that in order for one volume to have an effect on a CNN's prediction in another volume, the first volume must exist in the computational history of the second. If these volumes are not in the same local region (such as in the case of a fracture redirecting flow), a combination of many convolutional layers and/or large kernel sizes is required. The presence of such multi-scale interactions in this specific fluid flow problem thus drives standard CNN models to be deep (many layers) and/or wide (many weights/kernels per layer). These CNN

models can quickly become unwieldy, requiring lots of computational resources and time to train. Santos *et al.* found that achieving the required FoV by adding layers to their single scale CNN model limited the allowable sample resolution to 256^3 due to exhausting current GPU hardware capabilities. The MS-Net was developed to address and overcome this limitation by breaking the model into a set of smaller, more manageable CNN's. [25]

Figure 11 presents an overview of how the multiple CNN sub-models operate in a serial manner, feeding the output of one model into the next. The major components of this pipeline include the n CNN sub-models, a coarsening function, and a special refinement dubbed masked refinement (discussed later). The feed-forward process for the MS-Net begins with coarsening the input features $n-1$ times using a standard average pooling function. This yields the low-resolution input for the first sub-model (sub-model n on the left) and part of the inputs for the rest of the sub-models. Note that each sub-model receives a different resolution of the input features and the final sub-model (subscripted 0) receives the full-resolution input feature data. The first sub-model generates a prediction from the lowest-resolution input feature. This prediction is subsequently upsampled using the masked refinement algorithm such that the result is the same dimension as the input feature for the next sub-model. This sub-model receives as input the feature of corresponding resolution concatenated with the upsampled prediction from the previous sub-model and produces a result of the same resolution. The prediction of the current sub-model is obtained by performing an elementwise addition of the current output result with the upsampled prediction of the previous sub-model. This process of upsampling, concatenation, and elementwise addition is repeated for the remaining sub-models. The prediction of the final sub-model is the prediction of the MS-Net. [25]

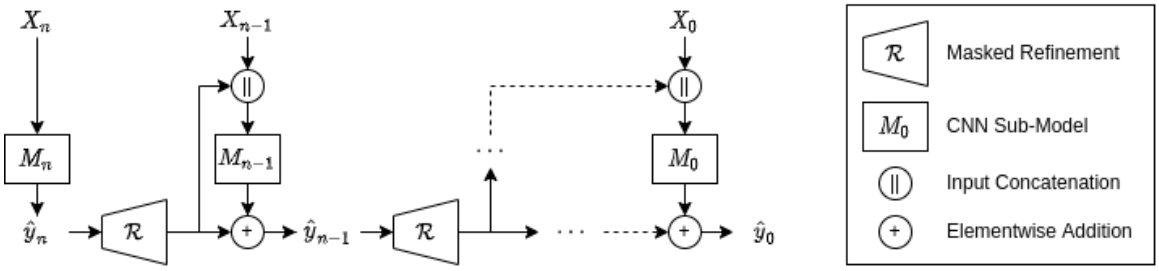


Figure 11. Overview of data operations in MS-Net. Adapted from [25]

Santos *et al.* implement an upsampling algorithm, which they term Masked Refinement, that is different than the transpose convolutions present in the U-Net (Section 2.6). The authors argue that upsampling the results between sub-models using transpose convolutions adds undue additional complexity and computational cost. This is because, the transpose convolution layer has tunable parameters itself which must be learned through training along with all of the other parameters. Instead, the authors choose to adopt a deterministic upsampling - Nearest Neighbor upsampling [25]. This operation increases the resolution of the given image by splitting each pixel or voxel in half along each axis (e.g. one pixel becomes four or one voxel becomes eight). The Nearest Neighbor upsampling operation is better suited for this type of physics problem than the transpose convolution. Consider the coarsening process, during the change in resolution, there is no radical change in the flowfield. Therefore, there is no need for the upsampling process to posses the ability to radically change the image. Additionally, the purpose of the sub-models in the MS-Net architecture is to perform these image transformations, thus using transpose convolution layers could introduce redundancy at the cost of increased computational complexity.

2.7.1 Loss Function.

Santos *et al.* implemented the loss function shown in Equation (27) for their implementation of the MS-Net. This function contains two summations, one over

each training sample and the second over the scales present.

$$L = \sum_{s} L_s = \sum_{s} \sum_{i} L_s^i = \sum_{s=0}^{S} \sum_{i=0}^{N} \frac{\langle (y_{i,s} - \hat{y}_{i,s})^2 \rangle}{\sigma_{y_s}^2} \tag{27}$$

where S is the number of scales, N is the number of training samples (in the current mini-batch), $y_{i,s}$ is the target response for sample i at scale s, $\hat{y}_{i,s}$ is the predicted response for the same. A volumetric average is applied to the squared element error between the two images and the result is scaled by the variance of the response at that scale, σ_{y_s}. The authors applied this variance scaling because of the response variable ranged a few orders of magnitude. [25]

2.7.2 Masked Refinement.

Standard nearest neighbor upsampling preserves the amount of information in an image and simply increases the resolution of the image by dividing each pixel in half along each dimension and assigning the original value to the new pixels. Near the boundaries of the physical geometry, this means that a new pixel generated from the subdivision could actually be inside the geometry. These pixels have to have zero-velocity by definition, but they have been assigned the value of the pixel from which they were generated which could have been non-zero. The authors found that using this method for upsampling decreased model performance as the model would have to learn additional spatial relationships in order to enforce zero velocity inside of any present geometry [25]. In other words, the MS-Net sub-models would need to incorporate additional specific information into their internal parameters thus reducing the overall model's capability to fit other spatial relations of interest. In a very loose formulation, this can be explained by some patterns (and therefore kernels) being used to enforce the zero-velocity conditions. However, because the number of kernels within a model is constant for a constructed instance, using some for this

reason would leave fewer patterns to model the internal flow characteristics which are of interest (varying the number of kernels is utilized for hyperparameter optimization). Fewer patterns/kernels dedicated to learn the flow characteristics could lead to a reduced fidelity in the flow predictions (as the authors did find). If there was a way to inject this zero-velocity information into the upsampling operation, then the MS-Net would not have to generate the previously described patterns in order to enforce the zero-velocity conditions. This would free internal resources to instead focus on matching patterns for flow characteristics instead, increasing performance for the same total number of kernels. This information injection is possible through the process of masked refinement [25].

The process of masked refinement utilizes the geometry masks at different scales. In the process of generating the input features, a binary mask denoting which pixels correspond to geometry and which correspond to free space/the flowfield (labeled with 0 and 1, respectively). This mask can be generated for every scale required by the model and contains the true geometry locations where the zero-velocity condition must be enforced. Figure 12 shows the a round trip of input data where it is first coarsened and subsequently upsampled using the standard nearest neighbor algorithm and the masked refinement algorithm. The top row from left to right shows the coarsening process, performed using average pooling, applied to a full resolution image (left) three times to produce scale three (right). This smallest resolution image is subsequently upsampled using both the nearest neighbor method (middle row) and the previously described masked refinement algorithm (bottom row). For the nearest neighbor refinement, the resolution of the image is increased by dividing each pixel along both the horizontal and vertical axes. This process introduces additional resolution but does not modify the information contained in the image, as a result the full scale image (left) is visually identical to the lowest resolution image (right). Using

this method, there are pixels which should be zero but instead have non-zero values. For the masked refinement algorithm (bottom row), the full resolution image (left) is not identical to the lowest resolution image. The geometry (pixels with zero value by definition) matches exactly with the target input image (top row, left image) despite that information being destroyed in the coarsening process. The same is true for every intermediate image in the masked refinement case - the geometry is exactly the same as in the target labels (top row). Note that in the regions which are not near the border of the geometry, there is no difference from the nearest neighbor upsampling. This is the expected behavior and it is the function of the MS-Net to perform the iterative refinement on these lower fidelity images to produce images which better match the target responses. Masked refinement allows for parameter-free upsampling where the resulting image already has the zero-value boundary condition enforced at all pixels where a wall exists.

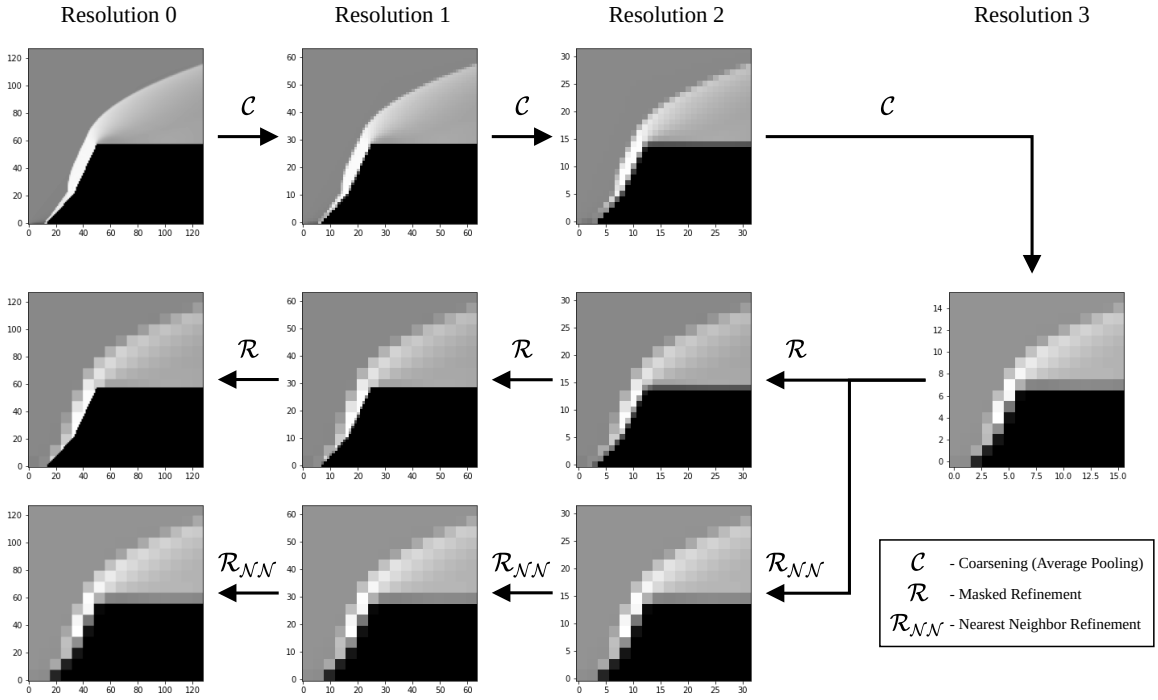


Figure 12. Input features are coarsened using average pooling for each successive MS-Net sub-model (top row). The naïve nearest-neighbors upsampling method (bottom row) merely divides pixels, retaining the same amount of spacial information as the coarsest resolution. Masked refinement (center) enforces the geometry conditions in each resolution, introducing information and reducing the amount of information required to be learned by the MS-Net model.

III. Model Implementation

The specific implementations of the Machine Learning (ML) models presented in Chapter II are detailed in the following sections. The models discussed are the k-Nearest Neighbors and Gaussian Process regression models and the two Convolutional Neural Network-based models: U-Net and Multiscale Network. The process of generating the training data from simulation output is subsequently described followed by the method used to extract the wall values from the model predictions.

3.1 K-Nearest Neighbors

The k-Nearest Neighbor (k-NN) algorithm computes the distance of a query design point from the samples contained in the training set. The k nearest points (neighbors) as determined by the distance function are used to compute the prediction at the query point. The predicted response at the query point is computed by applying a weighted sum to the responses of the k nearest neighbors where the weighting function is restricted to a uniform weight, irrespective of the distance between query points and the neighbors or an inverse distance weight such that the prediction at a design point in the training set will yield exactly the response of the respective training sample. The weight function was limited to these two methods as these are the methods provided in the Python library *sklearn* [45] implementation which was utilized to build the model. Integrating custom weight functions is possible, but was beyond the scope of this research.

Two implementations of the k-NN model were constructed: one to directly model the surface pressures and one to predict the pressure field around the hypersonic object. The *sklearn* implementation of the k-NN algorithm can fit responses of arbitrary size. The input variables used were the design variables independently standardized.

The hyperparameters of the k-NN algorithm are the weighting function, the number of neighbors considered, and the coefficient of the Minkowsky coefficient.

3.2 Gaussian Process Regression

The Gaussian Process Regression model was built in three distinct ways. The most straightforward construction utilizes only the existing *sklearn* [45] Gaussian Process Regression method. This method implements an algorithm similar to Algorithm 2.1 from [26]. This implementation utilizes a global kernel function which is provided as an input. When the model is fit to the data, the package performs an automated optimization routine to find the parameters of the kernel which maximize performance of the GPR on the training data. For this method, there is one degree of freedom in the hyperparameters which is the choice of kernel/correlation function. The kernels investigated for this research were the default kernel (some product of a constant and RBF), the standard Radial Basis Function (RBF), the exponential sine squared, and the Matern kernel.

The second GPR model type was built using the main idea from Regression Kriging where a regression model is utilized to model the deviation of the mean through the response space and the residuals are modeled by a GPR model (Equation (7)). The model utilized for the regression of the mean response is the k-NN model described above. For this configuration, there are a total of three hyperparameters, k and p for the k-NN regression model and the correlation function for the GPR model. Note that the k-NN model used for mean regression in this configuration utilizes the uniform weighting function and the distance-based weighting function is not utilized. The GPR model is trained on the residuals between the mean regression function and the target responses. The distance-based weighting function for the k-NN model yields the target response exactly at all training points, thus if the distance-base

weights were utilized, all residuals for the training set would be zero making the GPR model obsolete. This combined model was generated in two different manners - optimizing all hyperparameters concurrently and optimizing the hyperparameters of the k-NN model first and subsequently the hyperparameter of the GPR model.

3.3 U-Net Architecture

Thuerey *et al.* [24] have provided the source code for their U-Net model implementation under the Apache License 2.0 to which minimal changes were made to conform the model to the current work. The model has the structure presented in Figure 10. This architecture is comprised of a series of seven convolutional layers which form the compression leg followed by the same number of upsampling and convolutional layers to yeild an output of the same size as the input. Note that there skip connections connecting corresponding layers in either leg. The outputs from the compression leg of the network are concatenated onto the upsampled outputs along the expansion leg, increasing the number of input channels for each of those layers.

The two hyperparameters for this model are the number of channels in the output of the initial convolutional layer and the intial learning rate. All of the layers following the initial layer contain a number of channels which are computed relative to the number of output channels in the initial layer. This value was constrained to be a power of two such that the exponent of 2^n would be the hyperparameter which is controlled. The authors implemented a learning rate decay routine in which the learning rate is kept constant at the initial value and subsequently decays at an exponential rate after half of the prescribed epochs have been completed. The initial value of the learning rate was taken as the second hyperparameter and was drawn from a loguniform distribution between 1e-9 and 1e0.

3.4 Multi-Scale Network Architecture

The Multi-Scale Network (MS-Net) architecture used in this research was implemented using Pytorch. The implementation closely follows the architecture described in [25], the overview of which was presented in Figure 11. The model is made up of sub-models which operate on the different image resolutions. Each sub-model is constructed in the same manner as described in [25] and illustrated in Figure 13. All sub-models contain five blocks where the input is passed through a convolutional layer, instance normalization layer, and a CELU (Continuously Differentiable Exponential Linear Units) activation function layer. The output of the final layer is fed into the beginning of the next block, the convolutional layer. The final block does not contain an activation function layer, allowing arbitrarily negative values to be output which is not possible with the use of the a common activation function. The final layer of each sub-model is a convolutional layer with a single 1×1 kernel. This configuration condenses the number of channels into one channel [25]. This condensing allows for a more human-understandable output between each model as there is only one set of corrections/enhancement made to the the previous, upsampled intermediate prediction. Ding *et al.* found that using kernels of size 3×3 provided increases in evaluation speed and model accuracy on standard image dataset when compared to existing models with different architectures [46]. Therefore, the size of the convolutional kernels in the blocks of sub-models was maintained at 3×3. The CELU activation function [47] has the form:

$$\sigma_i = \max(0, z_i) + \min\left(0, \alpha \cdot \left(\exp\left(\frac{z_i}{\alpha}\right) - 1\right)\right) \tag{28}$$

where z_i and σ_i are the weighted sum and neuron layer output, respectively, as defined previously and α is a shape parameter which was assigned a constant value of $\alpha = 2$

for all instances in the MS-Net [25]. Except for the last convolutional layer, all convolutional layers of a sub-model contain the same number of output channels. The number of channels in a sub-model is double that of the sub-model with the next highest resolution. This configuration stipulates that the sub-model which operates on the largest resolution (i.e. the final sub-model) has the fewest number of internal channels and the number of channels increases toward the initial sub-model.

The number of sub-models included in the current MS-Net was dictated by the input image size. Recall that each sub-model operates on a different resolution of the input domain where each new resolution is constructed through a coarsening process which halves the resolution. If too many coarsening operations were performed on an input feature, then the final, lowest resolution could contain little or no useful information (e.g. if the image size was one or two pixels along both dimensions). To avoid such conditions, a lower bound of 10 pixels per dimension was placed on the coarsest resolution input feature. The full MS-Net model contains the maximum number of sub-models such that the smallest resolution does not fall below 10 pixels per dimension. The loss function used in training the implemented MS-Net varies minimally from that used by the original authors. The loss function used in this research is:

$$L = \frac{1}{N} \sum_{s} \sum_{i} L_s^i = \frac{1}{N} \sum_{s=0}^{S} \sum_{i=0}^{N} \frac{\langle (y_{i,s} - \hat{y}_{i,s})^2 \rangle}{\sigma_{y_s}^2} \tag{29}$$

where all variable definitions are consistent with Equation 27. Equation 29 is functionally identical to Equation 27 as the addition of the constant $\frac{1}{N}$ term does not modify the derivatives of the function with respect to the weights and biases of the network. The additional normalization by the number samples used to compute the loss enables comparisons of different models (as in the hyperparameter optimization) and different datasets (as in comparison between the training and validation sets).

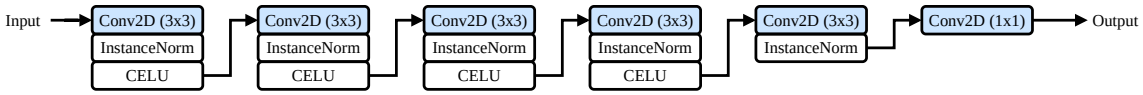


Figure 13. Architecture of the implemented MS-Net sub-model. Adapted from [25]

Without the normalization, the original loss function is heavily influenced by the sample size.

Two hyperparameters are allowed to vary in the construction of the implemented MS-Net, these are the number of internal channels of the final, full-resolution sub-model (which subsequently defines the number of channels in all other sub-models) and the learning rate of the optimizer. The number of channels in the final sub-model were constrained to be a power of two, 2^n, such that the exponent, n, was the independent variable with minimum and maximum bounds of zero and six, respectively. The learning rate was chosen from a lognormal distribution with the minimum and maximum defined as 1e-9 and 1e0, respectively. Investigating the effect of a learning rate decay schedule was beyond the scope of this work.

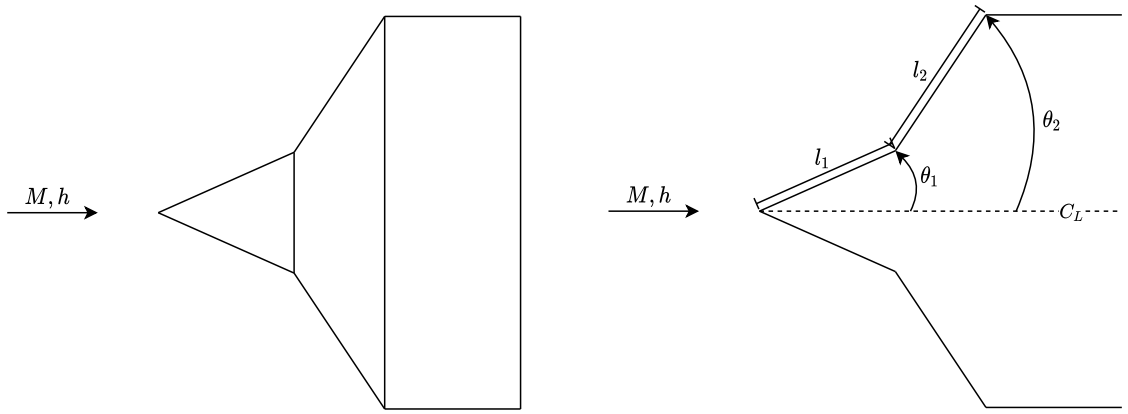
IV. Simulation and Data Collection

4.1 Double Cone

This research utilizes the double cone geometry as a vehicle for investigation of relevant high-speed flow fields. The double cone (presented in Figure 14a) is a geometry comprised of two axially-aligned cones with the tip of the second, downstream cone is embedded within the first. The tip of the first cone was set to have zero radius such that the leading edge of the vehicle is sharp. This vehicle was utilized in this research as the double cone (specifically the 25-55 double cone) has been the subject of several CFD studies [10], [48]–[52] as well as validation cases for the Kestrel solver package [53]–[55]. The double cone geometry has several characteristics which make it a desirable subject for such studies and CFD workshops. These characteristics include its ability to produce a variety of flow physics relevant to a 3D flowfield utilizing a 2D dimensional grid. This simplification is enabled by its axisymmetric nature which prevents asymmetric 3D effects that would be present for other 3D geometries (such as a finite wing) [10]. The double cone was parameterized using four variables to define the vehicle geometry as portrayed in Figure 14b. The side lengths of both cones were allowed to vary as well as the half-angles of each cone. The design space was selected such that this parameterization would encompass a wide range of vehicle configurations and generate different classes of flow field responses (further discussed in Section 4.2.1.2).

4.2 Design Space Sampling

The dataset used in this research was obtained through running CFD simulations using the methods and settings previously described.



(a) Geometry of the axi-symmetric double cone

(b) Schematic of the double cone with the geometric design variables labeled

Figure 14. Definition of the considered double cone (a) with the geometry as it would appear in an experiment (b) with the design variables labeled. Note that the body is axi-symmetric about the centerline, C_L, and the half-cone angles are defined from the same.

The design variables considered as well as their upper and lower bounds are presented in Table 1. Two variables were used to describe the freestream conditions - Mach number and geopotential altitude - and four variables were chosen to describe the double cone vehicle configuration. Two of these geometric variables describe each cone of the double cone. The upstream cone is denoted as Cone 1 and the downstream cone is denoted Cone 2. There was no constraint placed on the turning angle between the two cones, thus both compression and expansion corners exist in the design space.

The angle of attack of the vehicle with respect to the freestream was constrained to zero degrees for all samples as simulation is axisymmetric and a non-zero angle of attack would violate internal assumptions.

4.2.1 Design Matrix Generation.

Because one of the main goals of the present research was to learn pressure relationships in the flow field (namely shock propagation and interactions), a primary objective in generating the design matrix was to capture as many different shock strengths and propagation/interaction patterns as possible. Additionally, the effect

of training set size on the ML model performance was of interest. Finally, for computational expediency, all samples should be a part of the same design in order to reduce the number of CFD simulations required. Therefore, the design space had to be efficiently covered by not only the full design matrix, but also for select subsets of varying size. Sobol sequences provide this behavior.

Sobol sequences are designed to generate strings of numbers with low-discrepancies, i.e. the set of numbers is close to uniform distribution [56] thus satisfying the requirement for the design matrix to efficiently cover the design space. The most effective generation of Sobol sequences requires sample sizes which are powers of two [57] which satisfies the requirement for easy and consistent matrix augmentation. There exists two properties (A and A′) which measure the degree of coverage of a sample in multidimensional space by splitting the hypercube along each dimension and checking if points exists within these smaller hypercubes. Using appropriate "direction numbers" to initialize the underlying generator can satisfy both properties [58]. More uniform 2D projections can be obtained by either using direction numbers which are better suited for the sample dimensionality or by adding additional points to the existing design (by iteratively doubling the number of points such that the total number of samples remains a power of two) [57].

The Python package scipy was utilized to generate the Sobol sequence and corresponding design matrix. This code initializes sequences with the direction numbers found in [57]. Using this generator, $2^{10} = 1024$ points were sampled from the six-dimensional $[0, 1)^6$ unit hypercube space and subsequently scaled to the design space (Table 1). Figure 15 presents a pair plot of the final design where the marginal distribution along each single dimension is along the main diagonal and the 2D projections for each pair of dimensions populate the remainder of the lower triangular region. The 1D marginal distributions show that each dimension is uniformly distributed along

each single dimension. Most of the 2D projections do not exhibit any discernible grouping, however there are some noticeable gaps in the projections for some pairs of axes. These gaps are isolated and not much of a concern. The discrepancy of this sample is approximately 6.20×10^{-5}, one order of magnitude smaller than the discrepancy of a comparable Latin Hypercube Sample generated for the same design space.

4.2.1.1 Undesirable Latent Distributions.

When sampling a design space, there is a possibility that a combination of the design variables form implicit linked variables which may have a significant effect on the surrogate model performance unbeknownst to the researcher [59]. Not only is the distribution of the design variables important, but the distribution of the interactions of these design variables could introduce biases in the dataset. Two linked variables/interactions were identified as important in the design variable definitions - the turning angle between the cones and the freestream dynamic pressure.

Turning angle was defined as the difference between the cone angles of the upstream and downstream cone where a positive value indicating the downstream cone deflects flow towards the freestream (compression corner) and a negative value indicating deflection away from the freestream (expansion corner). If the turning angle is zero, then the configuration is identical to a sharp-nosed single cone with half-angle of the equal upstream and downstream cone angles. The ideal shape of the turning angle distribution is to have a peak at the most typical value expected in the operating environment. If the most typical operating condition is not properly represented, a ML model trained on the data may exhibit poor performance at and around this operating condition [22].

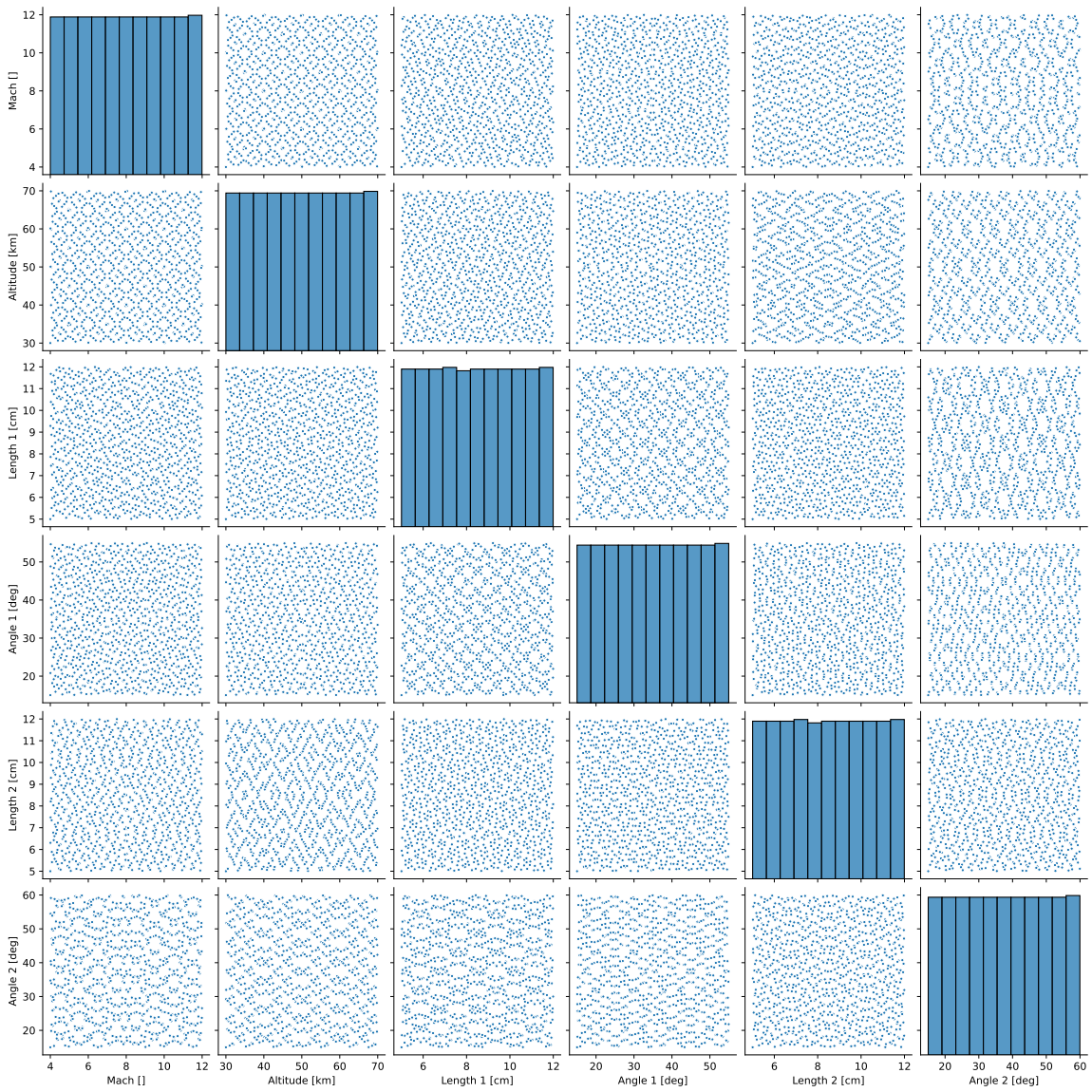


Figure 15. Pair plot of the 1D marginal distributions (along main diagonal) and 2D projections of design space samples generated using a Sobol sequence

Typically, components along a vehicle cause turning into the flow (compression), only some things cause large deflections (like possibly wings and engines). we wanted examples of expansion (like one would find on a blunt-nosed body) and examples of compression (like the interface between a wing and the fuselage)

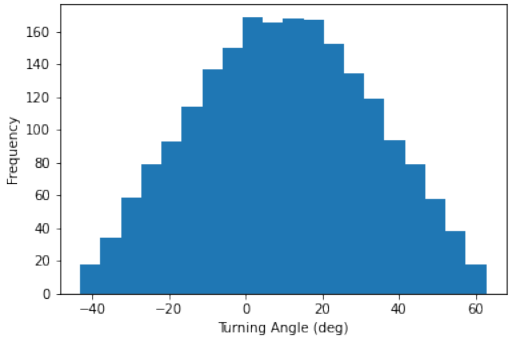
Dynamic pressure (Equation (30)) cannot be sampled directly as by itself as it does not uniquely encode a flight condition. The dynamic pressure of a flight condition is a function of the Mach number, M, and the altitude, h:

$$\begin{aligned} q &= \frac{1}{2}\rho v^2 \\ &= \frac{1}{2}\rho(h) \cdot (M \cdot a)^2 \\ &= \frac{1}{2}\rho(h) \cdot M^2 \cdot \gamma RT(h) \end{aligned} \tag{30}$$

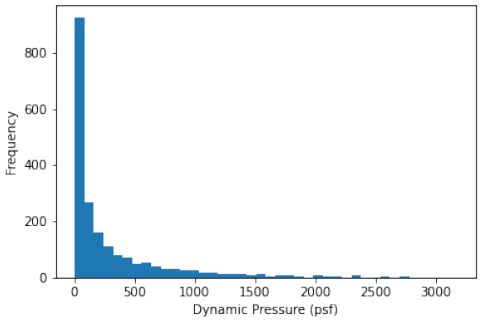
In the raw distribution of dynamic pressure, there is an exponential decay following the exponential decay of density through the atmosphere [60]. If training the ML models on this data, the greedy optimization function would likely exploit the distribution, giving good predictions where data is densely populated and poor predictions at the fringe values. Thus, the dynamic pressure was transformed using a logarithm to make the distribution more uniform (Figure 16c). In this formulation, there are no areas of interest which lack dense coverage in either the freestream dynamic pressure or the turning angle.

4.2.1.2 Design Space Bounds.

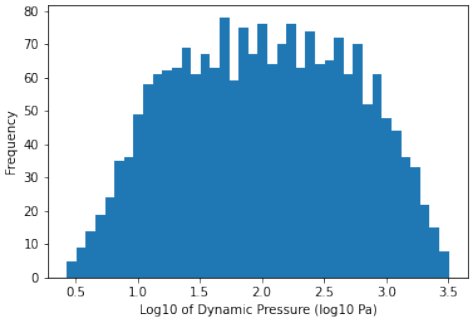
The choices of upper and lower design variable bounds were driven by a combination of previous vehicle configurations and operating conditions. This double cone model is intended to provide examples of the spacial relations relating to shock waves which occur when fluid is deflected around a hypersonic system. The machine learning models are trained with the intention to approximate flow fields across a wide range



(a) Turning angle distribution of the sampled design



(b) Freestream dynamic pressure distribution of the sampled design



(c) Freestream dynamic pressure distribution of the sampled design with a logarithmic transformation

Figure 16. Identified latent distributions in the design space: (a) turning angle between the cones and (b)-(c) freestream dynamic pressure

of flight conditions with a mix of oblique and bow shocks. As such, the parameterized double cone model is not intended to accurately recreate any specific class of vehicle, instead intending to cover as wide range of possible shock wave configurations which could appear around a hypersonic system. The minimum bound of the flight Mach number was chosen to be four. While Mach five is a generally accepted threshold for the beginning of the hypersonic flight regime [2], recent hypersonic vehicles still operate within the range of Mach numbers from four to six [9]. The the upper bound for Mach number was set as 12 which is in line with double cone test case which was performed. The flight altitude minimum was selected as 30 kilometers as this is the approximate regime in which a hypersonic cruise missile would operate [9]. The altitude defined by the design variable is the geometric altitude, or the actual altitude above Mean Sea Level (MSL). Kestrel utilizes the geopotential altitude to calculate freestream conditions using the framework of the 1976 Standard Atmosphere for a Standard Day [60]. This model is only valid for geopotential altitudes up to 84.852 km or 86 km of geometric altitude. The other extreme of hypersonic weapons includes Hypersonic Glide Vehicles (HGV) (or boost-glide vehicles) which operate near the edge of the atmosphere in potentially rarefied flows [9]. Rarefied flows introduce additional numerical considerations such as non-equilibrium effects and large mean free paths which break the assumptions of the air model used in the simulations. Consideration of real gas effects is beyond the scope of this research, thus in order to avoid flight conditions which may result in rarefied flows, the maximum Knudsen number of the flow was specified to be 0.01 such that the flow could be considered to be continuum flow. The applicability of the governing equations used to simulate the fluid flow within the computational domain begin to break down for conditions with

larger Knudsen numbers [61]. The Knudsen number can rearranged into the form:

$$Kn = \frac{M}{R_e}\sqrt{\frac{\gamma\pi}{2}} \tag{31}$$

where M is the Mach number of the flow, γ is the ratio of specific heats (held constant at $\gamma = 1.4$ in the diatomic Perfect Gas model), and R_e is the Reynolds number. The Reynolds number can be calculated as a function of Mach number and atmospheric properties at the specified altitude. A series of numerical trials show that a maximum altitude of 70 km ensures that all design points have a Knudsen number of less than the cutoff for continuum flow. Note that reference length for the Reynolds number was defined as the axial length of the simulated double cone model or approximately 29 cm. The dimensions of the computational domain were designed to fit the 25-55 double cone and closely followed the domain described in [10] with the horizontal no-slip wall downstream of the cones extended by 5 cm in order to accommodate longer double cone configurations while maintaining some distance from the end of the downstream cone and the numerical outlet boundary condition.

The extrema for the cone shape parameters (the length and angle of each cone) were selected with the primary intention of generating a diverse range of shock waves and shock wave interaction and impingement. Desired relations included attached oblique shock waves without downstream interactions and bow shocks as these both have their uses in the operation of a vehicle depending on the mission. The HIFiRE missile was used for hypersonic testing and the nose of that vehicle was elliptical with a minor axis radius of approximately 3 cm [62]. Therefore, the lower bounds of the four geometric design variables were chosen such that the minimum radius of the double cone was approximately the same, resulting in minimum lengths of 6 cm for each cone. The minimum angle of both cones were selected to be 15 degrees. This allows for the effect of small deflections to also be present in the dataset (possibly

like probes and inlet cone tips). The maximum angle of the first cone was selected to be 55 degrees. At this angle, the leading edge shock can be detached and form a bow shock for relatively low-hypersonic speeds. The maximum angle for the downstream cone was set to 60 degrees. This configuration, on average, will have a positive turning angle, causing any leading edge shock to have some interaction with the geometry or corresponding growing boundary layer. The large difference between the minimum angle of the first cone and the maximum of the second cone will allow for configurations with large circulation regions, causing changes in the shock shape. Because the angle ranges overlap, the configuration can degenerate into a single cone. No preference was given to either cone in terms of length as a good mix of possible interactions with geometry was desired.

Table 1. Design variable limits

Lower Bound	Design Variable	Upper Bound
4	Mach [·]	12
30	Altitude [km]	70
5	Cone 1 Length [cm]	12
15	Cone 1 Angle [deg]	55
5	Cone 2 Length [cm]	12
15	Cone 2 Angle [deg]	60

4.2.2 Training, Validation, and Test Splits.

The dataset consists of 1024 samples generated using a sobol sequence. A random selection of 10% of the total dataset was withheld as test data. This data would not be used for training any of the models in any capacity. The remaining 90% of samples were the training data. From the training data, another random selection of 20% were assigned to the validation set and the rest of the data was used as the training set. The validation set has a similar role as the test data. Model candidates generated using different hyperparameter settings were trained using the training set and subsequently

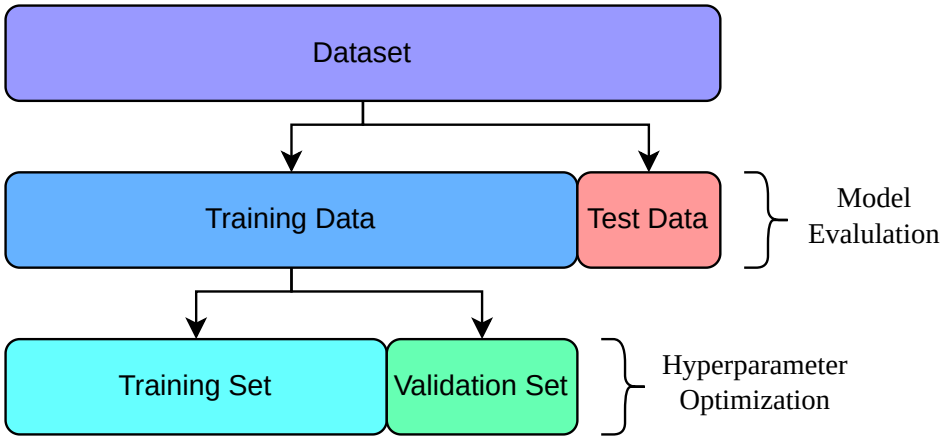


Figure 17. The full dataset is split into training and testing data. The ML model is trained using only the training data and subsequently evaluated using the test data. If the model contains hyperparameters, the training data can be split again and used for hyperparameter optimization without using the reserved test data.

their performances were evaluated on the validation set. The hyperparameter settings with the best performance on the validation set were selected as the the optimal hyperparameter values. After the determination of the hyperparameters, the final model was trained on the sum of the training and validation sets (the full training data) and performance evaluated on the withheld test data. Figure 17 illustrates the dataset splitting utilized.

4.3 Solver Configuration

This section details the settings utilized in the cell-centered, finite-volume CFD solver, Kestrel [63], for the computation of the flowfield solutions. As discussed previously, the full design space spans a large range of freestream conditions, and the solver settings used were held constant for all cases. Varying the solver settings within the design space would introduce additional, local biases as well as possible discontinuous changes between computed responses which lie on either side of a setting change boundary.

It was expected to have boundary layers which interact with the oblique shock generated by the leading edge in some double cone configurations. This interaction effect is studied by Knight and Youssefi [10] and has the potential to significantly alter the resulting flow field solution from the equivalent Euler solution. For this reason, a set of governing equations which considers the effects of viscosity was deemed necessary. The samples in the design space have Reynold's numbers which range from 1.8×10^3 to 1.2×10^6 when the axial length of the simulated geometry (approximately 28.6 cm) is utilized as the reference length. Figure 18 presents the distribution of Reynolds number from the design sample space. If the flat plate critical Reynolds number of 500,000 is used to predict if a transition to turbulent flow will occur, then approximately 8% of all samples are predicted to contain turbulent flow. Utilizing a laminar set of governing equations would introduce excess restrictions in the flow solver by disallowing the transfer of energy into turbulent structures. Therefore, the Reynolds-Averaged Navier-Stokes (RANS) governing equation set is used with the Spalart-Allmaras (SA) turbulence model. Kestrel specifically implements the negative SA model. Using the Rotation Correction (RC) or the Menter two-equation turbulence model were found to have no substantial effect of the final calculated solution. The Perfect Gas air model was utilized for all simulations as chemical reactions and non-equilibrium effects were beyond the scope of this research. Ramping of the Courant-Friedrichs-Lewy (CFL) number was utilized in the computation of the steady-state flow field responses. The CFL number was linearly ramped from 0.1 to 100 over the 300 start-up iterations. Reducing the maximum CFL number to did not have a significant effect on the final value of the numerical residuals.

The default early stopping stopping criterion was used to terminate simulations when the solution has converged with respect to the number of iterations. This criterion is that the drag coefficient does not change by more than 1e-5 over the final

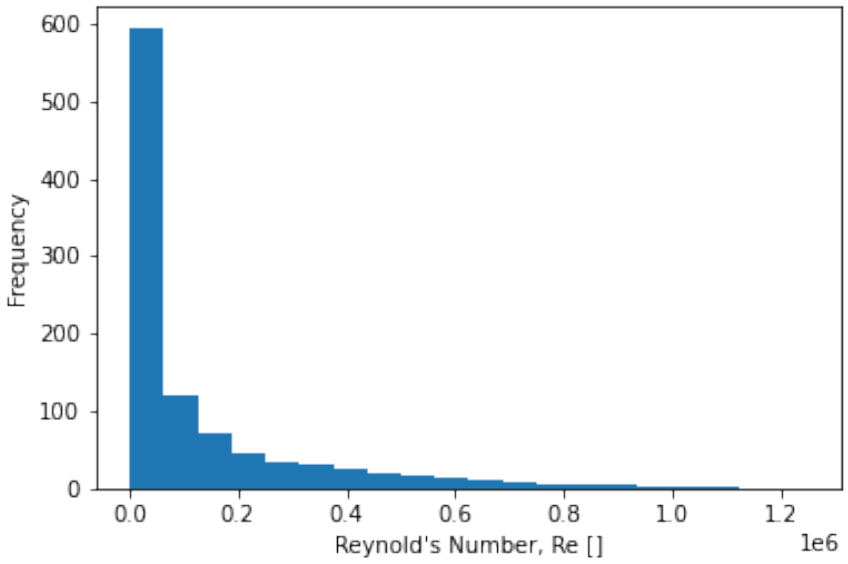


Figure 18. Distribution of Reynold's number based on vehicle total length

100 iterations. This criterion was used as the drag on the double cone is a function of components, the viscous drag and the pressure drag. For the drag force, and thus the drag coefficient, to converge, both of the drag sources must have stabilized. The pressure drag is a function of the surface pressure distribution and the viscous drag is effected by the viscosity in the boundary layer which is strongly related to the temperature distribution. The local viscosity is also enhanced by the turbulence model which additively contributes a turbulent viscosity term to the standard viscosity.

Three boundary conditions were applied to the computational domain (Figure 20). The far field boundary conditions are automatically handled and enforced by Kestrel. There is no need to distinguish between inflow and outflow conditions in the set-up of the simulation. The connectors which represent the surface of a double cone configuration were assigned a no-slip wall condition. Additionally, these surfaces were considered to be isothermal at a constant temperature of 300 K regardless of the freestream conditions. This condition is the same as is considered in the reference study [10] and was not altered as the other wall conditions were not evaluated against

experimental data. The Kestrel solver utilizes a cell-centered, finite-volume approach to solving the RANS governing equations and enforces the boundary conditions at the wall. Because the boundary conditions are not enforced at the cell centers, two competing boundary conditions may be weakly enforced on the same cell. The notable location where this occurred was at the intersection of the leading edge and symmetry plane. The 3D nature of the double cone was accounted for by enabling the axi-symmetric correction in the body definition.

4.4 Verification Case

Before simulating all of the generated samples, a test case was desired to ensure that previous results could be reproduced in a reliable manner using both the gridding method and the CFD solver. The 25-55 double cone geometry with a sharp leading edge was selected for this test case. This geometry was selected because it has been the subject of several CFD studies [10], [48]–[52] as well as validation cases for the Kestrel solver package [53]–[55]. These studies and CFD workshops have included the double cone geometry because it is able to produce a variety of flow physics relevant to a 3D flowfield utilizing a 2D dimensional grid as its axisymmetric nature prevents asymmetric 3D effects which would be present for other 3D geometries (such as a finite wing) [10]. The article by Youssefi and Knight [10] was selected to compare the test case against as they presented both experimental results and independent Perfect Gas simulation results generated using the commercial solvers General Aerodynamic Simulation Program (GASP) and GASPex. The authors utilized the experimental data produced by MacLean *et al.* [64] at the LENS XX expansion tunnel at the Calspan University of Buffalo Research Center (CUBRC) in 2014. The same experimental data has been used as reference for this research.

The case used for this solver verification was Run 1 from [10]. These conditions are reproduced in Table 2 along with the associated double cone geometric parameters. Figure 19 presents the simulation results for the tested flight conditions (in solid black) as well as the corresponding experimental results (in black with error bars in red). The simulation results obtained by the authors agree with the experimental data relatively well, especially in predicting the location of the recirculation zone (at approximately 7cm) and the location of maximum pressure (at approximately 10cm). It should be noted that the locations of the pressure ports on the test article may not have been adequate to capture the true maximum pressure as this may have occurred between sensors. The simulated surface pressure downstream of the shock impingement exhibits an oscillatory behavior as a train of compression and expansion waves form after the reattachment point. The surface pressure simulated and measured appear to have divergent behaviors after the reattachment point (from approximately 11 to 13 cm), however it should be noted that the spacial density of pressure sensors is not sufficient to capture the oscillations observed in the simulated data. At most locations, the simulated surface pressure falls within the error bounds of the experimental data. Because the authors' simulation results agree well with the presented experimental data, their simulation data is also used for comparison with the flow field results in this research (which are generated with Kestrel).

Table 2. Verification case flow field and geometric parameters

Flow Field Parameters		Geometric Parameters	
Mach [·]	12.2	Cone 1 Length [cm]	10.16
Density [g/m^3]	0.499	Cone 1 Angle [deg]	25
Static Temperature [K]	175	Cone 2 Length [cm]	10.74
		Cone 2 Angle [deg]	55

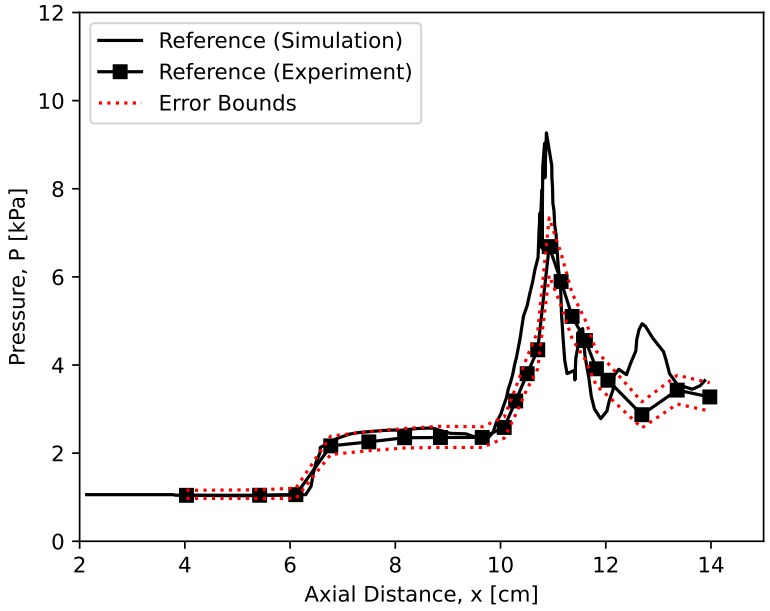


Figure 19. Surface pressure data from simulation results (black) and wind tunnel experiment (black) with error bounds on experimental measurements (red). Adapted from [10]

4.5 Computational Mesh

Kestrel is a finite-volume solver and therefore requires the physical domain of interest to be discretized into finite volumes or cells. This section defines this physical domain and its relevant boundary conditions. Due to the nature of the current research, each sample has a unique corresponding body definition and thus requires a unique computational mesh for each sample. Due to the large number of samples which were generated, manual creation of each mesh would have been prohibitively expensive. While mesh perturbation could have reduced the number of total grids required, its utilization is beyond the scope of this research. To accommodate the mesh generation, an algorithm was developed. The experimentation related to and final decisions regarding the mesh type, grid construction, and automation are explored in this section.

4.5.1 Domain Definition.

The relavent physical domain of the double cone is relatively small due to the hypersonic nature of the problem. As the fluid flow is supersonic, no information can be propogated upstream. This enables the inflow and outflow boundary conditions to be placed much closer to the actual body without concerns of sonic phenomenon introducing reflecting pressure waves into the solution. The computational domain for a sample double cone configuration is presented in Figure 20. The domain consists of seven connectors with three used to prescribe the surface of the double cone (labeled W_i). Three connectors define the outer far field (labeled FF_i) and the final connector represents a symmetry plane (labeled S). Note that the axi-symmetric correction in Kestrel assumes the body is a body of rotation about the x-axis. There are four points which are fixed in this definition of the computational domain. The origin (denoted by the x and y axes), the upstream corner formed by the symmetry plane and FF_1, and the two corners at the intersection of the far field boundary conditions. This computational domain is to be discretized, resulting in a computational mesh which the CFD solver can utilize to calculate flow solutions.

4.5.2 Mesh Type Selection.

There are two main class of mesh which can be used to discretize the computational domain: structured and unstructured. Due to the formulation of the governing equations for the finite volume discretization, gradients are best captured when the local cell faces are closely aligned with the direction of the gradient. The shock waves and boundary layers which form in supersonic and hypersonic flows are examples of flow features which cause large, local gradients in the flow variables. In the general case, structured grids are preferred for flow regions where large gradients are present (such as in the boundary layer or near shock waves. Generation of structured com-

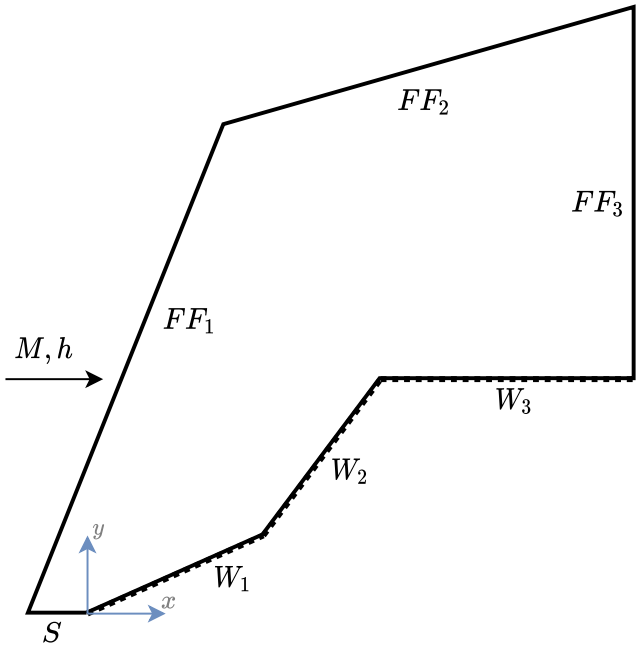


Figure 20. Computational domain setup for all samples. The segments labeled with *W* are no-slip walls, the section labeled with *S* is a symmetry plane, and the sections labeled with *FF* are defined as farfields. Note that Kestrel handles the inflow inlet and outlet conditions automatically for farfield boundary conditions.

putational grids for the prescribed domain topology were attempted; however, it was found to be intractable. The attempted structured grids suffered from cells of negative volume and cells with very large aspect ratios. The former is aphysical and would cause numerical issues in the finite volume solver. The latter allowed the CFD simulations to converge, however it was found that the calculated flow solutions were aphysical. Figure 21 presents a representative flow solution from a structured grid. The shock wave follows the face alignments in the grid upstream of the compression corner, suppressing the formation of the separation region and coupled shock deflection. Additionally, the shock exhibits a kink (at an approximate axial location of 10 cm) as it propagates downstream of the shock interaction Attempts to generate higher quality grids to assuage these issues included performing smoothing operations, modifying the domain topology, and utilizing varied clustering techniques of the seed points along the connectors. None of these attempts were successful in

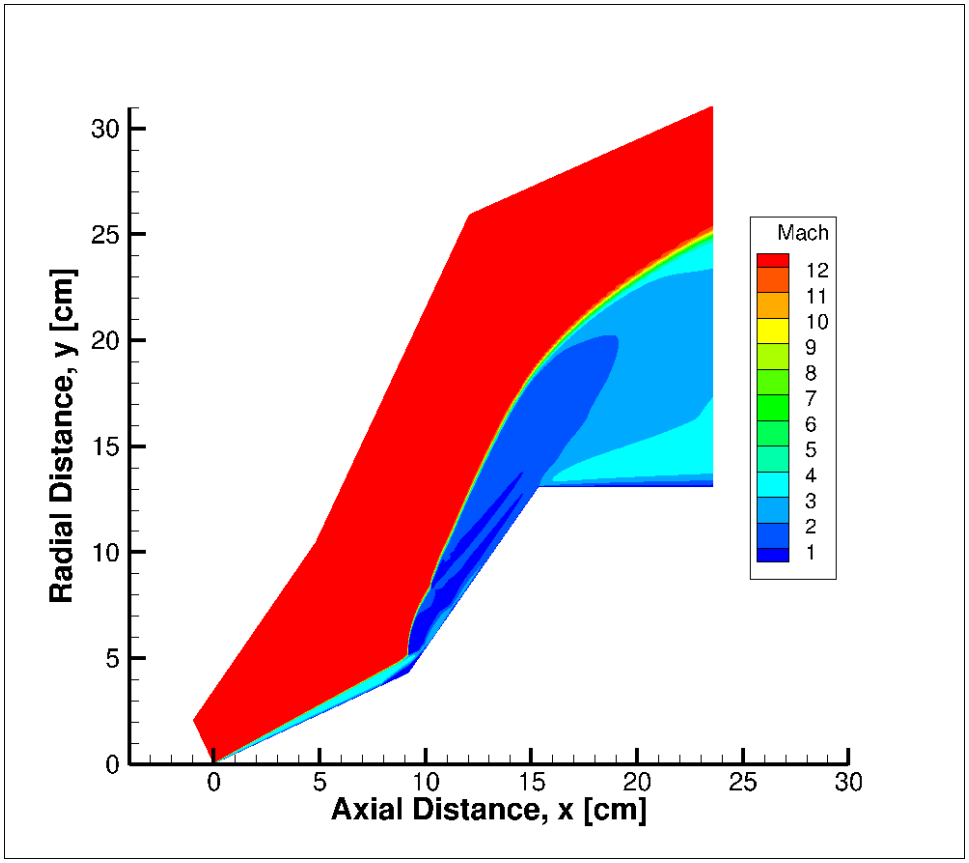


Figure 21. Representative solution on structured grid. The shock propagating from the leading edge follows the grid lines, suppressing the separation zone. Additionally, there is an aphysical kinking present in the shock at approximately 10 cm.

yielding a structured grid which could produce physically consistent results or results which matched the reference results. In addition to unsuccessful attempts to create a high-quality grid for the 25-55 double cone test configuration, the configuration of the computational domain is not constant throughout the design space of interest. As high-quality grids would be desirable for all, arbitrary double cone configurations, the structured grid generation algorithm would be required to generalize well to the full design space. Because of the described issues encountered in generating a structured grid for a single design point and the fact that the developed procedure would need to generalize without loss in grid quality, it was decided that utilizing a structured grid for this study was intractable.

Unstructured computational meshes were investigated subsequent to the abandonment of the structured grid approach. Unstructured meshes are much more flexible than structured grids and are generally more simple to successfully fit to complex geometry as their constituent cells are not constrained to have a rigid, quadrilateral-based (in the 2D case) structure and can also contain triangular cells. These triangular cells enable unstructured tiling of the computational domain at the cost of solution accuracy. For cell-centered finite volume solvers (such as Kestrel), the velocity or momentum vector in an arbitrary cell is guaranteed to not align perfectly (either parallel or perpendicular) to at least one cell face. In contrast, a rectangular cell has two sets of sides which are parallel to each other, thus the cell's momentum vector can be perfectly aligned with all sides at the same time. In the calculation of convective fluxes between neighboring cells, a perfectly aligned momentum vector in a rectangular cell will only have a flux through the two faces which are perpendicular to the vector. This alignment does not generate any diffusion of the momentum in the cell. In a triangular cell, because this alignment is not possible with all sides of the cell, therefore there is a tendency for diffusion of the flow to occur. This artificial diffusion can lower the accuracy of the simulation results. The trade for introducing artificial diffusion allows an unstructured mesh to have no preferred flow direction in which its calculations are most accurate. This property of producing results which are insensitive to the flow direction is attractive for this research as the true direction of the shock waves is not known for an arbitrary case in the design space. Additionally, the use of unstructured triangles allows for the aspect ratios of cells to be maintained at reasonable values near unity. The unstructured mesh approach allowed the generation of well behaved meshes after some structural issues related to the boundary conditions were addressed (see Section 4.5.3). This mesh generation procedure is very robust, producing well-behaved meshes throughout the design space. The util-

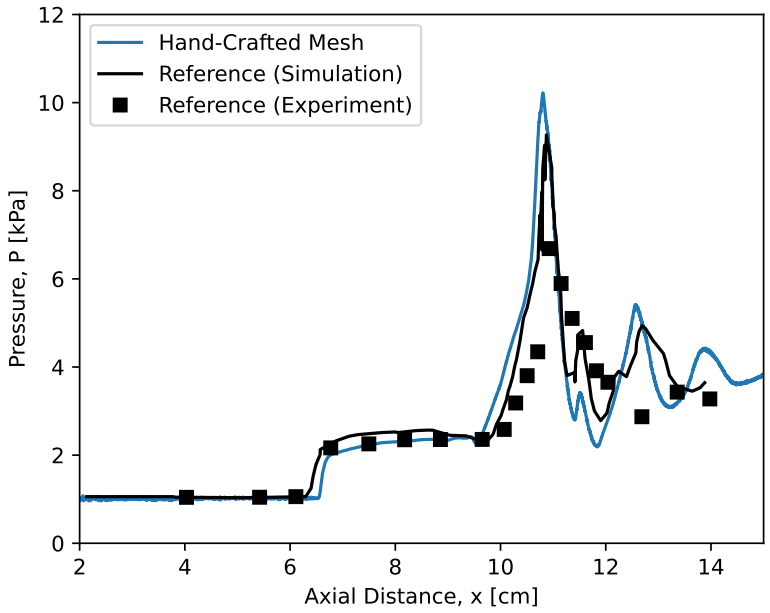


Figure 22. Surface pressure distribution result generated using the hand-crafted unstructured mesh compared to the reference data from [10]

ity of the procedure is further demonstrated by the ability of the generated grid to closely match the reference experimental and simulation results from [10]. Figure 22 presents the computational and experimental surface pressure results from [10]. The black squares represent the experimental data with error bars and the independent simulation from the reference is shown in the solid black line. Note that the presented results were generated using a hand-crafted mesh and not the process described in Section 4.5.4.

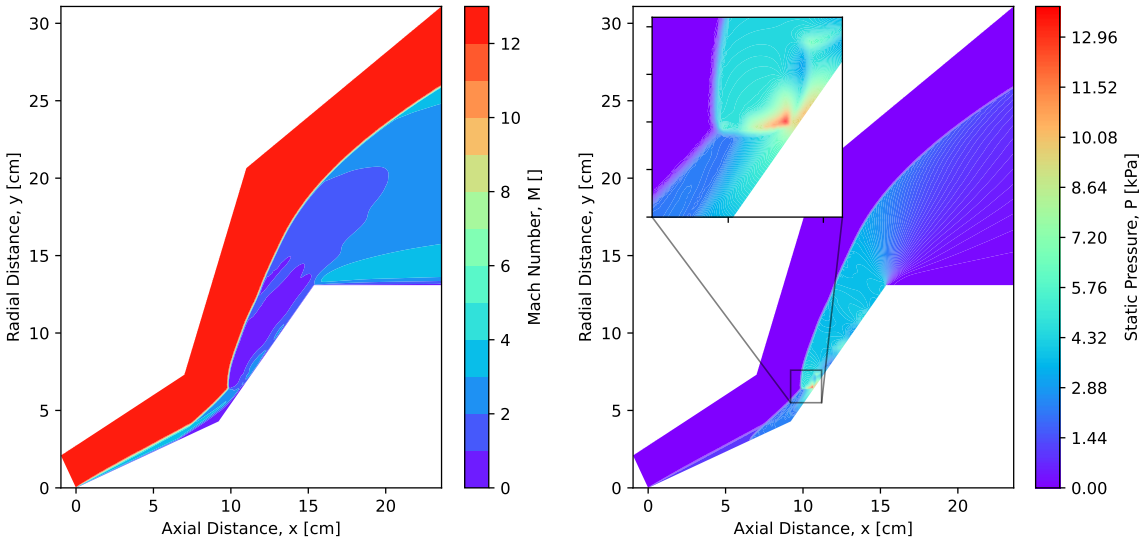
One additional computational mesh type exists - the hybrid mesh. This technique is a hybrid of the unstructured and structured meshes described previously. The mesh near viscous walls are generated using quadrilateral cells which allow for the computational benefits of structured grids such as good gradient resolution in the boundary layer. Beyond the boundary layer, the mesh then transitions into an unstructured grid so the benefits of the unstructured tiling and insensitivity to flow direction are realized in the freestream. In theory, this type of mesh should produce comparable results in the freestream and more accurate results in the boundary layer compared

to the fully triangular unstructured mesh. The hybrid mesh was not utilized in this research due to the measures that had to be taken at the leading edge of the double cone and the fact that well-behaved meshes could not be ensured for all points in the design space.

In this research, it was found that producing a robust structured or hypbrid meshing procedure was not feasible for the arbitrary geometries present in the experiment design space. Additionally, the unstructured grid yielded results which were a reasonable match to the reference simulation and experimental wind tunnel data. Thus, the fully triangular unstructured mesh approach was selected to be used for the rest of this research.

4.5.3 Boundary Condition Considerations.

During experimentation, it was found that the behavior of the shock wave at the leading edge of the double cone geometry had a profound impact on the final flow field solution. The initial computational domain used to generate the solution at the walls shown in Figure 22 was hand-crafted and not congruent with the domain shown in Figure 20. This domain and a close approximation of the flow field Mach number distribution presented in [10] are presented in Figure 23 . The most notable difference between the two domains is the lack of a symmetry plane upstream of the double cone. The symmetry plane was not included in this version of the domain as it was not intended as a template for the rest of the simulation cases. This domain and corresponding mesh were intended for use only in the validation case. The addition of the symmetry plane is necessary for the full study as bow shocks are expected to form, yet the addition of the symmetry plane also dramatically changed the behavior of the flow near the leading edge and consequently the final computed flow solution.



(a) Mach number distribution

(b) Static pressure distribution with inset of shock impingement

Figure 23. Flow field (a) Mach number and (b) static pressure response calculated for the reference condition. These results agree visually with those presented in [10].

The initial construction procedure for generating a mesh for an arbitrary double cone configuration was built to be straightforward and robust. The entire computational mesh for a double cone configuration would be controlled using a single value - the spacing along the walls. The no-slip walls and symmetry plane were built with seed points equally spaced along the connectors with the same absolute spacing distance applied to all four connectors regardless of the configuration or lengths of the connectors. The initial spacing on both the first inlet and the outlet (FF_1 and FF_3, respectively) were also set to this equal spacing value. From this value, the spacing of points along these two far field connectors increased exponentially such that the final spacing was similar for both. The mean of these two final spacings was then applied as an equal spacing on the top far field connector FF_2. The effect of this generation procedure was to cluster cells near the double cone body and in the upstream, near the leading edge in order to maintain relatively high resolution near where shocks were expected to form.

Constructing a mesh for the 25-55 double cone validation case configuration utilizing this method and simulating it produced vastly different results than the previous hand-crafted mesh. The resulting surface values are presented in Figure 24 alongside the surface values from the previous unstructured mesh as well as the experimental data. The results on the mesh generated using the procedure previously described do not match well with the either the previous results or experiments. After investigation, it was found that the dramatic change in the surface pressure distribution was caused by the shock wave generated at the leading edge impinging on the downstream cone at a shallower angle. This pushes the impingement location downstream slightly and causes the resulting shock to be weaker which generates lower pressures than expected. The root cause of this discrepancy was traced to the initial generation of the shock at the leading edge. Figure 25 presents the leading edge of both the solution on the hand-crafted mesh (top) and the solution on the procedurally generated mesh (bottom). The top solution behaves very similarly to the expected attached oblique shock. The initial boundary layer height at the leading edge of the double cone is small. The bottom solution exhibits a bow shock with a small standoff distance; this is not the theoretically expected behavior for this flow. This bow shock is a strong shock where it is normal to the incoming flow, causing a sharper velocity decrease and pressure rise near the symmetry plane intersection with the leading edge wall. These results serve to bolster the initial boundary layer height on the first cone and deflect the shock to a higher angle (as measured counter-clockwise from the x-axis). All of these coupled effects culminate in a surface pressure distribution which is much different from the expected result.

In pursuit of the cause of the modified behavior near the double cone leading edge, multiple possible causes were explored. The first possible cause explored was the overall grid resolution. As discussed previously, the resolution of the entire grid

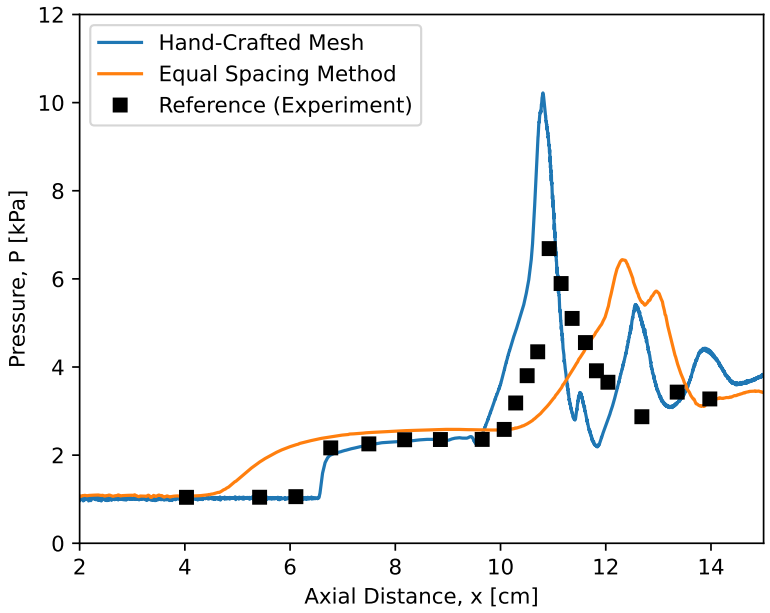


Figure 24. Surface pressure distribution result generated using equal spacing of seed points along the walls and symmetry plane

is determined by the selected equal spacing for the walls. The grid resolution was explored by successively halving the equal wall spacing until a mesh size was reached such that the computational resources required to generate it exceeded those available. The hand-crafted mesh utilized hyperbolic tangent spacing [65] in order to cluster points near the leading edge. The spacing at the leading edge was set to 5×10^{-6} meters. In contrast, the highest resolution procedurally-generated mesh had an equal wall spacing of approximately 1.4×10^{-5} meters. It was found that the standoff distance of the shock was reduced slightly by increasing the resolution, however after a point, the standoff distance remained constant and the shock merely become more resolved. The next possible cause explored was the turbulence model. The Menter two-equation turbulence model [66] was also available along with the Shear Stress Transport (SST) and Rotation Correction. All combinations of the turbulence model settings were simulated, to include the addition and suppression of wall functions. It was found that none of these configurations had any significant effect on the results for the 25-55 double cone.

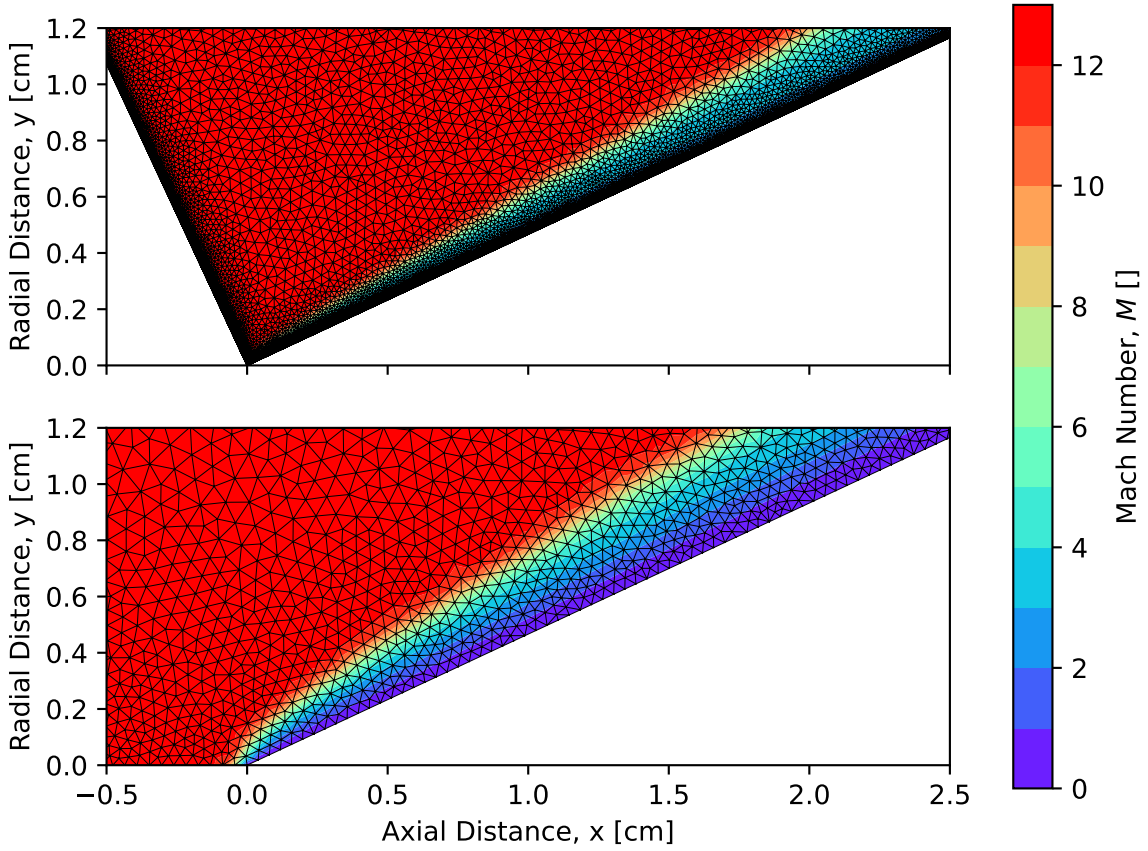


Figure 25. Flow field Mach number response near the leading edge for both the hand crafted mesh (top) and the more general, procedurally generated mesh with equal cell spacing (bottom). The shock wave behavior changes dramatically with the addition of the symmetry plane in this configuration.

The next cause investigated was the angle between the inlet condition and the double cone wall. The hand-crafted mesh defines the freestream boundary condition to be orthogonal to the double cone wall and intersects the body at the leading edge. To investigate the effect of this angle, the symmetry plane was converted into a fourth far field boundary condition and its angle was varied between horizontal and 15 degrees from the wall. These configurations with various angles were generated at both the original mesh resolution and the highest resolution previously achieved. It was found that this also had no effect on the surface pressure distribution.

All previous attempts to bring the predictions generated by the procedurally generated mesh in line with the reference data utilized an equal spacing along the walls and symmetry plane. This equal spacing allowed for consistency and uniformity in meshes of varied double cone configurations. The final attempt to generate better meshes was to abandon this equal spacing and adopt the hyperbolic tangent spacing utilized in the hand-crafted mesh. The hyperbolic tangent spacing allows for the prescription of different cell spacings at either end of a connector. Thus, the grid can be non-uniformly refined and additional cells clustered where required, such as near the leading edge of the double cone. The main disadvantage of this method is the smooth transition from the prescribed initial and final spacing. The first cone was the only connector to utilize the new spacing. The spacing near the leading edge was set to be the same 5×10^{-6} meters as in the hand-crafted mesh while the spacing near the interface with the second cone was prescribed to be the equal wall spacing. This ensured a smooth transition between cell sizes on the first and second cones. The average cell spacing on the first cone was set to be the equal wall spacing in order to maintain robustness of the meshing procedure. The same procedure was applied to the symmetry plane to match the spacing immediately on either side of the leading edge. Prescribing the initial, final, and average spacing forced the hyperbolic tangent

spacing function to generate larger cells near the center of the first cone to balance out the effect of the small cells near the leading edge. The largest cells along the first cone generated by the new spacing function could be up to an order of magnitude larger than those generated using the equal spacing for a given average spacing. This variation in cell sizes also contributed to increases in the $y+$ values of the cells along the wall, peaking near ten for the 25-55 verification case. While Kestrel will automatically utilize wall functions, this variation in first-cell spacing was undesirable. Figure 26 presents simulation results from this non-uniform spacing attempt. The additional resolution near the leading edge introduced by the hyperbolic tangent spacing did allow the shock to form in a manner more consistent with the expected behavior. However, the separation zone did not form as expected and thus did not deflect the shock in the same manner as in the hand-crafted mesh. The result of this under-deflection is that the shock wave impinges on the double cone ahead of the expected location and at a steeper angle. Thus the spike in surface pressure is larger and to the left of the reference data. Additional, slight modifications to this spacing setup were attempted with no significant change in the resulting surface pressure distribution. Utilizing adaptive mesh refinement measures were beyond the scope of this research.

The procedural mesh generation method with the hyperbolic tangent spacing along the first cone with an average cell spacing equal to that of the equally spaced connectors was ultimately utilized to generate all of the training data (Figure 26). When compared to the construction procedure utilizing the equal spacing (Figure 27), the results from the hyperbolic tangent spacing produce results which are more relevant to this research. The strong pressure spike which occurs over a small local area is present when utilizing the latter method. Additionally, the shocks which form downstream are also predicted at a similar spacial scale with similar pressure mag-

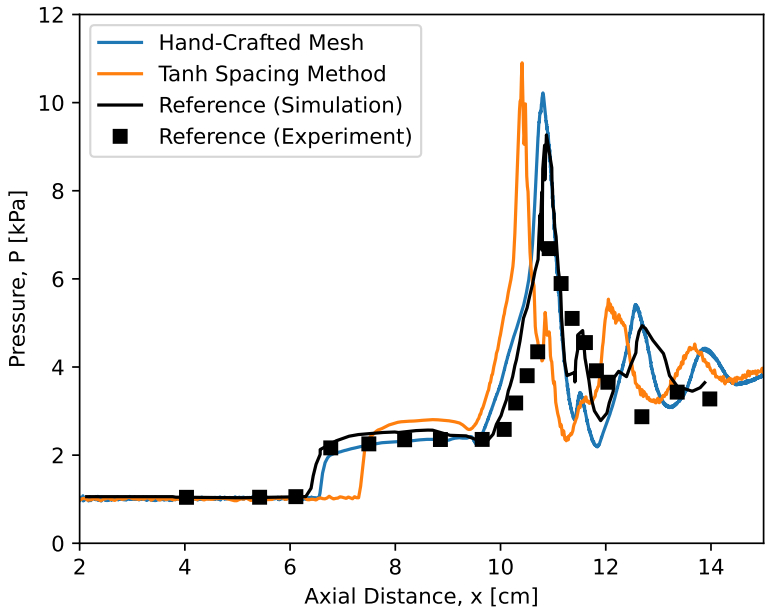


Figure 26. Surface pressure distribution result generated using the hyperbolic tangent spacing of seed points along the walls and symmetry plane

nitudes as in the reference simulation. The ability for the Machine Learning (ML) models to capture these nuances in the flow is part of the scope of this research. Therefore, the training data must include the flow structures which the models are intended to reconstruct. Despite both procedures generating solutions which do not accurately match the reference values, the hyperbolic tangent provides the best approximation while enabling a robust meshing procedure which works on all double cone configurations in the dataset. The specific procedure which resulted from this initial case is described in Section 4.5.4.

4.5.4 Procedural Generation.

As discussed previously, the large number of unique double cone configurations considered in this research necessitated an automated procedure for generating the corresponding computational meshes. Some of the relevant decisions for this procedure have also been discussed previously. This section focuses on the implementation of the final mesh generation procedure.

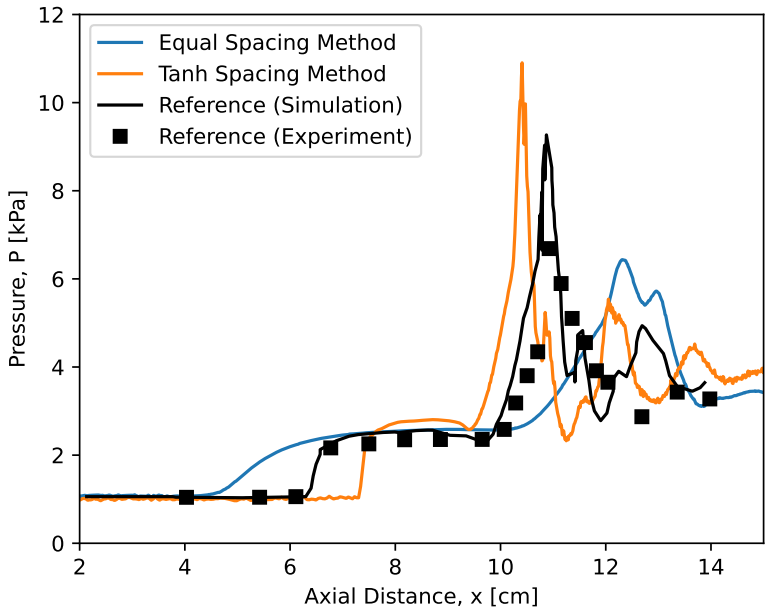


Figure 27. Surface pressure distribution comparison between results generated using the hand-crafted mesh and procedures with equal and hyperbolic tangent spacing

This procedure is intended to discretize the parameterized computational mesh presented in Figure 20. In the domain, three of the points are allowed to move: the points between the first and second cone, between the second cone and the horizontal aft section, and the bottom corner of the outlet boundary condition (FF_3). The first two points can move freely as required by the current double cone configuration. The third point is constrained to a set axial location. The downstream point of the second cone W_2 is extruded horizontally until it intersects the maximum axial distance, simulating a constant cross-section body.

Pointwise was utilized to distribute seed points along the connectors which define the domain and subsequently build the mesh using a Delaunay triangulation and advancing front algorithm. For a given double cone configuration, the mesh construction is controlled by a single parameter - the equal wall spacing. The seed points on connectors W_2 and W_3 are distributed uniformly with spacing equal to the equal wall spacing parameter. The connectors describing the symmetry plane and the first cone are populated with points according to a hyperbolic tangent spacing function. The

spacing on both connectors at the leading edge of the double cone is set to be 5×10^{-6} meters and the spacing on the other side of the connectors is set to the equal wall spacing. The inlet, FF_1, and outlet, FF_3, connectors are seeded such that the spacings between points monotonically increases and follows an exponential distribution. The initial spacing on both (where they contact the symmetry plane or W_3) is set to the equal wall spacing to ensure a smooth transition at the corners as well. The rate of exponential growth is selected independently for both connectors such that the final spacing (where they both contact F_2) is very nearly the same. This final spacing is then used to seed points along F_2 with uniform spacing. The intention of the exponential spacing on the inlet and outlet connectors was to keep the mesh density high near the geometry where shocks and other large-gradient phenomenon were expected and the reduce the density in the freestream where fewer cells were needed. The difference between the final spacings on the inlet and outlet were minimized for a growth rate of unity, corresponding to equal spacing. This was undesirable as the density of points in the freestream was intended to be minimized to reduce computational costs. Thus a minimum and maximum ratio between the equal wall spacing and the final inlet and outlet spacings was implemented to drive the solutions to have similar cell size distributions between different double cone configurations.

With all of the required initial points seeded on all connectors, the advancing-front Delaunay triangulation processes was utilized to discretize the interior of the computational domain. The growth rate of cell sizes away from the walls was controlled. Pointwise utilizes a decay rate, which was set to a value of 0.9, corresponding to a growth rate of approximately $\frac{1}{0.9} \approx 1.1$. This measure aids in keeping the spacial density of cells high near the double cone geometry, where the most complex flow phenomena occur. The minimum and maximum equilateral triangle sizes were set to the minimum and maximum spacings in the initial seed points, respectively. This

limited the range of possible triangle sizes to also be more consistent between double cone configurations. The result of this generation algorithm is the unique computational mesh for a single sample in the dataset. Because the interior points are a function of the seed points which are themselves controlled by a single user-specified spacing parameter, the overall resolution of the grid can be modified using this single parameter.

4.6 Grid Independence Study

The construction of the meshing process is controlled by the average wall spacing. As the wall spacing is reduced, the resolution of the entire grid is increased. Therefore a grid convergence study could be conducted by repeatedly reducing the average wall spacing by a factor of two. A grid convergence study was conducted on the the 25-55 double cone verification case utilizing the procedural mesh generation method described in Section 4.5.4. The initial equal wall spacing considered was 2.2×10^{-4} meters. This was the medium grid, the medium-coarse and coarse grids were generated by doubling the equal wall spacing parameter and the medium-fine and fine grids were generated by doubling the spacing parameter. Figure 28 presents the surface pressure distributions obtained from Kestrel for the meshes in the convergence study. The wall spacing for the medium grid is the same as the result shown in Figure 26, yet the computed results are slightly different. This was likely caused by the choices made in the generalization of the process, especially those in regards to the minimum and maximum equilateral triangle size.

As discussed previously, it was more important for the simulations to be used as training data to contain examples of relevant flow phenomena than for them to have high physical accuracy. While there is some small increase in the physical accuracy of the surface pressure distribution with increased mesh resolution, it was determined

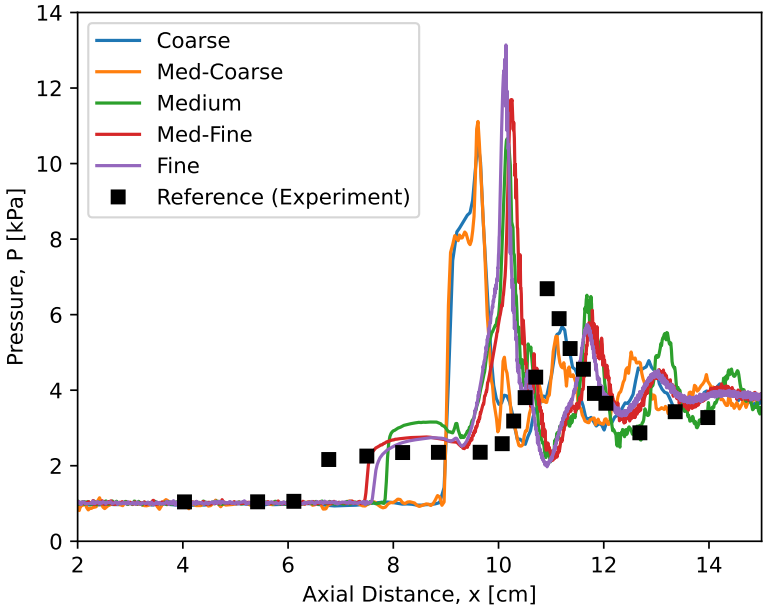


Figure 28. Surface pressure distributions resulting from the grid convergence study

that this was not desirable enough to justify the increased computational resource consumption. Performing grid independence studies throughout the design space to ensure the general mesh quality was beyond the scope of this research. Therefore, for the rest of the simulations in the dataset, the equal wall spacing parameter value of 2.2×10^{-4} meters was used to generate the corresponding computational meshes.

4.7 Data Processing

The training data for the ML models is generated from the steady-state pressure distributions over a parameterized double-cone configuration in hypersonic flight. This pressure data is obtained from the CFD solver package, Kestrel , at each vertex in the computational mesh. The ML models require training data to be in a specific, structured format. This section details the steps taken to obtain the training data in the correct formats for the ML models from the raw simulation output.

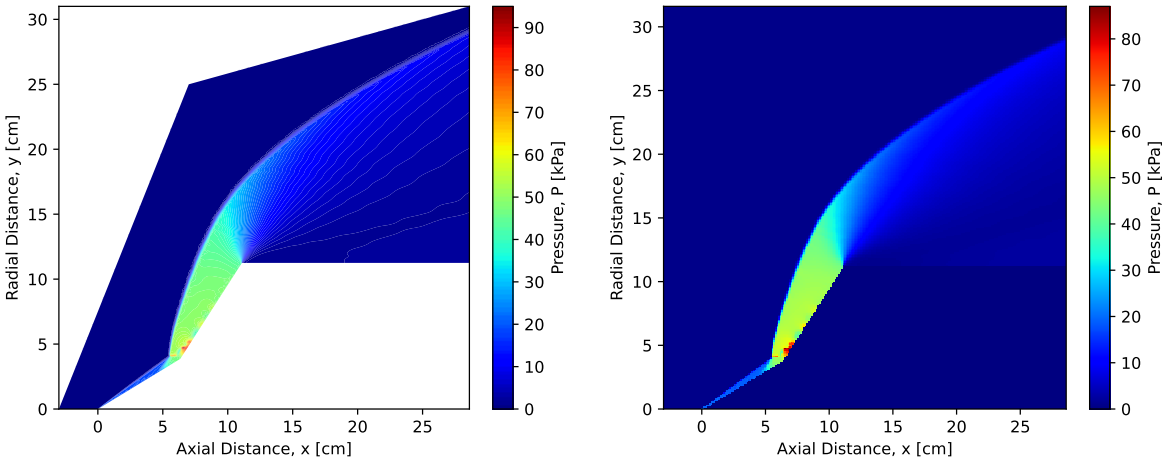
4.7.1 Image Generation.

The data gathered from the simulation output is in an unstructured mesh and must be translated into a structured image in order to be used in the training of the ML models. The Python library scipy [67] contains various interpolation schemes which are suitable for extracting the required structured data from the unstructured mesh. Before extraction, the structured, Cartesian grid must be constructed. The structured grid was constructed such that each cell was square with the same side lengths in the horizontal and vertical directions. The structured grid was constructed such that the horizontal dimension spanned the entire axial dimension of computational mesh (i.e. the extremes of the structured grid x-axis were the same as the extremes of the computational mesh). The horizontal side lengths of each cell were determined by this distance and the image dimension hyperparameter and the vertical side length were set to the same dimension. This algorithm enabled the construction of the vertices of a structured mesh, however for the training data, images must be generated.

The data and responses used in the ML model training are in the form of 2D arrays of values which can be viewed as images. Figure 29 presents a sample both before and after the transformation from unstructured mesh data to an image. Individual samples of the dataset features and responses will be referred to as arrays and images interchangeably. These images are defined by the values at the centers of pixels, not the values of the corners of the pixels. One method of generating pixel center values would be to average the interpolated values at the pixel's corners. This method was not preferred as there are complex flow structures which form at small scales and this averaging would introduce a blurring or dilution of these structures. Alternatively, the values at the center of the pixels could be interpolated from the values at the corners. Again, this would introduce errors in the pixel values. Thus, it was decided to directly interpolate pixel values from the simulation output at the pixel centers.

There are a variety of schemes available which are able to interpolate values from within an 2D unstructured mesh including nearest neighbor, linear, cubic, spline, and heavyside interpolation. Of these options, a linear interpolation scheme was chosen. Kestrel is a cell-centered code, thus only the values at the center of each triangular cell is calculated during the fluid simulations. The output files generated by the utilized version of Kestrel (v11.1) are written with the cell-vertex values interpolated from the solution. These values at the cell vertices are what are presented in this document and are the values from which the datasets are generated. Generating the datasets require interpolating from the unstructured vertices values which have already been interpolated from the cell-centered solution. The interpolation utilized by Kestrel is a linear interpolation of the cell-center values of all cells adjacent to a vertex. The nearest-neighbor interpolation was regarded as not fully utilizing the information available in the calculated response and thus was not used. The linear, cubic, and spline methods had the possibility of introducing new maximum or minimum values beyond those present in the target response. This was deemed undesirable as the interpolated training data should be representative of the target response. Heavyside or RBF interpolation had the potential to be more representative of the calculated response in some areas, however, these methods would introduce undesirable dissipation in regions of large gradients. The linear interpolation method was selected as it would not introduce new artificial extreme values and would minimize dissipation beyond that already introduced by the Kestrel output generation procedure. In this work, this linear interpolation from the unstructured mesh data to the structured pixel-center values was completed using the griddata method from the scipy library.

The computational domain is not the same shape as the required image form of the data. This causes the interpolation function to fail to interpolated at points which are beyond the freestream boundary condition and within the geometry of the double-



(a) Sample simulation pressure field

(b) Image translation of simulation output

Figure 29. Sample of interpolation from simulation output to image used for training

cone configuration. The value of the pressure in the points beyond the freestream boundary condition was set to the freestream pressure as the freestream velocity is supersonic thus pressure disturbances cannot propagate upstream and cause a change. The pixels which were fully inside the double-cone geometry were set to a pressure value of zero. This allows these pixels to act as a mask and provide a sharp gradient in interpolated pressure near the wall (because pressure is always a positive number in simulation output without error). Finally, there are pixels for which the true double-cone surface passes through the pixel. In an effort to encode as much information about the target values from the simulation, these cells were treated in a special manner.

For the pixels which included part of the true double-cone surface, the pressure interpolated at the cell center was not used. Instead, the cell center was projected onto the double-cone surface and the surface pressure at that location was used for the pixel value. This method of interpolation was used because the surface pressure distribution predictions of the ML is of interest. Thus, the target values were included so that the models would have access to the target distribution. Otherwise, a pressure value distant from the body which was not similar to the wall pressure may have been

used. This decision to introduce additional information where available is similar to the decisions made in Section 2.7.2.

4.7.2 Data Preparation.

With all of the simulation data interpolated into a form which is compatible with the ML models, the final step in preparing the data is to ensure that the data satisfies any assumptions made by the models.

The NN-based models (U-Net and MS-Net) do not have any underlying assumptions about the training data which must be addressed. However, due to the greedy nature of the gradient descent training algorithm, the interpolated pressure fields and freestream dynamic pressure are not used directly. Both values in both distributions are far from normally or uniformly distributed (Figure 16b). Because Mean Squared Error (MSE) is used to calculate the loss between predictions and the target values, there is a risk that these outliers will shift the mean prediction of the NN models. To mitigate the effect of these outliers, the base-ten logarithm of the static and dynamic pressures are used. This operation transforms the distributions into more normally distributed, pulling in all of the right-tail outliers (Figure 16c).

For the k-NN and RK models, the unaltered pressure response values are utilized, however the design variables are standardized utilizing the formula:

$$x_{i,j} = \frac{x_{i,j} - \bar{x_i}}{\sigma(x_i)} \tag{32}$$

where $x_{i,j}$ is the ith design variable of the jth sample, $\bar{x_i}$ is the mean value of the ith design variable over all samples, and $\sigma(x_i)$ is the standard deviation of the same. This formulation centers all samples around the origin and contracts each dimension to have the same standard deviation of unity. Standardizing the input data for these distance-based algorithms is necessary because of the vastly different scales present

in the design space (Table 1). Without some transformation, the distance metric would be primarily driven by the variable of largest magnitude (Altitude). None of the simulations which comprised the final dataset experienced errors, therefore there is no missing data from the dataset and no samples had to be discarded or imputed.

Due to the axisymmetric nature of the dataset, it is possible to easily and efficiently augment the training data. The simulation results are axisymmetric and are valid regardless of circumferential location, thus the results can be mirrored to produce a valid result of the full axial slice of the double cone flow field. In the original results, there is only the top portion of the double cone, but these mirrored results include both the top and bottom as well as the tip of the double cone. This introduces data on sharp corners and on how shock waves can propagate and interact on the underside of a hypersonic body. Because the models are built to operate on constant sized inputs, the double sized flowfield must be cropped to an appropriate height. This can either be done such that the LE is always in the center of the image, or a random crop can be applied such that the LE of the upstream cone can be at any vertical location from the bottom of the image (recreating the original training data), at the top of the image (the mirror image of the original training data), or at a random location between. The cropping is random for each distinct sample and retains the width of the image as the width is not modified by the mirroring application. A selection of randomly cropped target responses are presented in Figure 30. The advantage of a random cropping scheme is the random translation of the features of interest which can help the network learn relations which are translation invariant [68], [69]. This forces the model to learn the features directly instead of more abstract information about the features. For example, if no augmentation is applied, then the model could always predict a value of zero pressure along the bottom and be correct, however, translation of the double cone invalidates this strategy.

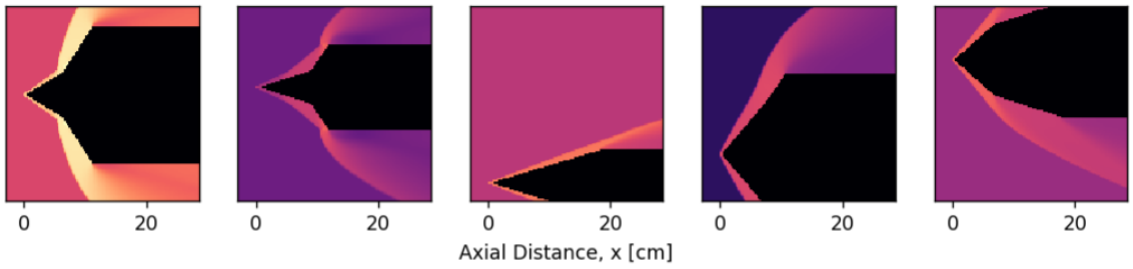
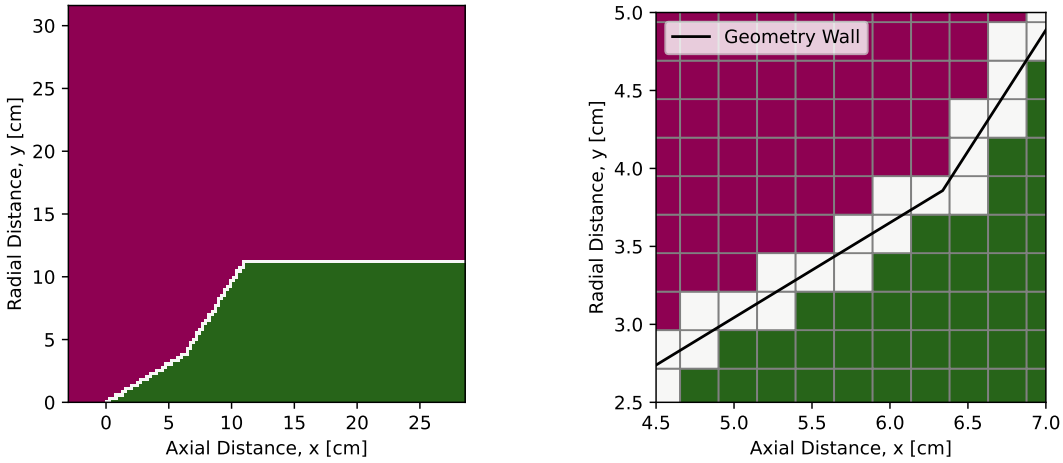


Figure 30. Selection of five randomly cropped target flowfield responses

4.7.3 Insertion and Extraction of Surface Values.

The surface pressures present on the walls of the sample double cone configurations were of interest to this research. As such, in order to provide as much relevant information to the Machine Learning (ML) models, the surface pressure values were directly inserted into the training data. This insertion removes ambiguity regarding the desired surface pressure distribution as the values which the ML model predictions are compared against are not obfuscated. These values are explicitly included within the images of the training data. Additionally, performing this insertion in a standardized way allows for the inverse operation, extraction of the surface values from the ML predictions, to be performed in a comparable manner. This consistent manner of requesting and retrieving the surface values using the model inputs and outputs, respectively, was implemented so that the target response was clear and the models would have the data necessary to form relationships to produce these distributions along the edges of given geometries.

When the images for the dataset are generated from the simulation output (i.e. interpolated to a Cartesian grid), three types of cells arise. Pixels (Cartesian cells) can be fully contained within the fluid flow, fully contained within geometry of the corresponding double cone configuration, or can intersect the surface of the geometry. Figure 31a present the classification of each pixel for a sample double cone configuration for the entire domain. The pixels which are dark green reside entirely inside of



(a) Cells within the flowfield (purple), within the geometry (green), and cells which encompass the geometry wall

(b) Enlarged view of compression corner with geometry wall and cell boundaries overlaid

Figure 31. Visual identification of the cells which lie entirely within the fluid flow, within the configuration geometry, and those in which the geometry boundary is immersed

the of the double cone configuration's geometry and the pixels which are purple reside within the flowfield. The pixels which intersect the geometry boundary are colored white. Figure 31b presents a zoomed-in view of the image focused near the compression corner between the two cones. Additionally, the boundaries between the pixels have been denoted with grey lines and the surface of the double cone configuration has been overlaid in black. Note that all cells which the surface passes through have been identified and colored in white.

Those cells which do not lie on the boundary between the fluid domain and the geometry are assigned a value interpolated from the flow field or a fill value of zero, respectively. The cells which lie on the boundary by definition contain portions of both fluid and geometry. These cells were used to insert the target wall values into the dataset images. This process is depicted in Figure 32. All pixels were defined by a single value. In the flowfield, this was interpolated at the centroid of the pixel. For the boundary pixels, their values were interpolated from the surface pressure output by the CFD simulation. This computed surface pressure distribution along the wall is

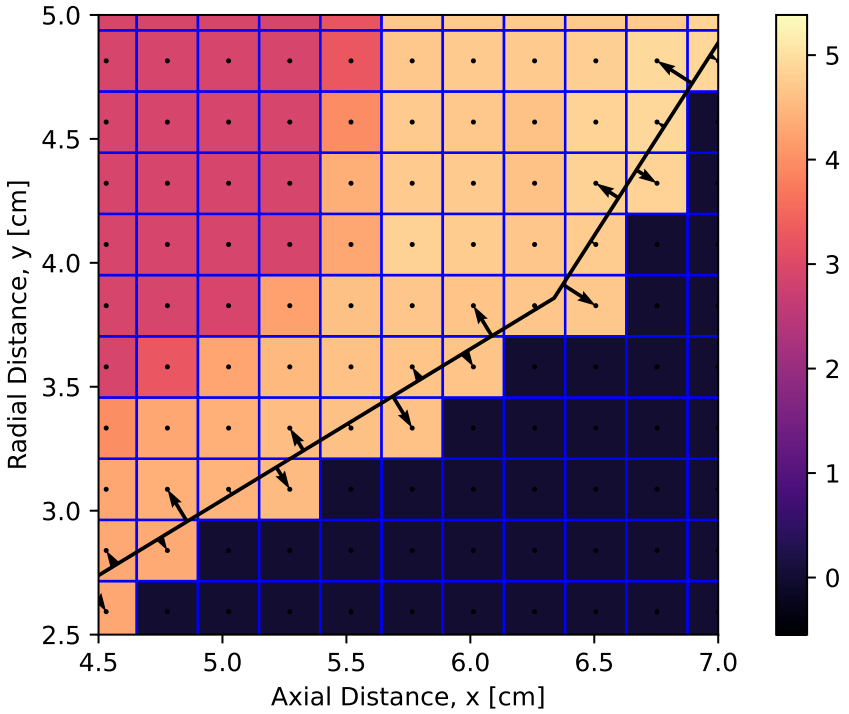
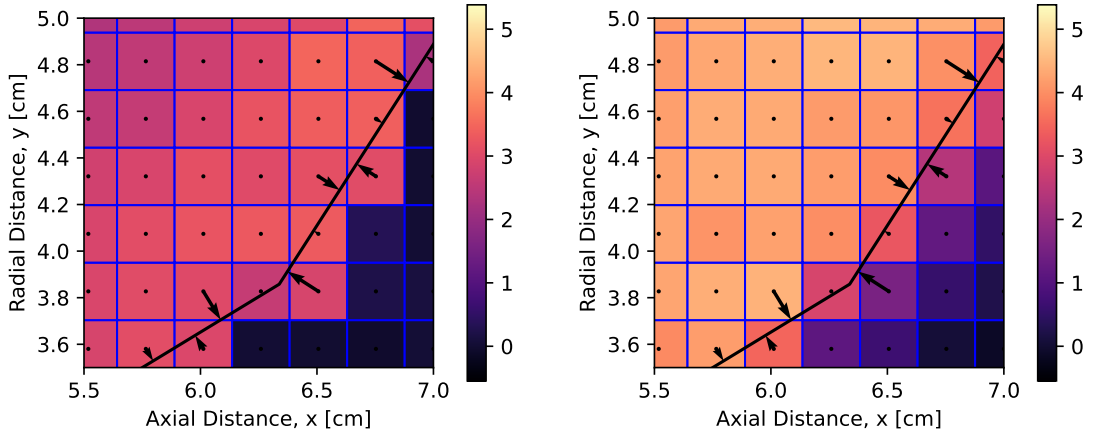


Figure 32. Values interpolated from the target wall distribution at closest points along the wall are inserted into the corresponding pixels

defined in terms of the axial distance from the leading edge and is densely populated compared to the pixel density. The centroids of the boundary pixels were projected onto the wall and the simulation surface pressure was interpolated at all corresponding axial positions. The resulting values were inserted into the corresponding pixels, thus the target wall pressure distribution was present alongside the target flowfield in a spatially-intuitive manner without sacrificing the inclusion of other relevant data.

The extraction of the predicted surface value distribution utilized the inverse of the insertion operation and is presented in Figure 33. The centroids of the boundary pixels were again projected onto the true, known double cone configuration wall to obtain the axial distance which each pixel accounts for. This is the same procedure as in the insertion and produces the same axial location results. The predicted value at each boundary pixel is then used in conjunction with the corresponding projected axial location to produce the predicted surface pressure distribution. This distribution can



(a) Extraction process from sample predicted response

(b) Extraction process is agnostic of the predicted geometry edge and only utilizes the true geometry

Figure 33. Extraction process is the inverse of the insertion process. Boundary cell centroids are projected onto the wall and the resulting axial distances and corresponding pixel values are used to build the predicted wall distribution.

be extracted from any predicted flow field if the true configuration geometry is known. In all of the cases studied, the geometry is known and is used in the production of the model inputs. Figure 33a shows an extraction from a flowfield where the predicted edge of the geometry is exactly the same as the input image. Figure 33b presents a case where the predicted flow field does not perfectly return the geometry in the input. In this case, the extraction process is no different than in any other case. The same pixels are utilized in the same manner.

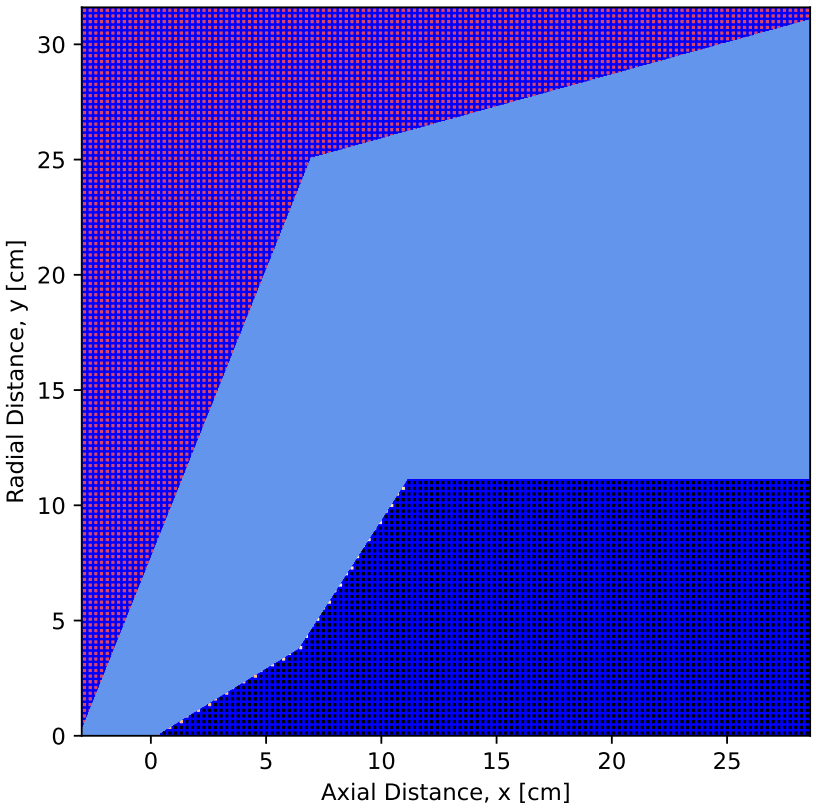
4.7.4 Interpolation Error Quantification.

The process of interpolation of target responses from the unstructured computational mesh to the Cartesian grids (images) used in training the ML models introduces error. This error arises from the fact that the Cartesian cells (pixels) are large compared to the triangular cells of the unstructured mesh. In the areas near the wall, the pixels can be large enough to contain tens or more triangular cells. The scale of the pixels can be large enough to obfuscate or erase local, small-scale flow phenomenon. Figure 34 presents the unstructured mesh (light blue) overlaid on the interpolated

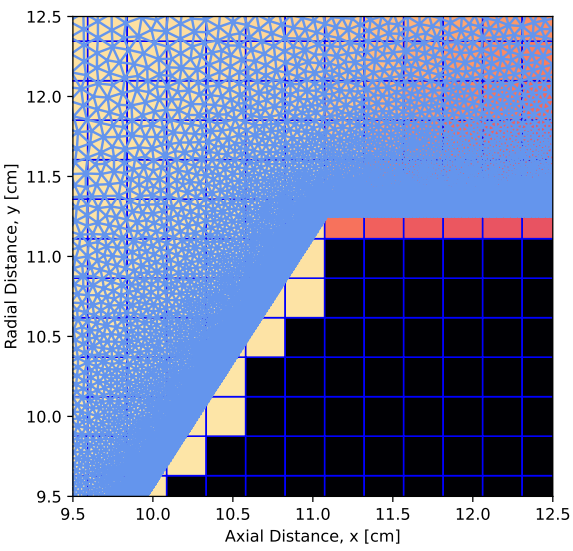
128×128 Cartesian grid (dark blue). Figure 35 presents a zoomed view near the surface of the double cone configuration where many triangular cells are contained within a single pixel. The local variations in the simulation output are replaced by a single value interpolated at the centroid of each pixel (displayed using black dots). Figure 36 presents a comparison in the far field where the unstructured mesh size is more comparable to the 128×128 Cartesian grid cell sizes.

The effects of this interpolation error were studied for three grid resolutions: 64×64, 128×128, and 256×256. Figure 37 presents the same target flowfield interpolated to each of the three resolutions for visual reference. In order to calculate the error introduced by the interpolation from the unstructured mesh to the Cartesian mesh (or image), the values in each image resolution were projected back onto the unstructured mesh. The result of this is shown in Figure 38. Figure 39 presents the absolute differences between the original target and the values reconstructed from the corresponding images. As is expected, the errors are largest in areas of high gradients and generally reproduce the square shapes of the image pixels. Figure 40 presents the distribution of errors from Figure 39. Much of the flowfield has little or no error introduced by the interpolation, resulting in a large peak at a difference of zero.

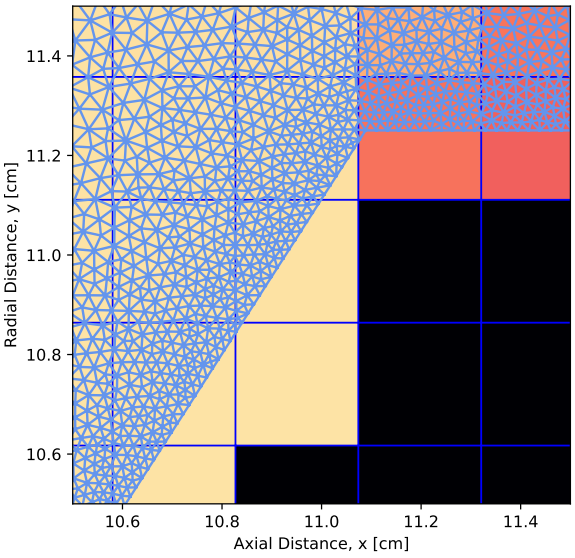
The effect of the Cartesian grid resolution on these introduced errors was quantified over the entire dataset. For each resolution, the error values between the unstructured simulation output and the reconstructed unstructured mesh were calculated for each sample. These values were subsequently aggregated and analyzed together. Because a majority of points had little to no associated error, these values dominate the overall distribution. In order to more effectively visualize the distribution of errors, the error values within 0.5% of zero were removed. Approximately 40% of all errors fell within this removed range. The resulting distribution for the 64×64 pixel training data are presented in Figure 41. A Kernel Density Estimate (KDE) is also fitted to



(a) Full computational domain and generated image

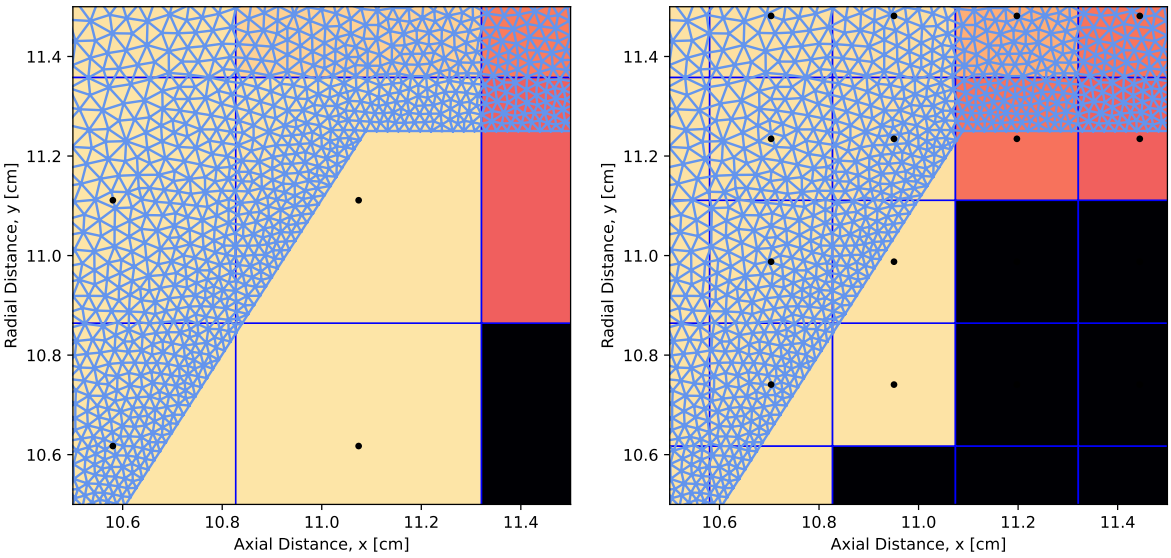


(b) View near expansion corner



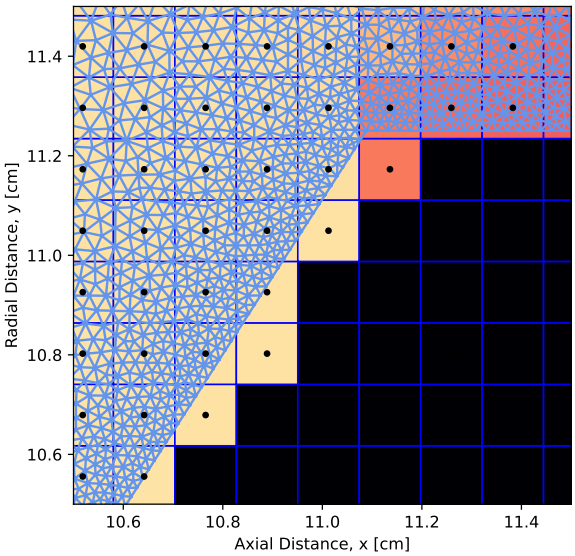
(c) Further zoomed view near expansion corner

Figure 34. Comparison of cell sizes of the unstructured mesh (light blue) and the 128×128 Cartesian grid cells (dark blue) for the (a) entire computational domain and (b)-(c) near the compression corner.



(a) Resolution 64×64 grid

(b) Resolution 128×128 grid



(c) Resolution 256×256 grid

Figure 35. Comparison of the cell sizes of the unstructured mesh (light blue) and the interpolated Cartesian grid cells (dark blue) near the expansion corner for the three considered resolutions. The Cartesian cell centroids are indicated with black dots.

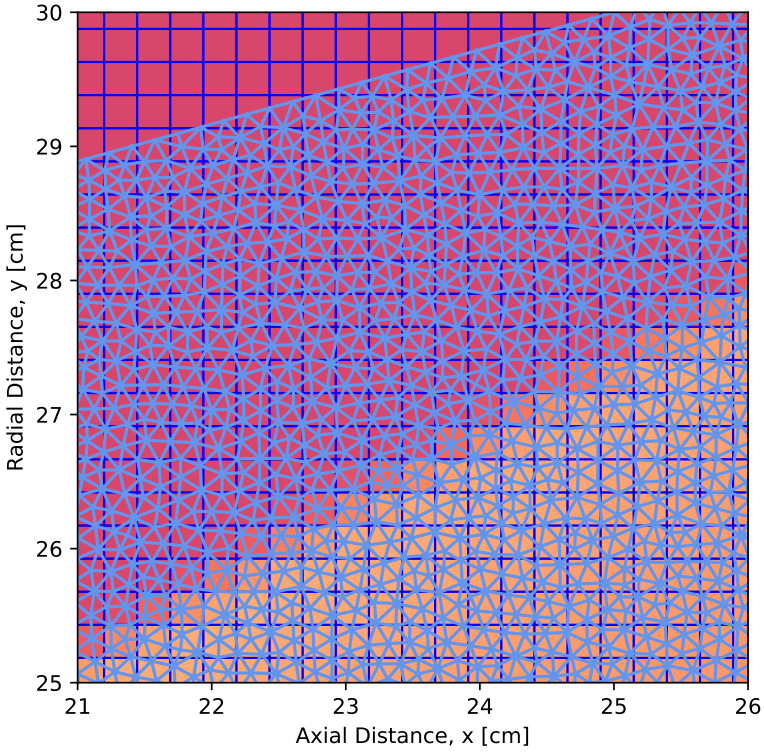
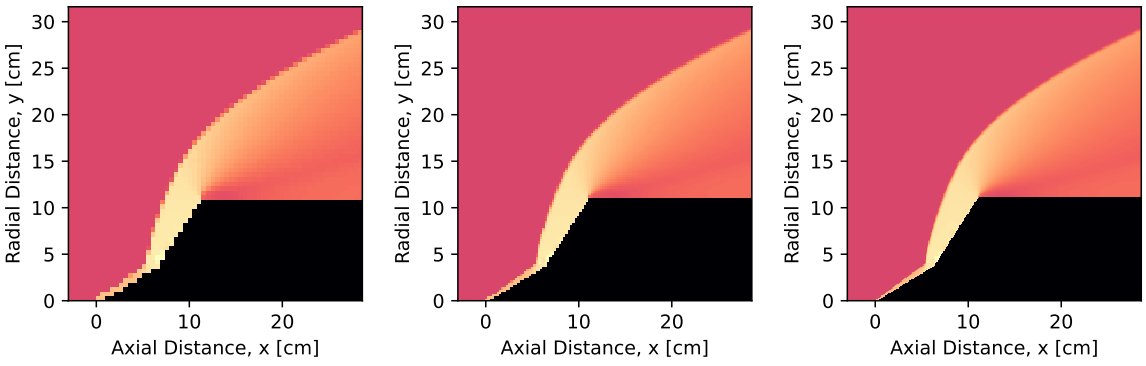
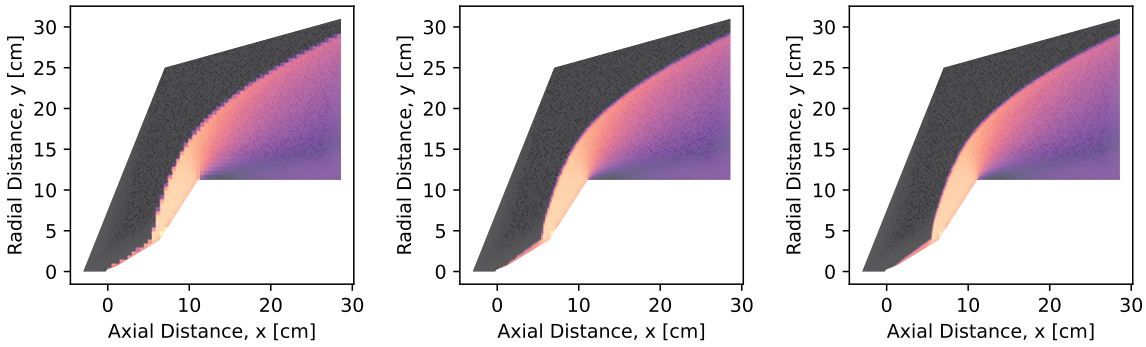


Figure 36. Comparison of the cell sizes of the unstructured mesh (light blue) and the interpolated Cartesian grid cells (dark blue) near the far field boundary condition.



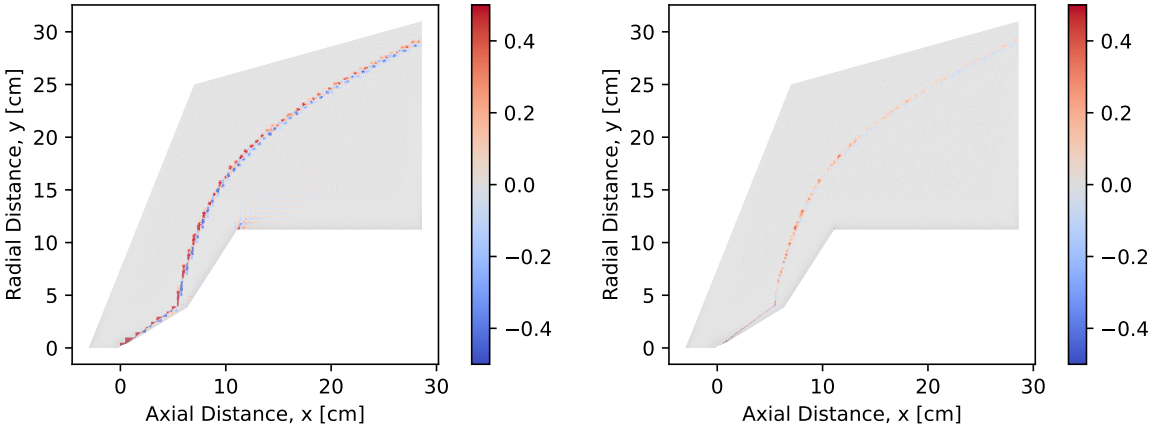
(a) Interpolated to 64×64 grid (b) Interpolated to 128×128 grid(c) Interpolated to 256×256 grid

Figure 37. Visual comparison of the unstructured simulation output interpolated to the three Cartesian grid resolutions of interest



(a) From 64×64 grid (b) From 128×128 grid (c) From 256×256 grid

Figure 38. Visual comparison of the re-interpolation quality of different Cartesian grid resolutions back onto the original unstructured mesh.



(a) Error in re-interpolation from 64×64 grid (b) Error in re-interpolation from 256×256 grid

Figure 39. Visual comparison of the error introduced by the interpolation process. The 128×128 resolution grid is omitted for brevity and the error magnitudes have been clipped for clarity.

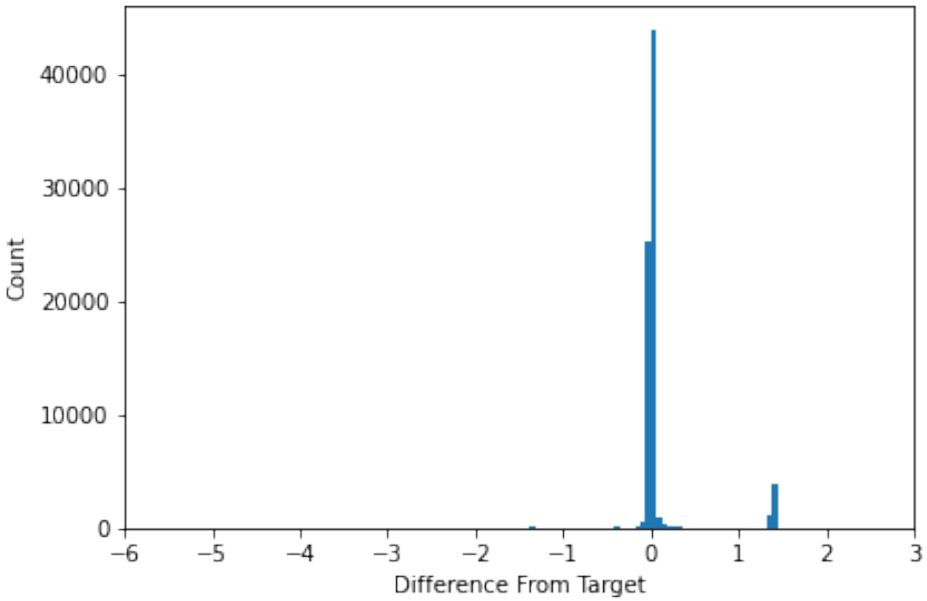


Figure 40. Distribution of errors introduced by the interpolation process for the case presented in Figure 39a

this data. This reduced error value distribution appears to be bi-modal with a mean value less than zero, around minus one and long, heavy tails. The median value of all error values is approximately -0.32. This indicates that the interpolation scheme used to create the Cartesian grid generally causes under-prediction of the target flow field response where there are introduced errors. The target response values within the dataset range from approximately 0.66 to 7.04, covering a range of 6.38 units. Therefore, a pixel which introduces appreciable error will, on average, introduce an error on the order of 15% of the total range of the dataset responses. Figure 42 presents the KDE representations (scaled to have total area of one) for the error distributions of all three resolutions. The errors of all resolutions converged on the same distribution. These discrepancies could be reduced by utilizing more advanced interpolation methods, however, as shown in Figure 39, these errors are concentrated near regions of strong gradients and are generally only one pixel in axial width. In practice, these errors could prove to be beneficial as they indicate that the pixel value

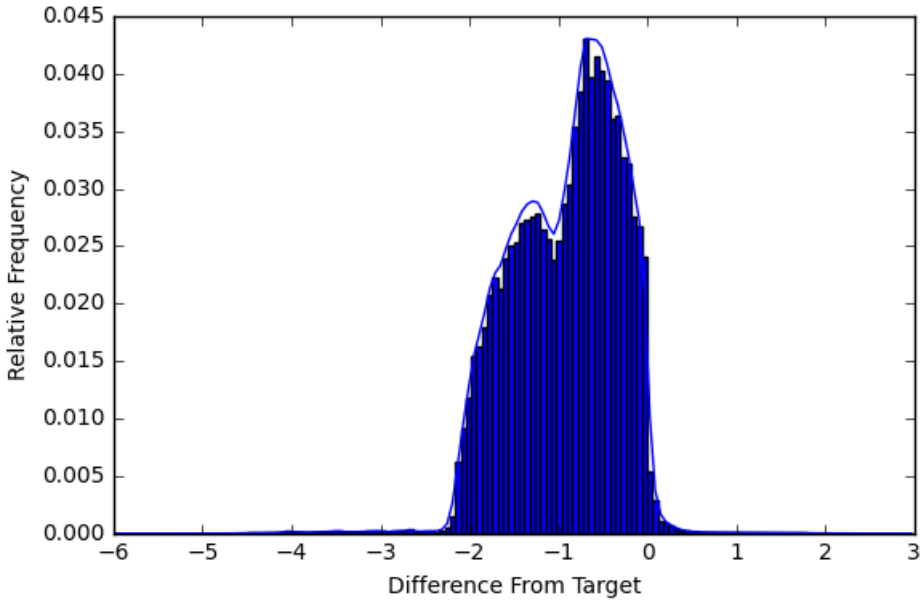


Figure 41. Distribution of all errors introduced across all samples in the dataset for the 64×64 resolution Cartesian grid. The values with a distance of less than 0.5% of the total error range from zero are excluded for clarity (approximately 40% of all error values).

is biased to be on one side of these strong gradients. If pixels are given values which correspond to either the upstream or downstream sides of a shock wave instead of the mean value inside the shock, the representation of the shock wave in the training image will be more defined. Flow features which are more distinctly defined and experience less diffusion are likely to simpler to capture with the neural network based models.

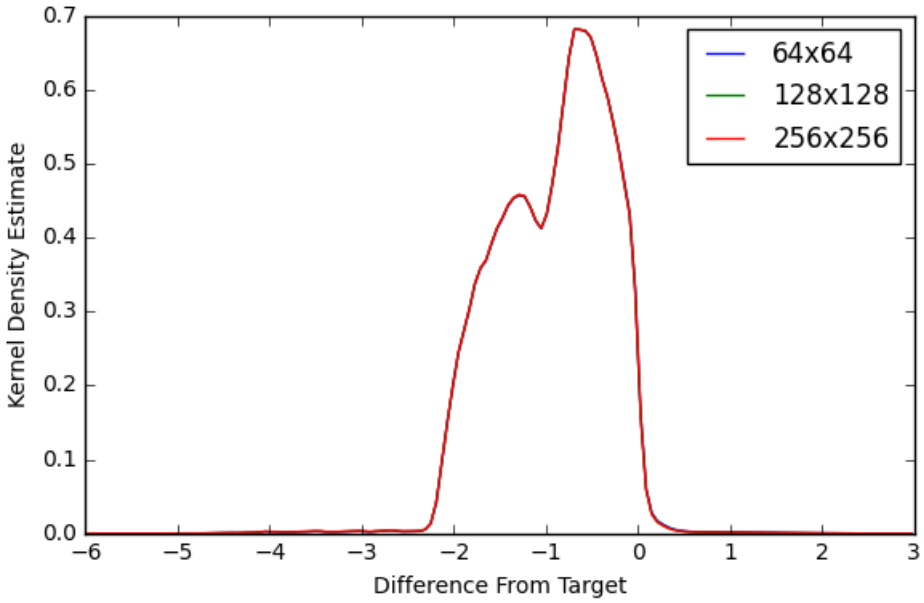


Figure 42. Kernel density estimation for the distribution of all errors introduced across all samples in the dataset for all resolutions of the Cartesian grid. The values with a distance of less than 0.5% of the total error range from zero are excluded for clarity (approximately 40% of all error values). The errors introduced at all three resolutions converged on the same distribution.

V. Results

In this chapter, the training and hyperparameter optimization results are detailed. The performance of the resulting models with optimal hyperparameters are presented. Each model is used to predict the pressure field for each configuration on a 128×128 Cartesian grid (producing an image as described previously). The Mean Squared Error (MSE) between the predictions and the target pressure field are considered. Further, the MSE between the corresponding, encoded surface pressure distributions and the target surface pressure distributions are analyzed.

5.1 Model Tuning

As discussed previously, each of the surrogate models explored in this research have hyperparameters which introduce degrees of freedom into the construction or architecture of the model. The choice of these parameters can have a significant effect on the final performance of the model, thus some optimization is run in the hyperparameter design space to maximize model performance. The k-Nearest Neighbors (k-NN) model has two hyperparameters, the number of neighbors used in calculating the prediction at the query point and the exponent of the Minkowsky distance metric which defines the distance between the query point and the training samples. The GPR model has one hyperparameter: the kernel type. The U-Net model has two hyperparameters: the initial learning rate for the gradient descent step and the number of channels created by the first layer. The Multiscale Network (MS-Net) model similarly has two hyperparameters: the learning rate and the number of channels in the lowest-resolution sub-model. As described previously, the selection of the optimal hyperparameters is made using the training and validation datasets. The candidate model with the current hyperparameter settings is trained on the training data and

its performance is evaluated on the validation set. The set of hyperparameters which yields the best cross-validation performance is selected. The corresponding model is then trained using both the training and validation sets and then evaluated on the test set.

The Mean Squared Error (MSE) was utilized as the loss function for the k-NN and GPR models in order to determine which hyperparameter configuration was the optimal. This choice was driven by the desire for the model to effectively model peak pressures. The dataset contains large peaks in pressure where shock impingements occur. Because these large pressures are what drive the sizing or material choice constraints at those locations, it is important to accurately capture these peaks. The MSE algorithm penalizes points based on the square to the difference between the prediction and the target distribution - many small deviations from the target value accrue relatively small penalties compared to a few large deviations from the target distribution. Thus, the algorithm should attempt to match the peak pressures first in a greedy manner.

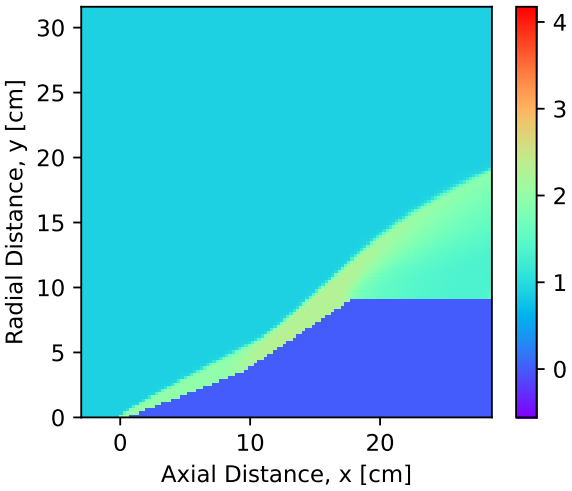
Both of the CNN-based models have no restriction on their input size, however training and evaluation times are influenced by the resolution of the input data. This research did not have a CUDA-enabled GPU available for model training which limited the input sizes which could be efficiently computed. An input size of 128×128 pixels was chosen for use with both models. This is also the resolution used by [24] in their study of their U-Net architecture. Utilizing the same input resolution should allow for the most direct comparison of the U-Net and MS-Net architectures as well as the other two model types.

It should be noted that two main datasets were considered in the generation of these models. The output from the CFD simulations contain the static pressure at all mesh nodes. The dataset was originally constructed utilizing these static pressure

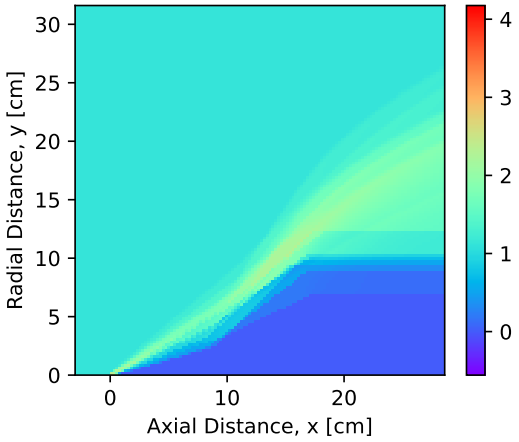
values. The pressures in the resulting data varied by many orders of magnitude both between samples and within samples. These large fluctuations in the desired response were theorized to reduce the performance of the CNN-based models as the kernels and constituent weights are, by definition, constant values while generating predictions. Thus if the models were tuned to best predict one section of the spectrum of possible pressure values, the performance elsewhere in the design space would suffer. Thus, a transformation was applied to this dataset in order to bring the distribution of responses into a more well behaved space where the range of the responses was more manageable for the models and the distribution of values within the space was more uniform such that one space would not have a disproportionate influence in the model training. The base-ten logarithm was selected as the transformation. Initial testing with the original static pressure dataset showed relatively poor performance with the k-NN model and the CNN-based models, especially when compared with the transformed data. Thus the transformed dataset consisting of the log base-ten of the static pressure was utilized for the remainder of the study. Representative outputs of all models using this transformed dataset are presented in Figure 43.

5.1.1 k-Nearest Neighbors.

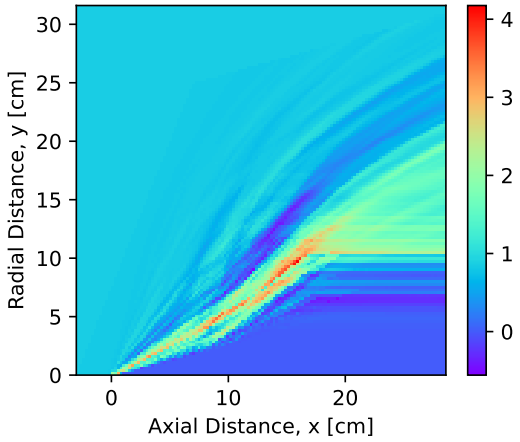
For the k-Nearest Neighbors (k-NN) model, the number of neighbors to consider, k, is restricted to integer values between one and the number of samples in the training set. If the maximum number of neighbors is utilized, then the result for every query would simply be the mean of the entire training dataset. Selecting a large value of k reduces the variance of the model at the cost of introducing bias. To reduce the search space and limit the maximum bias configuration in the search space, the range of k was restricted to be approximately 10% of the number of samples, or 100 neighbors. The coefficient of the Minkowsky distance metric can take any real, non-zero value



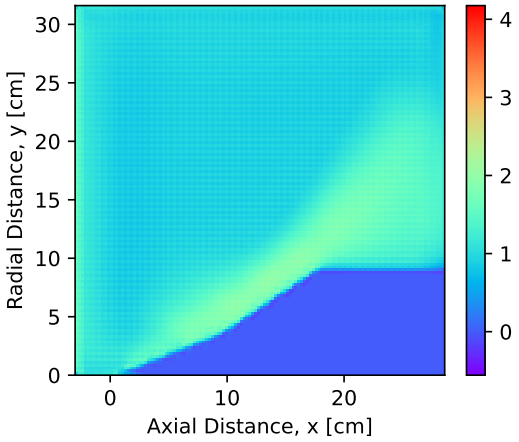
(a) Target Response



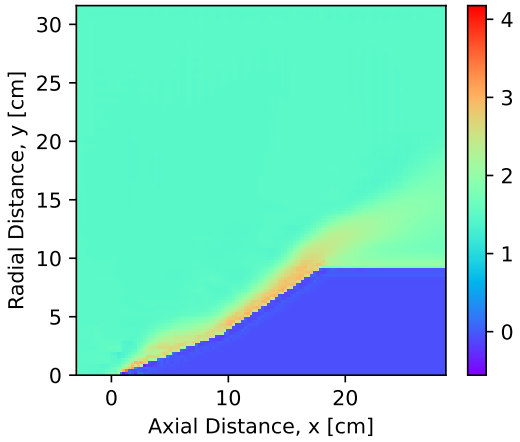
(b) k-NN Prediction



(c) RK Prediction

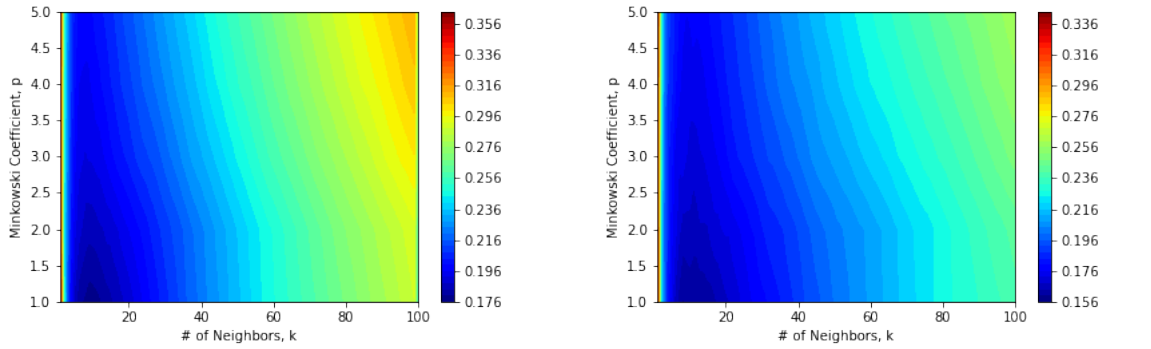


(d) U-Net Prediction



(e) MS-Net Prediction

Figure 43. The target (a) and predicted (b)-(e) flow field response for a configuration which is not in the training dataset. The flight conditions for this sample are Mach 6.74 at an altitude of 66.4 km.



(a) Loss surface for uniform weighting function (b) Loss surface for distance weighting function

Figure 44. The surface of the MSE loss of the k-NN algorithm produced during the grid search over the hyperparameter space. These results are the logarithmic transform of the data.

however it is generally restricted to whole numbers. For the grid search, the value of the Minkowsky metric coefficient was swept from one to five in steps of 0.1 in order to better capture the loss surface.The hyperparameter optimization consisted of a grid search over the space and utilized a five-fold cross validation technique to compute the candidate model loss at each point. The results of this search are presented in Figure 44. The optimal hyperparameters found in the grid search were determined to be $k = 8$, $p = 1$, and the distance-based weighting scheme. The corresponding loss surface does not display any indication that there is an optimal value outside of the search bounds.

5.1.2 Gaussian Process Regression.

The Regression Kriging-inspired GPR model utilizes the k-NN model as the regression function for the mean component and a GPR model to model the residual component. Thus, the combined model has three hyperparameters: the number of neighbors, k, and Minkowski coefficient, p, from the k-NN model and the correlation function from the GPR model. The hyperparameter optimization for this model was performed in a sequential manner where the hyperparameters of the k-NN model were

optimized first (Figure 44a). The residuals from the k-NN model fit to the training data were used to train the GPR model. The k-NN model was restricted to using the uniform weighting scheme as using the distance scheme results in residuals which are identically zero, which would defeat the purpose of this composite model. The optimal hyperparameters are already available from the search performed in generating Figure 44a: $k = 9$, $p = 1$. The kernels investigated were the standard RBF-based (Radial Basis Function) kernel, a purely RBF kernel, an Exponential Sine Squared kernel, and the Matern kernel. Note that the internal parameters of each kernel are automatically optimized by the search algorithm to find the best possible performance for each kernel. The Exponential Sine Squared kernel was not able to converge on a set of internal parameters. Of the rest of the options, the standard RBF combination kernel was found to produce the best performance.

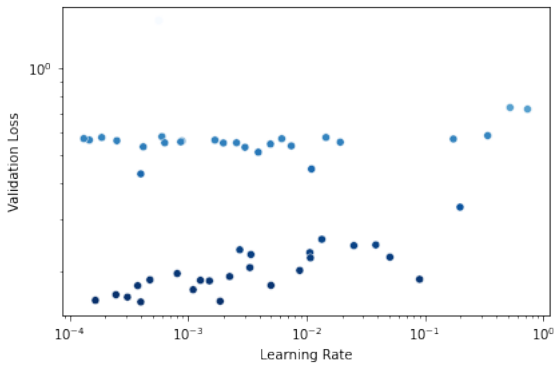
5.1.3 U-Network Architecture.

The number of channels produced by the first layer of the U-Net architecture was constrained to be a power of two, 2^n, such that the exponent, n, could be used as the hyperparameter. The value of this exponential was restricted to whole numbers in the range $[0, 5]$, which corresponds to a minimum of one initial channel and a maximum of 32 channels. The initial learning rate was drawn from a loguniform distribution over the range $[1e-9, 1e0]$. A Bayesian optimization routine was used to explore this hyperparameter space. For each hyperparameter setting, a candidate U-Net model was generated and trained on a subset of the training data. Training consisted of 200 epochs and was terminated early if the loss on the validation set did not improve for 30 epochs. Figure 45 presents the search results after considering 50 candidate models. The best configuration found by the hyperparameter optimization was setting the exponent for the initial number of channels to $n = 4$ (corresponding

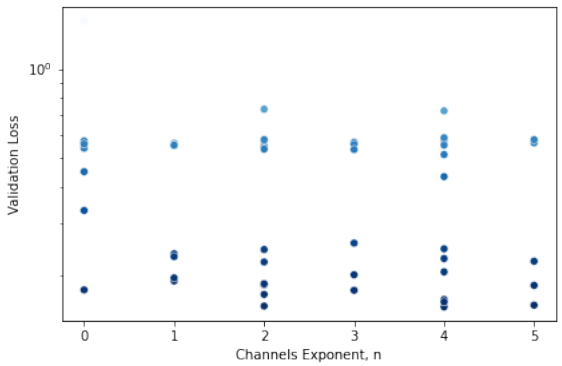
to $2^4 = 16$ total initial channels) and an initial learning rate of $10^{-3.4}$. Figure 45c presents the region of the space which the hyperparameter optimization algorithm explored with the candidate results color-coded by the corresponding validation loss while Figures 45a and 45b present the distribution of resulting validation loss values with respect to the learning rate and exponent, n, respectively. There appears to be a slight positive correlation of the validation loss with respect to the learning rate and no discernible correlation with respect to the initial number of channels. There appear to be two distinct bands in both plots. This differentiation likely is likely a side effect of the restarting of the search algorithm with a less stringent early stopping criterion. The first stage had stricter criteria in an effort to force more exploration in the first half such that the most promising settings could be better exploited in the second half. This exploitation was enabled by loosening the early stopping criteria such that training of the network could span more epochs with the intention of allowing the gradient descent algorithm to find ways out of local minima.

5.1.4 Multiscale Network Architecture.

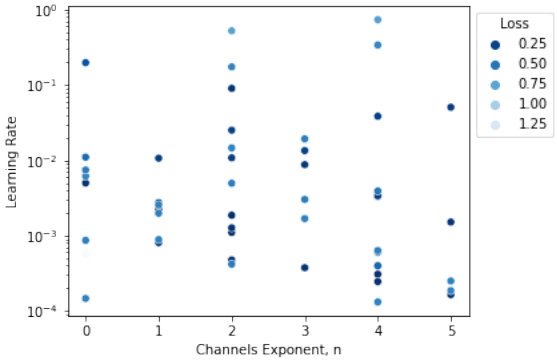
Similarly to the U-Net architecture, the Multiscale Network (MS-Net) architecture is defined using two hyperparameters: the number of channels in the initial sub-model and the learning rate. The number of channels produced by each of the convolutional layers in the five blocks of the final (highest resolution) sub-model was constrained to be a power of two, 2^n, such that the exponent, n, could be used as the hyperparameter. The value of this exponential was restricted to whole numbers in the range $[0, 5]$, which corresponds to a minimum of one initial channel and a maximum of 32 channels. The learning rate was drawn from a loguniform distribution over the range $[1e-9, 1e0]$. The same Bayesian optimization routine and setup as the U-Net were used to explore the hyperparameter space for the MS-Net. Figure 46 presents the loss results of 68



(a) Variation of loss with respect to learning rate

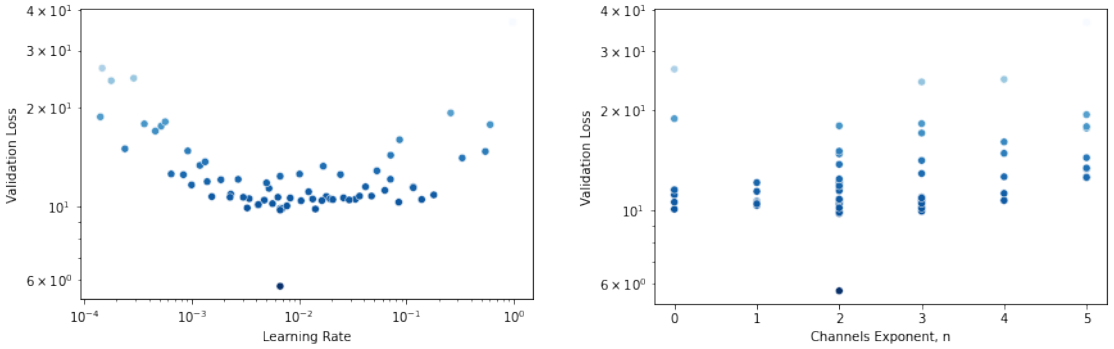


(b) Variation of loss with respect to exponent of the 2^n initial convolutional channels



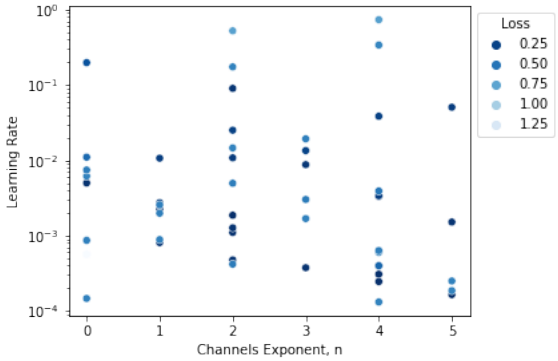
(c) The extent of the design space explored by the algorithm

Figure 45. Hyperparameter settings for the U-Net architecture explored by the hyperparameter optimization algorithm. The displayed data are color-coded by the corresponding model's validation loss (lower is better).



(a) Variation of loss with respect to learning rate

(b) Variation of loss with respect to exponent of the 2^n initial convolutional channels



(c) The extent of the design space explored by the algorithm

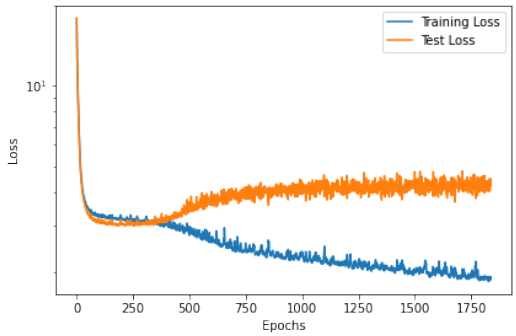
Figure 46. Hyperparameter settings for the MS-Net architecture explored by the hyperparameter optimization algorithm. The displayed data are color-coded by the corresponding model's validation loss (lower is better).

candidate models. The MS-Net validation loss appears to have a strongly parabolic relationship with the learning rate (Figure 46a) and a weak positive, if any, correlation with the number of channels in the layers of the first sub-model (Figure 46b). The best hyperparameter settings were found to be a learning rate of $10^{-2.18}$ and $2^2 = 4$ initial channels. There is no banding in the hyperparameter search for the MS-Net, however the optimal settings found are a major departure from the other results. This could have happened by chance as the weights for each candidate model are randomly initialized.

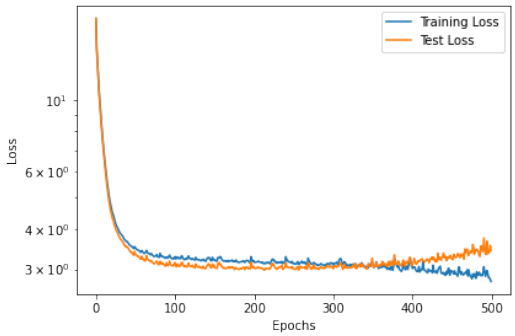
5.1.5 Model Convergence.

The MS-Net model was trained on three resolutions of the dataset in order to investigate the effect of the resolution on the relative prediction quality. The models were all built using the optimal hyperparameter settings found previously. Figure 47 presents the model losses over the course of the respective training epochs. The training of the models with resolution 64×64 and 128×128 (Figures 47a,47b and 47c, respectively) both exhibit overfitting. Overfitting can be identified by the divergence of the loss on the training and test set losses. In the beginning of training, the model initially learns features which are general to the underlying dataset distribution, driving down loss for both training samples and samples not used in training (the test set). After most of the relevant, general trends have been learned, the model begins to further extract information which is only present in the training data. This causes the loss on the training data to decrease. However, these secondary relations are not general and only apply to the training data, therefore biasing performance toward the training data. This causes the loss on the test set to rise. It is desirable to train such neural network models until their performance on the test set stops improving. For these models, this was accomplished by saving a checkpoint of the model at the training epoch where the test set loss was at a minimum. These checkpoints were then used to generate wall results.

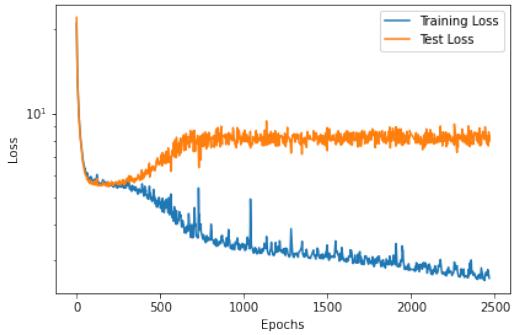
A selection of samples were selected and the predicted wall results of all three models for each are presented in Figures 48-50. The cases for which the error between the predicted wall distributions and the target distribution is maximized (the worst predictions) for each resolution are presented in Figure 48. The cases for which the error is minimized (the best predictions) for the same are presented in Figure 49. A selection of other, more representative samples are presented in Figure 50. From the worst predictions, the distributions predicted at each resolution can vary



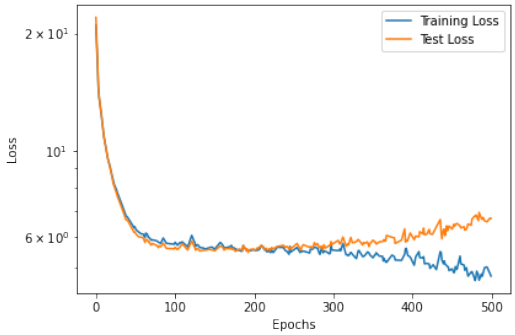
(a) Resolution 64×64 training losses



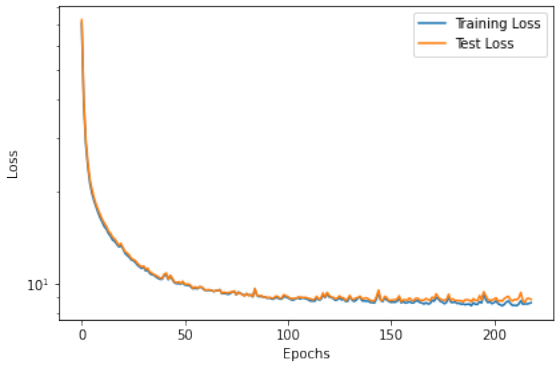
(b) Start of overfitting for resolution 64×64



(c) Resolution 128×128 training losses



(d) Start of overfitting for resolution 128×128



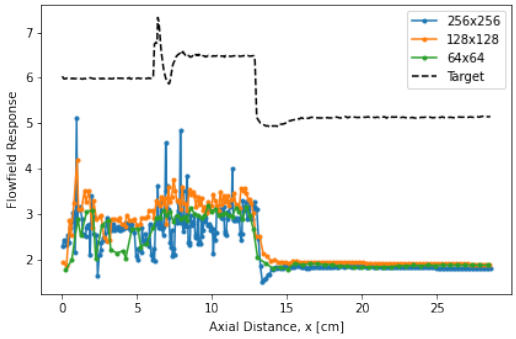
(e) Resolution 256×256 training losses

Figure 47. Training progression for the three resolutions of the MS-Net architecture: (a)-(b) 64×64, (c)-(d) 128×128, (e) 256×256. The two lower resolutions exhibit signs of overfitting and a view of training up until the start of overfitting is also provided. The 256×256 resolution model was trained until the start of overfitting was detected as the training of this resolution was more computationally expensive than the other two. For all models, the state of the model at the minimum test loss is utilized.

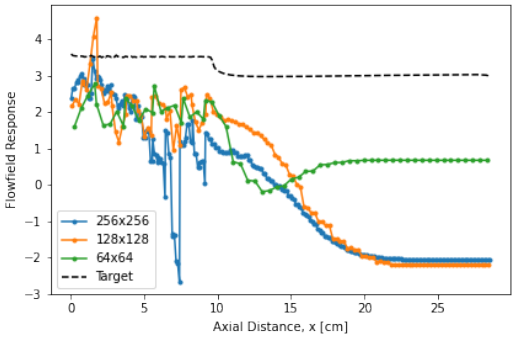
by a significant margin compared to the target wall distribution. However, there are many samples in which the predictions made on the three resolutions tend to oscillate around similar mean distributions. In general, the density of points in the prediction and the corresponding oscillation frequency and amplitude increase as the resolution of the image dataset is increased. Especially in the case of the 256×256 resolution dataset, there are consistently very large spikes in the predictions which occur for only one or a small number of consecutive points (notably in Figures 48c and 49b). Unlike in the computational mesh convergence study for the fluid dynamics simulation (Section 4.6), the wall results predicted by the MS-Net model do not converge to a single distribution as the resolution of the Cartesian grid is increased. Figure 51 shows the distribution of Mean Squared Error (MSE) errors between the target and predicted wall distributions for all samples. Overall, training the MS-Net model on the 128×128 resolution dataset produced the smallest average loss for the wall predictions. Therefore, the 128×128 resolution Cartesian grid will be utilized for subsequent analysis as it represents a compromise between solution stability (where 64×64 has the least significant oscillations) and solution resolution (where a larger resolution is more desirable) in addition to having the best average performance of the three resolutions investigated.

5.2 Surface Distributions

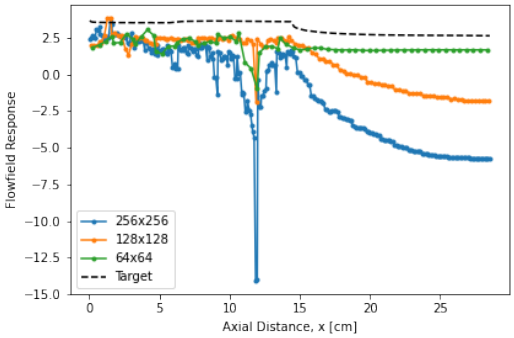
Two metrics for evaluating the generated models are analyzed in this section. The first is the overall quality of the regression fit produced by each model. This is measured by calculating the Mean Squared Error (MSE) between the target surface pressure distribution and the corresponding prediction made by the models. The models' prediction of the surface pressure is extracted from the prediction of the flow field using the procedure described in Section 4.7.3. The MSE metric utilizes



(a) Worst performance for 64×64 resolution

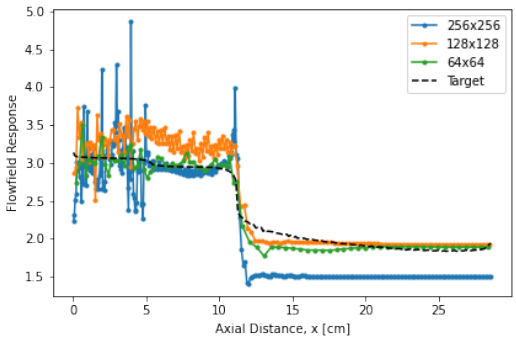


(b) Worst performance for 128×128 resolution

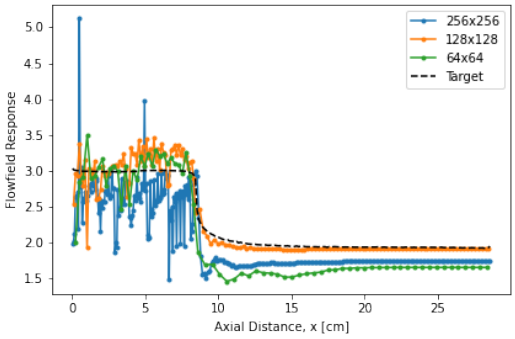


(c) Worst performance for 256×256 resolution

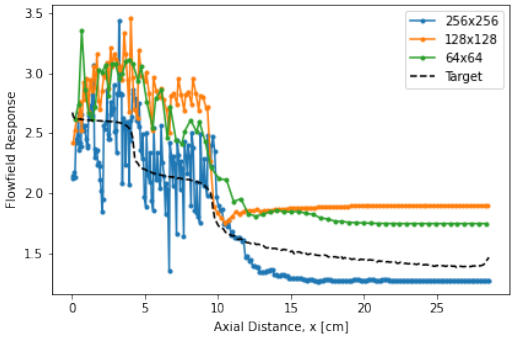
Figure 48. Samples for which the three MS-Net models utilizing varied resolutions produced wall value predictions which performed the worst (maximized MSE with respect to the target distribution)



(a) Best performance for 64×64 resolution



(b) Best performance for 128×128 resolution



(c) Best performance for 256×256 resolution

Figure 49. Samples for which the three MS-Net models utilizing varied resolutions produced wall value predictions which performed the best (minimized MSE with respect to the target distribution)

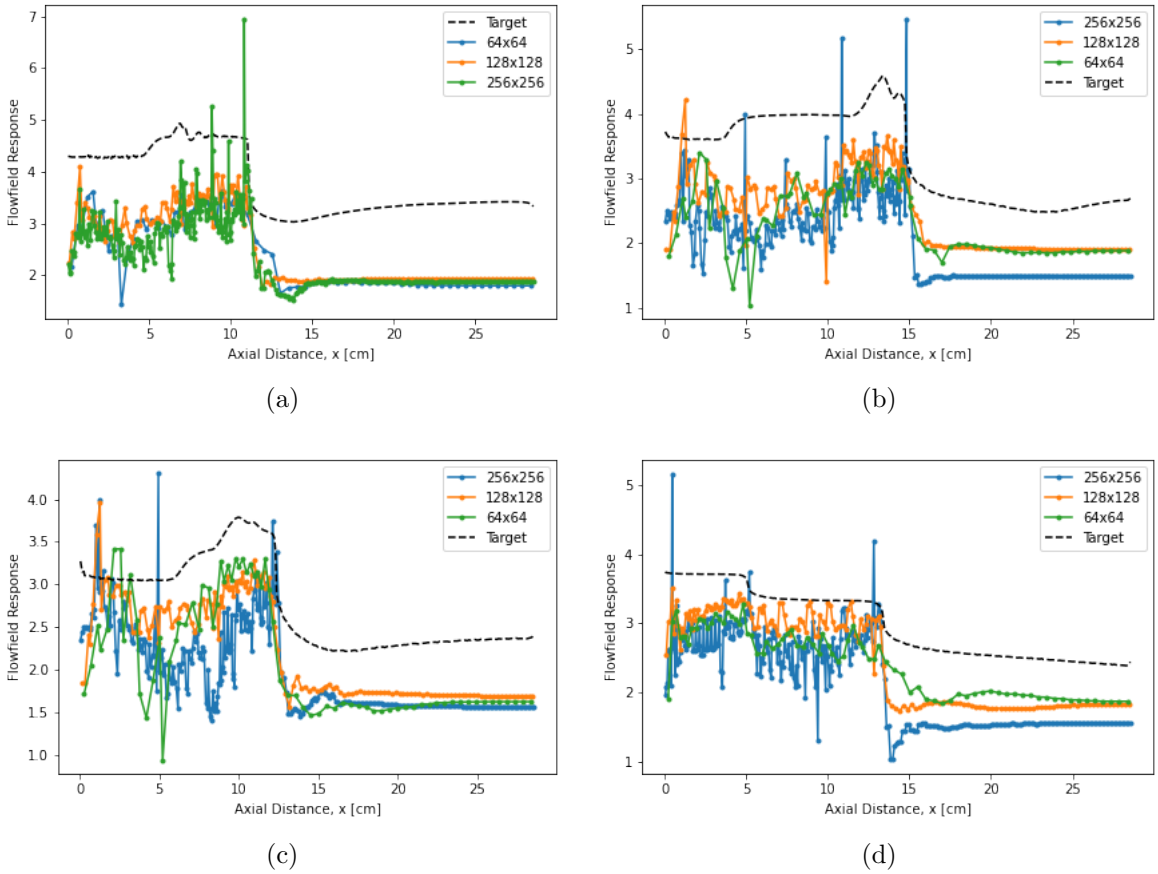


Figure 50. Select samples from the dataset and the respective predictions produced by the three MS-Net models utilizing varying image resolutions

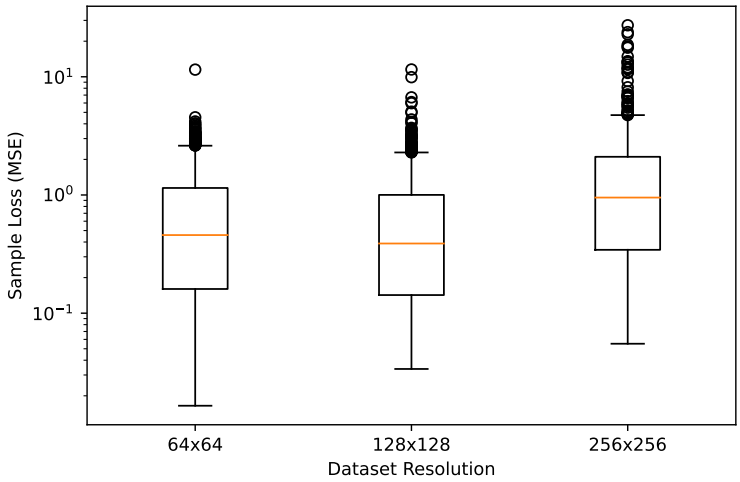


Figure 51. Distribution of MSE loss between predicted and target wall distributions for all samples. Predictions were generated using the MS-Net trained on the three dataset resolutions.

continuous values and serves as a measure of how well the magnitudes of the predicted responses match the target distributions. A lower MSE value indicates better agreement with the target response. The second measure is based on the categorical responses of either a shock impingement being present or not. The accuracy and precision of the models will be evaluated and analyzed. The accuracy of a categorical response is the ratio of the total number of correct classifications divided by the total number of samples and provides the ratio of true positives. The precision is calculated by dividing the number of true positives by the sum of both true and false positives and weighting the results by the number of true samples per class. The accuracy and precision scores all range from zero to one with one being the best score.

As detailed in Chapter 1, it is important to be aware of possible shock impingements which may occur on a hypersonic vehicle. Shock interactions near the material of a vehicle can cause large, localized aerothermal loads which are able to degrade the structural integrity of the vehicle. In certain cases, such a compromise may lead to loss of the vehicle. Thus, the ability for each of these models to predict possible shock impingements was investigated. A model was assumed to predict a shock impingement if the peak pressure predicted along the wall was over the downstream cone. Figure 52 provides an example of this procedure for two cases, one with a shock impingement and one without. The flowfield images (Figures 52a and 52b) present the known, target response for two double cone configurations where the target response is the log-transformed pressure. Below these images are the surface pressure distributions extracted from the image. Overlaid are two vertical black dotted lines which mark the extents of the downstream cone. In the example on the left, the peak surface pressure lies within the black lines and thus occurs on the downstream cone - this configuration is considered to contain an impinging shock. In the example on the

right, the peak pressure does not occur on the second cone and thus the configuration is not considered to contain a shock impingement.

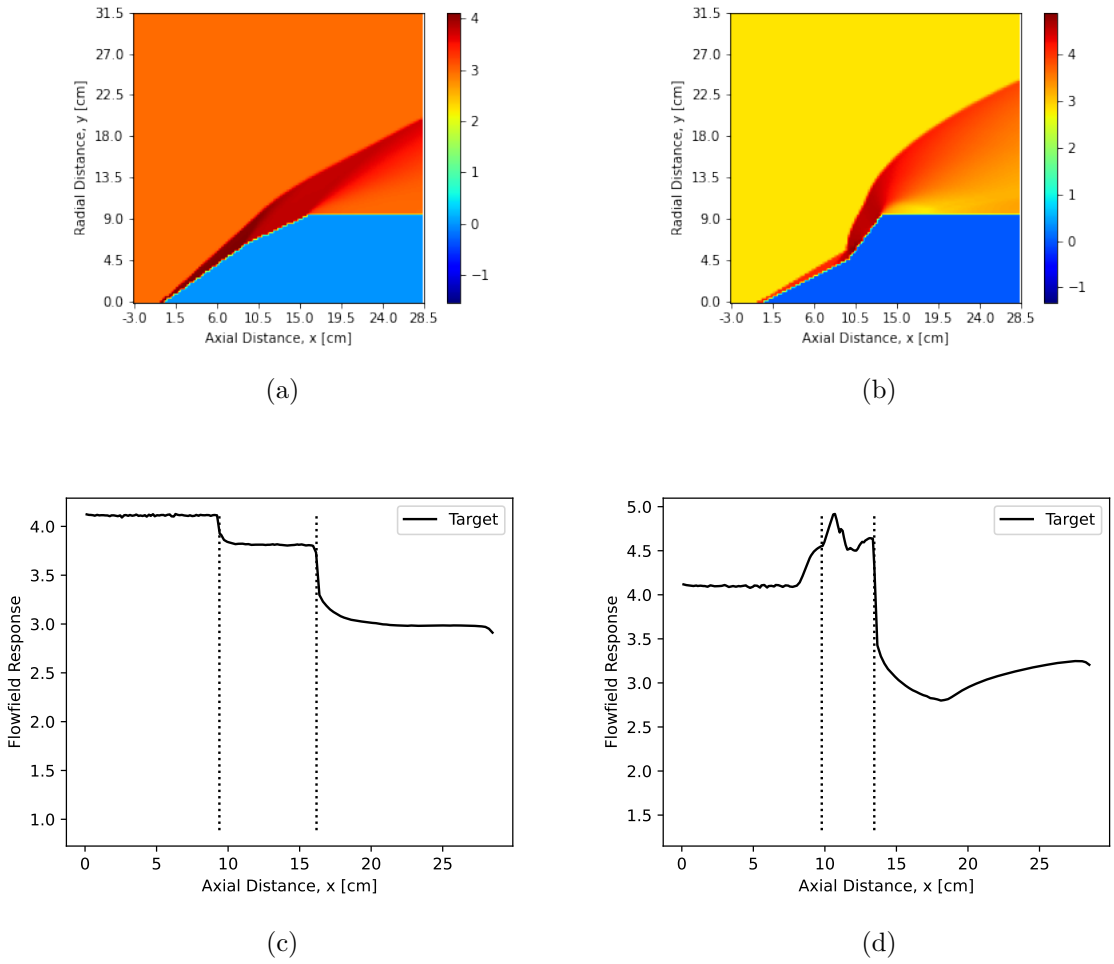


Figure 52. Example of surface pressure extraction process and resulting shock impingement classification

5.2.1 Model Performance.

The generated and tuned models were evaluated on their prediction of the surface pressure distribution along the wall. Each model generated an image of resolution 128×128 as its prediction from which the pressure values along the wall were extracted as described previously. The Mean Squared Error (MSE) between the predicted wall distribution (pulled from a full flowfield prediction) and the wall value distribution

interpolated from the simulation data was calculated and used to compare the relative performance of the models. The MSE of all surface pressure predictions (for both the training and test datasets) for all models are presented in Figures 53-56. The values plotted in blue correspond to the samples in the training set and the values plotted with the red squares correspond to those samples in the test dataset.

Figure 53 presents the MSE of the surface pressures predicted by the k-Nearest Neighbors model as compared to the target, interpolated values. Note that no blue points are present. Recall that the optimal hyperparameter configuration for the k-NN model included the distance-based weight function. Therefore, if a query design is coincident with a training point (as was the case in generating the MSE loss), then the exact response from the training dataset is returned. This results in a MSE of zero for all training cases and thus these samples cannot be visualized on the semilog plot. If the uniform weighting scheme is utilized instead, then the MSE values of the training samples lie within the same order of magnitude as the presented test results. It appears that the model prediction performance is relatively insensitive to the cone lengths or the angle of the second cone. The k-NN model performance is maximized (i.e. has minimal MSE loss) at low Mach numbers and high altitudes as well as when the angle of the first cone is large.

The performance of the Regression Kriging-inspired Gaussian Process Regression (GPR) is presented in Figure 54. These results exhibit a trend similar to the k-NN model. This is not unexpected as the k-NN model with uniform weighting scheme is utilized for the mean regression portion of the RK model. There are two distinct bands of data points in these results with one band being at extremely small values around 1×10^{-19} and the other band at relatively large numbers, around 1×10^{1}. The lower band can be explained by the fact that the GPR portion of the combined model is an interpolative model and thus the predictions of the model as a whole should be exactly

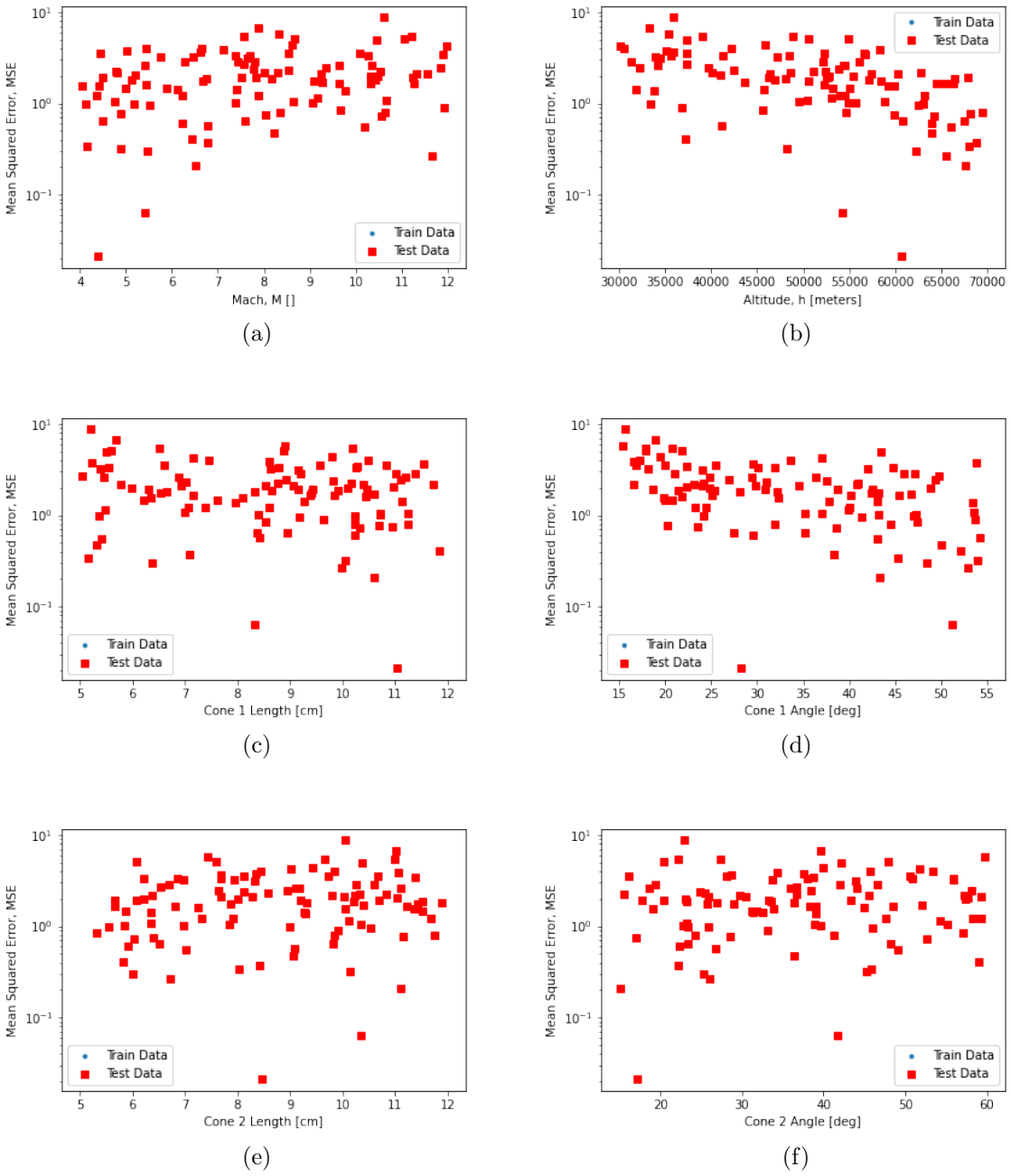


Figure 53. MSE of surface pressure prediction from k-NN model along each design variable

correct when a query point is in the training set (to within some numerical tolerance due to the algorithm implementation). The lower band is exclusively comprised of the training set. The upper band is exclusively comprised of the test set. The combined model produced predictions of the test set which have MSE loss values consistent with the k-NN model.

Figure 55 presents the MSE results of the U-Net model. The mean MSE for this model is approximately the same as the previous models, however, the range is smaller, forming tighter distributions. Similar to the previous models, the U-Net model's performance is relatively constant when the vehicle configuration is altered. This model exhibits a more subtle performance boost at lower Mach numbers and higher altitudes. The boost is not as pronounced as the previous two models, indicating that the U-Net model is better able to model flow across the design space as a whole. Across all dimensions, there is no discernible distributional shift between the U-Net model's performance on the training data and the test dataset. This indicates that overfitting the model to the training data has not occurred and the underlying distributions learned by the model generalize at a consistent performance to novel data.

Figure 56 presents the MSE results for the MS-Net model. This model performed better than the U-Net model in terms of minimizing the MSE between the predicted and target surface pressure distribution as the mean of the MSE is smaller than that of the U-Net model. The MS-Net model performance is robust to changes in all of the geometry configuration variables with no discernible trends. This model is also relatively robust to changes in the freestream Mach number as well. Similar to the U-Net, there is no distributional shift in performance from the training data to the test dataset as the MSE loss values behave similarly for both datasets. The MS-Net model did have a noticeable performance optimum near the 60km flight altitude where the

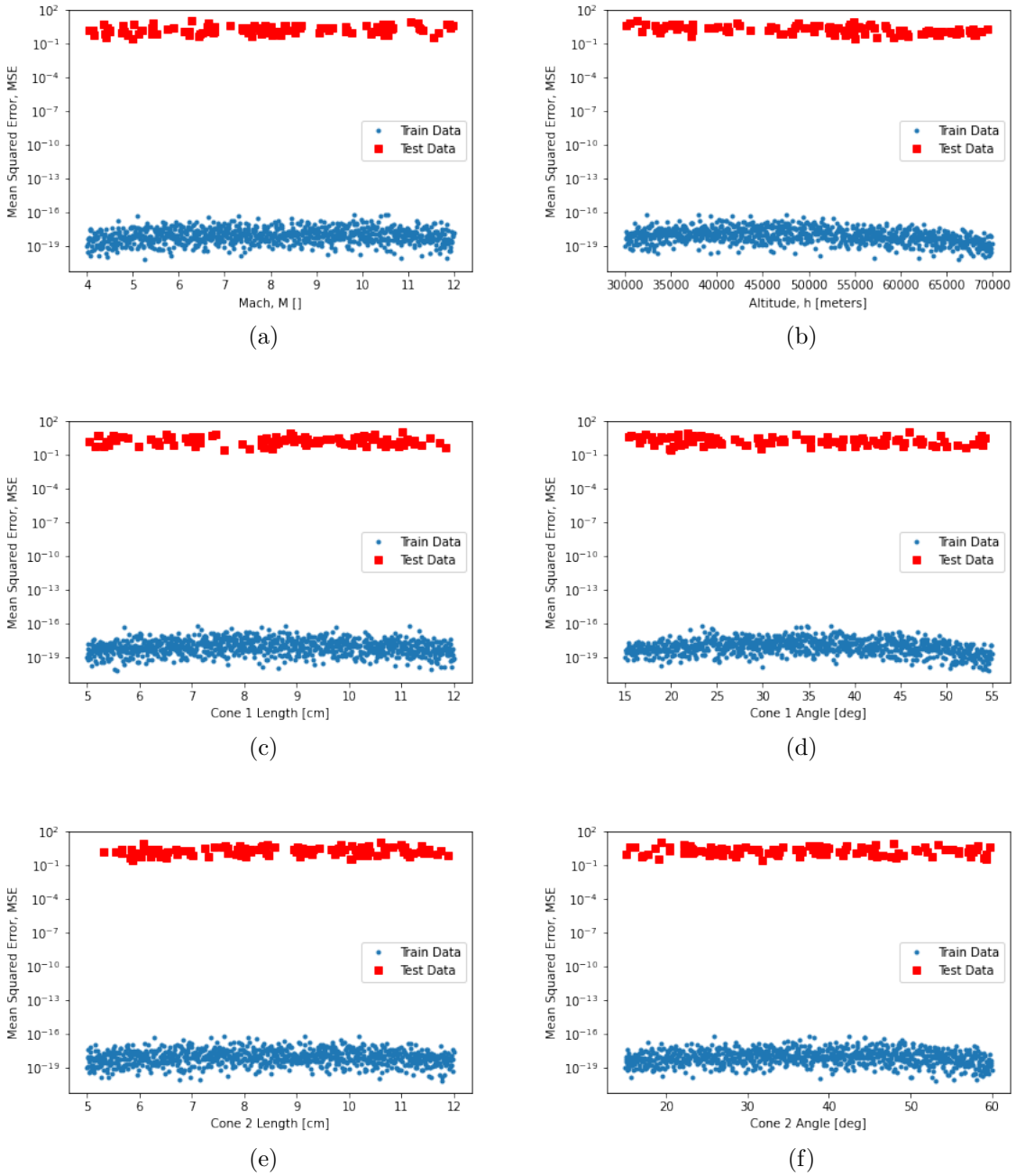


Figure 54. MSE of surface pressure prediction from RK model along each design variable

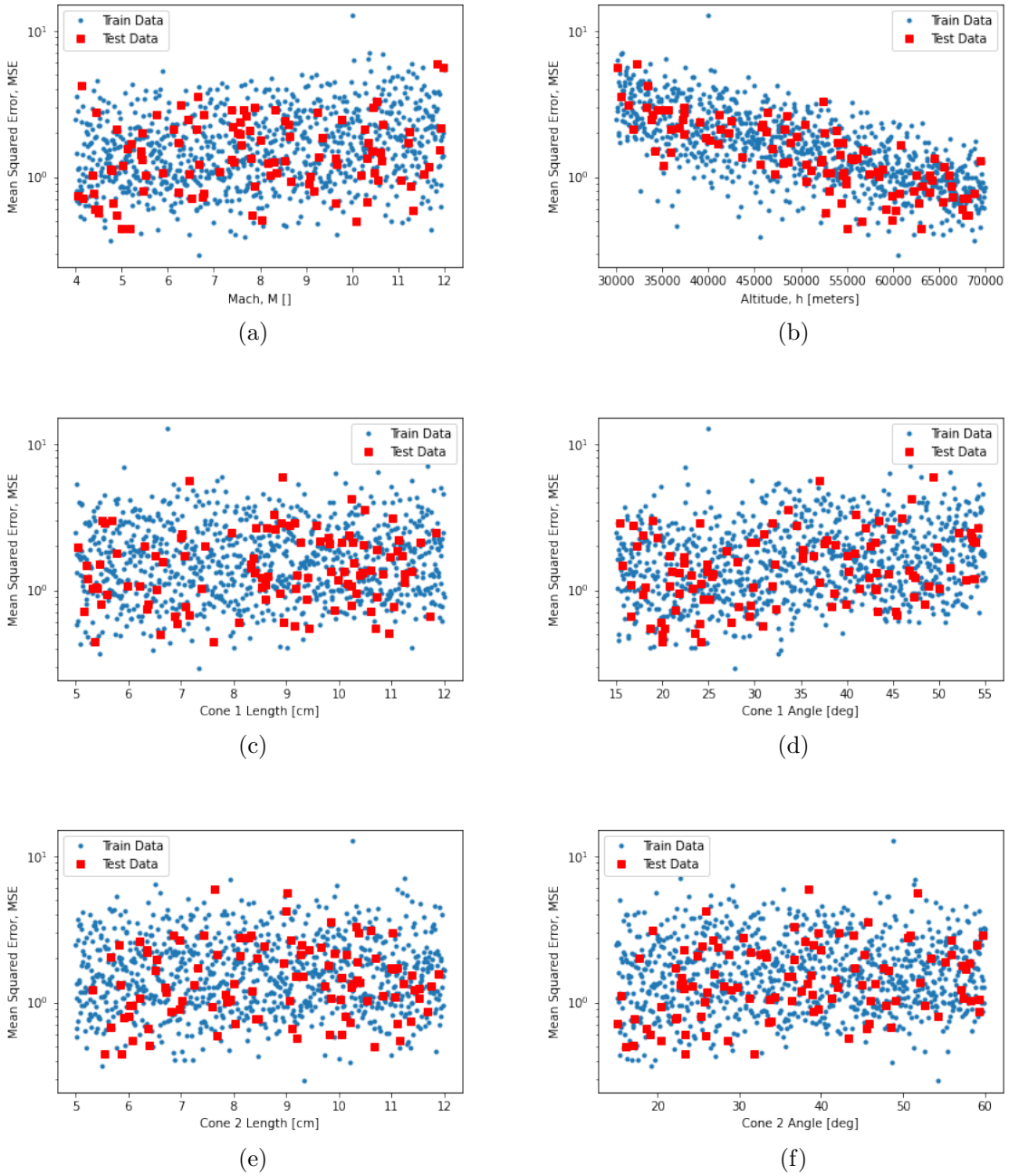


Figure 55. MSE of surface pressure prediction from U-Net model along each design variable

model's MSE bottoms out and increases near exponentially as the altitude changes away from the optimum. One explanation for the general increase in performance at high altitude and lower Mach number is magnitude of the pressures experienced at these conditions. At higher altitudes, the freestream pressure is less while the pressure ratio across similar shocks remains constant, thus lower static pressures are experienced behind the shock. Similarly, at low Mach numbers, the pressure ratio across the shock is lower, resulting in smaller static pressures downstream when compared to faster flight conditions at the same altitude. These lower pressures likely fall closer to the the mean pressure of the entire training set and are thus better captured by the models.

These results are summarized in Figure 57 where the distributions of the MSE losses of each model are compared together. The RK model has a highly skewed distribution, the reason for which is discussed previously, that obscures the distributions of the other models and thus is removed in Figure 57b for clarity. This figures shows that the k-NN model and U-Net model have similar performance when comparing the MSE between the predicted surface pressure and target surface pressure results. The U-Net's performance is more consistent throughout the design space as it has the smaller inter-quartile range (IQR). The k-NN model has less consistent performance as it has a larger IQR and numerous outliers and a lighter tail in the direction of better performance. The MS-Net has a more uniform performance throughout the design space even though it showed a very clear relationship with freestream altitude (see Figure 56b). Overall, the MS-Net mean performance is significantly better than the other models with the 75th MSE quantile being less than the mean of the other two models. The biggest drawback exhibited by the MS-Net model is that it has the widest range in performance, engulfing the other two models except for a single outlier in the k-NN performance.

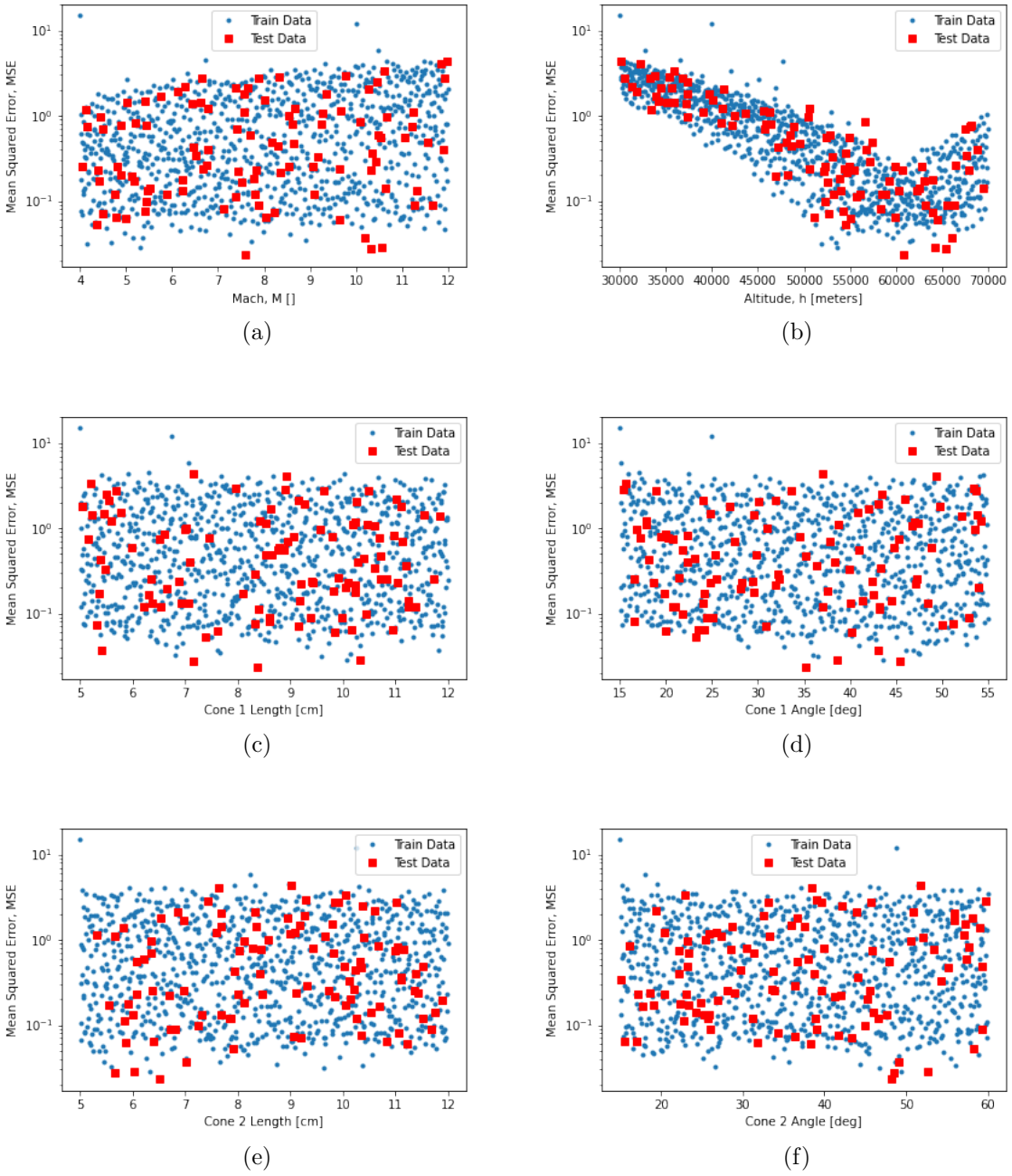
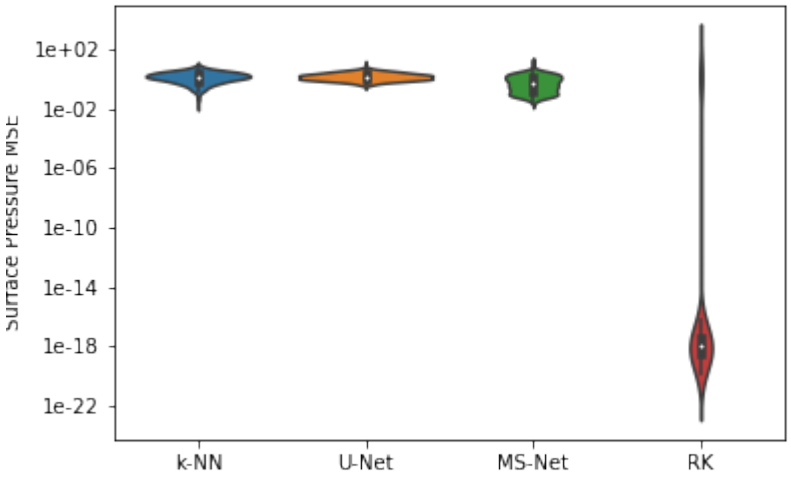
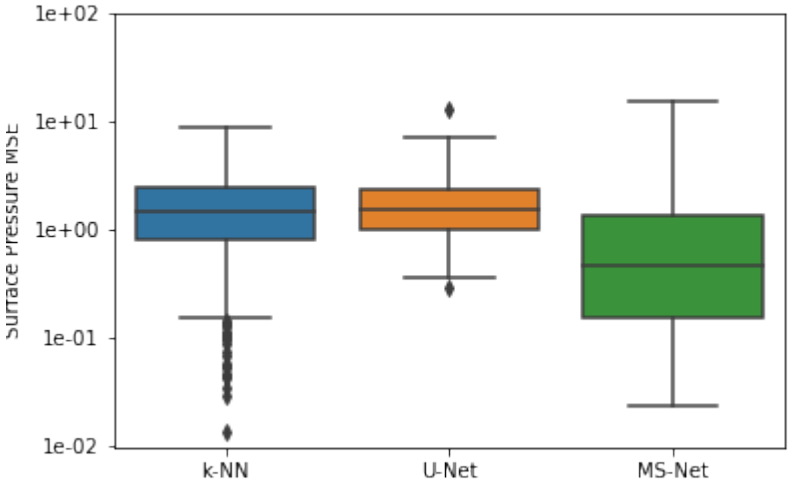


Figure 56. MSE of surface pressure prediction from MS-Net model along each design variable



(a)



(b)

Figure 57. MSE of surface pressure prediction distributions for each model type compared side-by-side. The RK results have been removed in (b) for clarity.

5.2.2 Categorical Metrics.

The procedure for extracting the shock impingement classification from a response was applied to the predicted responses of all samples for all four models as well as the target responses. The raw classification data utilizing the entire dataset is presented for each model in Table 3 and the same results for only the test dataset are presented in Table 4. In each sub-table, the classifications for the target responses are on the vertical axis and the classifications from the predicted responses are along the horizontal axis. The marginal totals are included as well. The accuracy and precision metrics are summarized in Table 5.

As expected, the interpolative models (k-NN and RK) have perfect values for accuracy and precision for the training dataset (Table 5a). The CNN-based models do not perform nearly as well as the previous two models on the full dataset with much lower accuracy and precision scores. The U-Net architecture outperforms the MS-Net model by approximately ten percentage points in both categories. The performance of all models falls when considering performance on the test data (except for the U-Net for which precision slightly improves; this is about a 1% fluctuation). Of note is that the first two models experience significant performance drops. The RK model is reduced to approximately the accuracy of a coin flip and the k-NN model performance drops to approximately that of the U-Net. The CNN-based models experience relatively little performance degradation between the training and test datasets. The MS-Net classification performance drops on the order of ten percent while the U-Net experiences a change on the order of only 1%.

5.3 Prediction Samples

This section presents the worst and median case for the extracted surface pressure distributions based on the MSE loss metric. Figure 58 presents the extracted surface

Table 3. Confusion matrices for predictions for shock impingement classifications for all models on the entire dataset

(a) k-NN model results

Target/ Predicted	No Impingement	Impingement	Totals
No Impingement	509	0	509
Impingement	46	469	515
Totals	555	469	1024

(b) RK model results

Target/ Predicted	No Impingement	Impingement	Totals
No Impingement	504	5	509
Impingement	38	477	515
Totals	542	482	1024

(c) U-Net model results

Target/ Predicted	No Impingement	Impingement	Totals
No Impingement	369	140	509
Impingement	108	407	515
Totals	477	547	1024

(d) MS-Net model results

Target/ Predicted	No Impingement	Impingement	Totals
No Impingement	445	64	509
Impingement	381	184	515
Totals	776	248	1024

Table 4. Confusion matrices of predictions for shock impingement classification for all models on the test dataset

(a) k-NN model results

Target/ Predicted	No Impingement	Impingement	Totals
No Impingement	55	0	55
Impingement	46	2	48
Totals	101	2	103

(b) RK model results

Target/ Predicted	No Impingement	Impingement	Totals
No Impingement	50	5	55
Impingement	38	10	48
Totals	88	15	103

(c) U-Net model results

Target/ Predicted	No Impingement	Impingement	Totals
No Impingement	36	19	55
Impingement	7	41	48
Totals	43	60	103

(d) MS-Net model results

Target/ Predicted	No Impingement	Impingement	Totals
No Impingement	46	9	55
Impingement	34	14	48
Totals	80	23	103

Table 5. Accuracy and precision of the model classifications

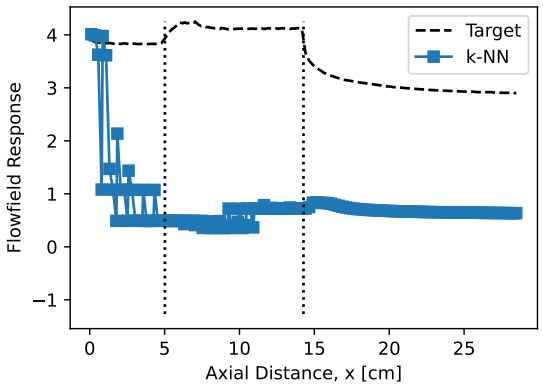
(a) For only training samples

Model	Accuracy	Precision
k-NN	1.000	1.000
RK	1.000	1.000
U-Net	0.759	0.759
MS-Net	0.618	0.666

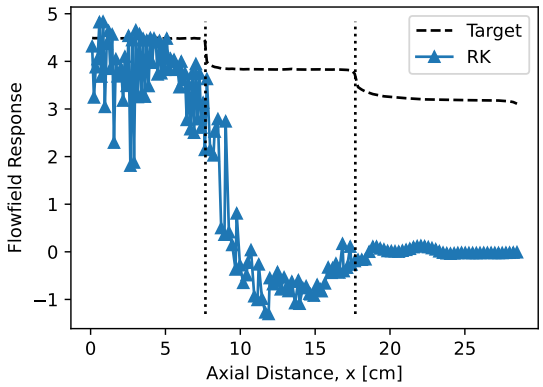
(b) For only test samples

Model	Accuracy	Precision
k-NN	0.718	0.816
RK	0.515	0.508
U-Net	0.748	0.766
MS-Net	0.583	0.591

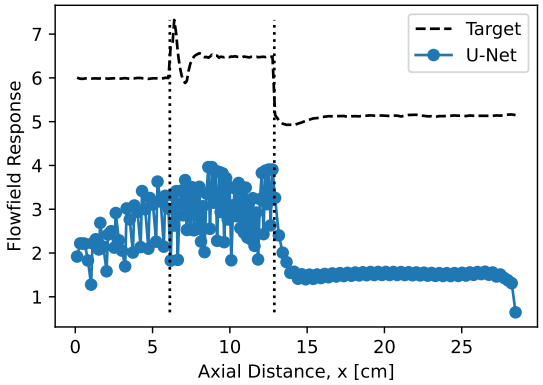
pressure distribution for the worst predictions from all of the models. These results display the range of differences between the target distribution and the predicted distributions. For these worst-performance sample, the predicted surface pressures can be between three and six units below the target distribution. If these results were to be transformed back into the static pressure space, then the predicted pressure would be between three and six orders of magnitude lower in some locations. Figures 59 and 60 present two extracted wall results whose performance is near the median MSE loss for the model's surface pressure predictions. Again, the general trend appears to be that the predicted wall values generally sit about one unit lower than the target distribution. In the case of the MS-Net, this discrepancy is reduced to be only about half of a unit. Figure 61 presents the samples with the best wall prediction results. Note that the results displayed for the k-NN and RK models were selected from the test data as these models are interpolative and return the target solution for all training samples, thus all training sample produce optimal results. The best k-NN and MS-Net results oscillate about the target wall value distribution with low amplitude. The minimal oscillations about the target response in these results are desirable for use in fast reconstruction and prediction of the flowfield. On the other hand, the RK and U-Net models experience much larger amplitude oscillations in their respective best performing cases. The RK solution does oscillate around the target distribution, however, the target distribution does not appear to be well aligned with the local mean of the oscillations. The U-Net model predicts oscillations of similar amplitude, but the mean of the distribution is far from the target wall distribution. Additionally, both the RK and U-Net models have relatively large deltas between the target and predicted distribution downstream of the second cone. This issue is much less pronounced in the k-NN and MS-Net models' best cases.



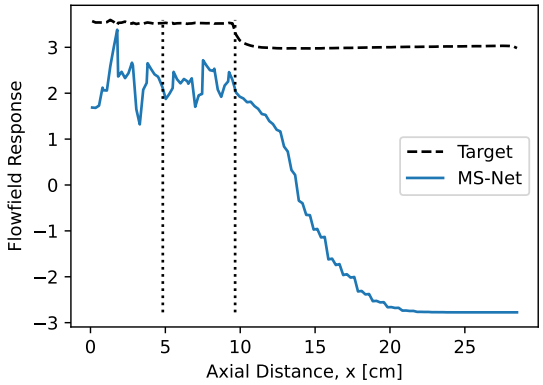
(a) Worst performing surface pressure distribution for the k-NN model



(b) Worst performing surface pressure distribution for the RK model

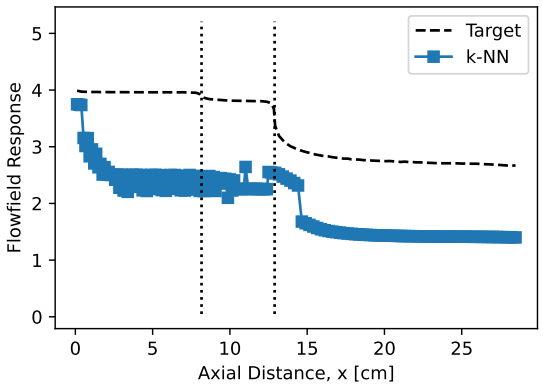


(c) Worst performing surface pressure distribution for the U-Net model

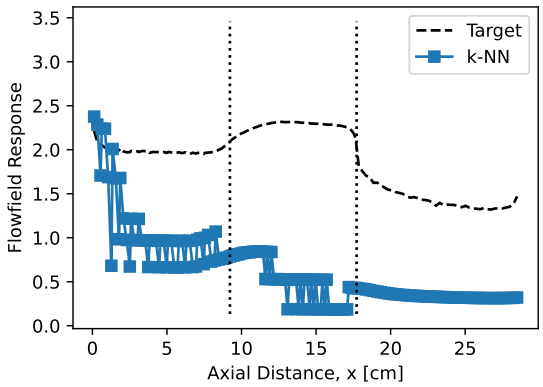


(d) Worst performing surface pressure distribution for the MS-Net model

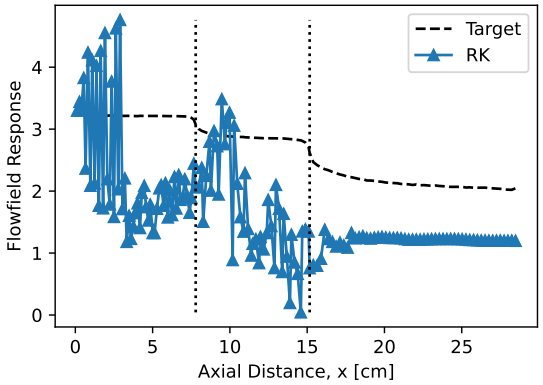
Figure 58. The cases which produce the worst performing wall predictions (on MSE) for each model



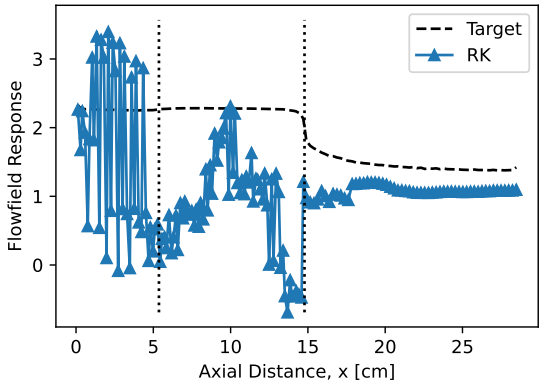
(a) Median performance surface pressure distribution for the k-NN model



(b) Median performance surface pressure distribution for the k-NN model

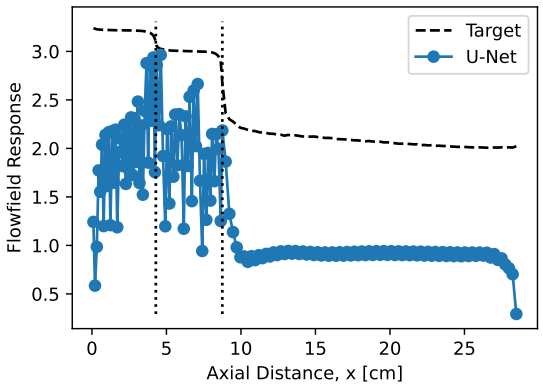


(c) Median performance surface pressure distribution for the RK model

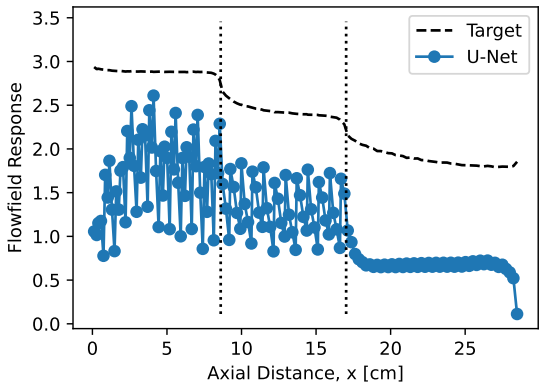


(d) Median performance surface pressure distribution for the RK model

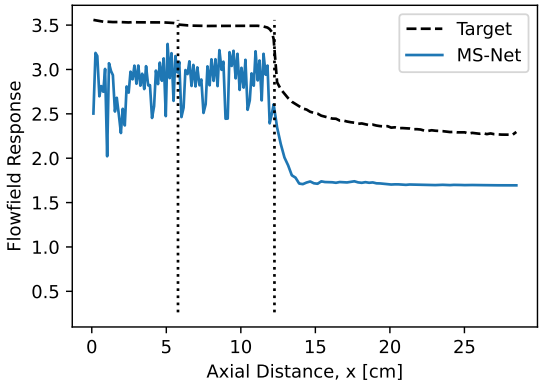
Figure 59. The cases which produce a median performance wall prediction (on MSE) for the k-NN and RK models



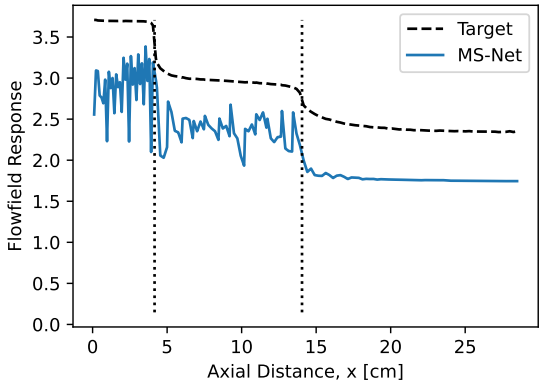
(a) Median performance surface pressure distribution for the U-Net model



(b) Median performance surface pressure distribution for the U-Net model

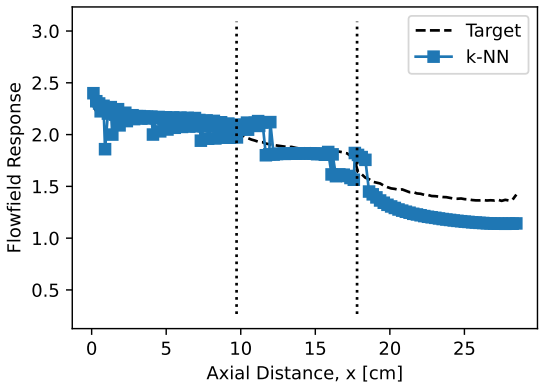


(c) Median performance surface pressure distribution for the MS-Net model

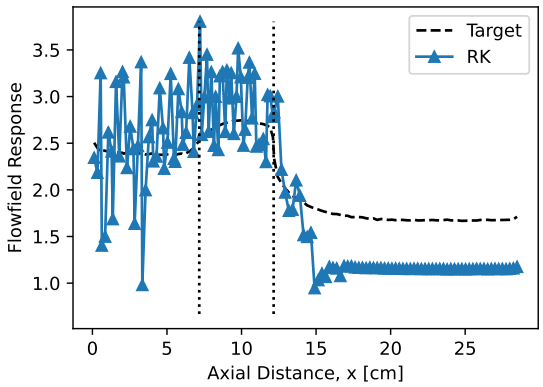


(d) Median performance surface pressure distribution for the MS-Net model

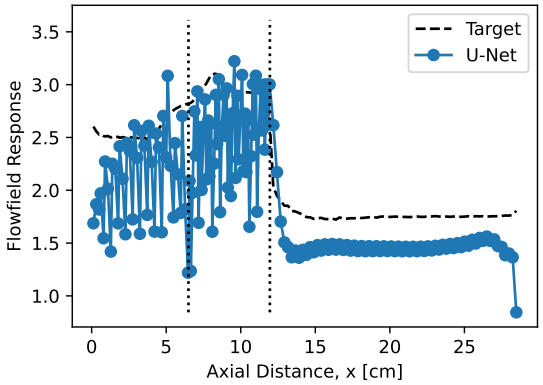
Figure 60. The cases which produce a median performance wall prediction (on MSE) for the U-Net and MS-Net models



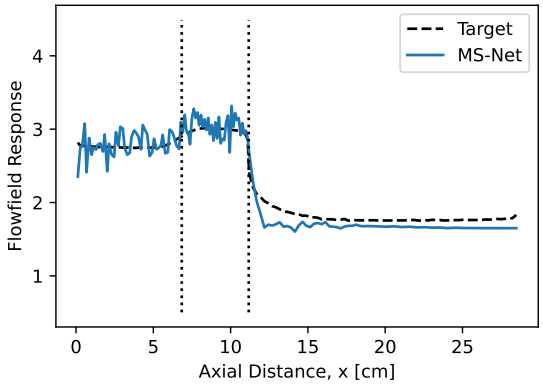
(a) Best performing surface pressure distribution for the k-NN model



(b) Best performing surface pressure distribution for the RK model



(c) Best performing surface pressure distribution for the U-Net model



(d) Best performing surface pressure distribution for the MS-Net model

Figure 61. The cases which produce the best performing wall predictions (on MSE) for each model

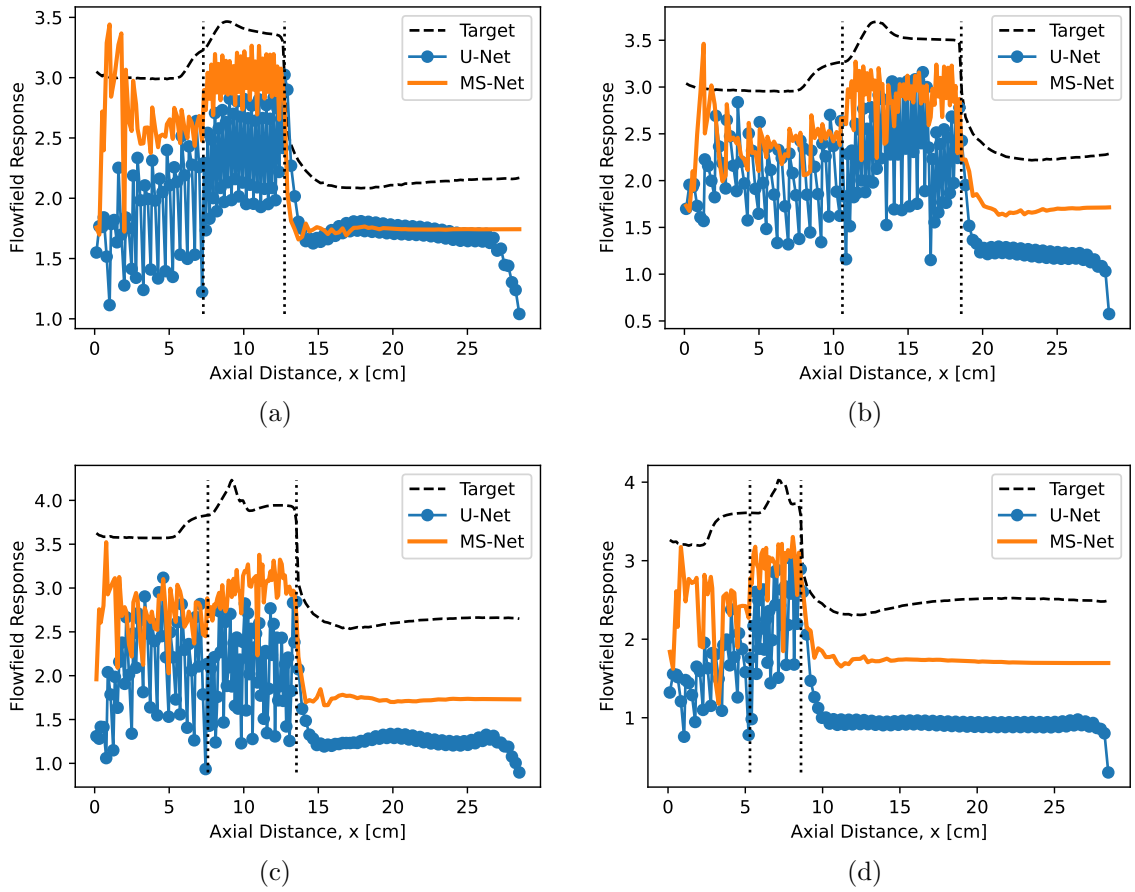


Figure 62. The target and predicted flow field response for a sample of configurations which contain shock impingements. These examples are from the training dataset, thus the k-NN and RK predictions are exactly correct and are omitted for clarity.

Figure 62 presents a selection of training samples which contain shock impingements. The actual surface pressure distributions for these cases contain large spikes in static pressure. However, the large spike is lessened by the interpolation procedure to the Cartesian grid. Then the additional log transformation would further lessen the magnitude of the spike. Because of this, the NN models do not appear to have as strong of a pressure to accurately map the small local shock impingement zones.

5.4 Model Comparison

The time required to generate one and multiple flowfield predicitions using each method was computed and the results are presented in Table 6. On average, over

the entire dataset, one RANS solution took under 300 seconds to compute (using a 20 logical core machine). This does not take into account the time required to generate the corresponding, unique mesh. This time scales linearly, so if simulations were requested for a batch of 64 double cone configurations, the total time required would be approximately 5.3 hours to complete the computations. It was found that using one of the models discussed in this research, the time required to generate a prediction for a configuration of interest is approximately five orders of magnitude quicker than performing the RANS simulation. Additionally, generating batches of predictions was also incredibly quick.

Table 6. Time required to predict the pressure field for both one query configuration and a batch of 64 queries.

Prediction Method	1 Prediction (sec)	64 Predictions (sec)
RANS	294	18.8×10^{3}
k-NN	358×10^{-6}	14×10^{-3}
GPR	5.3×10^{-3}	122×10^{-3}
U-Net	3.05×10^{-3}	154×10^{-3}
MS-Net	7.11×10^{-3}	208×10^{-3}

5.4.1 Extended Applicability.

Due to the construction of fully convolutional neural networks such as the MS-Net and U-Net (discussed in Chapter 2), the models are agnostic to the absolute size of the input data or flow to be queried. The only requirement is that the relevant scales of features in the input are congruent with the features which the models were trained on. Figure 63 demonstrates how the input shape does not need to be constant and that the models will make predictions based off of the local patterns in the input. On the left, the geometry masks of two distinct double cone configurations are presented. In the center, both vehicles have been placed in close proximity to each other and the distance metric is computed for this formation flight configuration. On

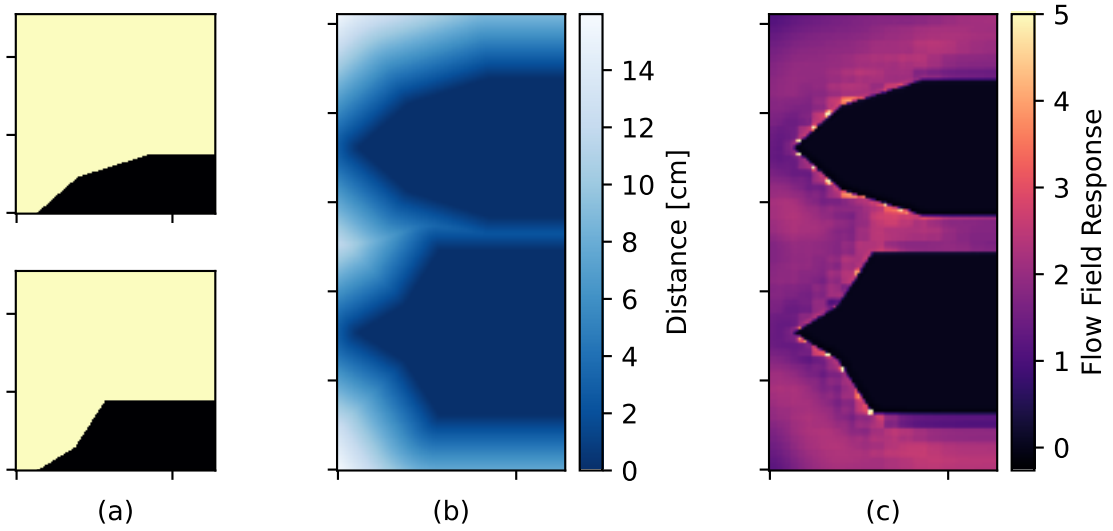


Figure 63. Example of CNN-based model applicability to more general geometries. (a) Two separate double cone configurations. (b) Minimum distances between prediction points and the two configurations placed in close proximity to each other. (c) MS-Net prediction of flowfield response over a domain shape and content (two separate vehicles) which is not present in the training data.

the right is the prediction for this flight condition predicted by the MS-Net model. It is important to note that the MS-Net model has not been modified in any way in order to accept this novel input size. There are no results in the training data which include either an image of these dimensions or an example of more than one vehicle. The predictions made for this formation of double cones use data which is exclusively derived from a simpler class of computational problems (namely the flow around only one double cone configuration). The NN model was able to provide a prediction which is consistent with those where only one double cone is present. The novel geometries do not have to explicitly exist in the training data as in the example presented in Figure 64. Because these geometries do not exist in the training set, the relationships internalized by the model from the training data are used to construct these predictions. Additionally, the size of the prediction domain can be of arbitrary size and include arbitrary geometry. The prediction quality on these novel samples could be increased by performing additional training using new, relevant data.

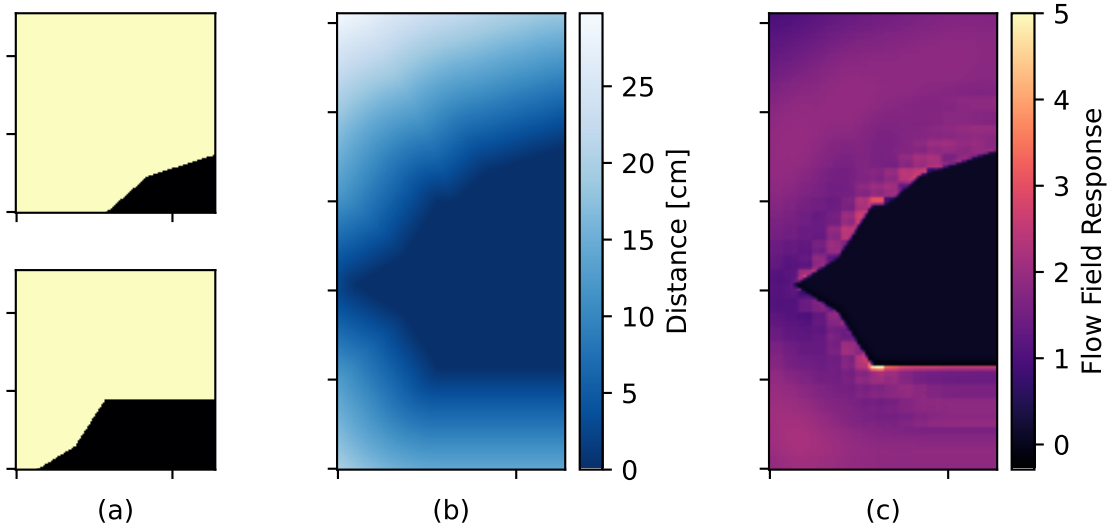


Figure 64. Example of CNN-based model applicability to more general geometries. (a) Two separate double cone configurations with the first translated downstream. (b) Minimum distances between prediction points with the two configurations fused to generate a novel geometry. (c) MS-Net prediction of flowfield response over a domain shape and content which is not present in the training data.

VI. Discussion

This research investigated the ability of Machine Learning models to learn and provide rapid predictions of the flow field around a hypersonic system. A brief history of and motivation for the development of hypersonic systems was presented. A strong motivation for the development of hypersonic systems is to maintain superior long-range strike capabilities which are difficult for adversaries to detect and destroy and to enable response to adversaries' high-speed weapons. The development of such novel vehicles necessitates design searches. Frequent full-scale flight tests or wind tunnel testing is not currently practical in the field of hypersonic development, thus much of the design process utilizes computational fluid dynamics simulations to understand vehicle responses. Simulations which can predict the flow around a hypersonic vehicle with great spacial resolution and physical accuracy are possible, but are computationally expensive. Thus, performing such simulations for each iteration of a candidate vehicle design is prohibitively expensive, necessitating the need for computationally cheap simulations which can provide insight into the resulting flow field and drive the design process to focus on more promising vehicle configurations. One feature of hypersonic flows which can pose an existential threat to a vehicle are shock waves. In the hypersonic flight regime, shock waves can be pushed towards the surface, closely following the surface of the vehicle. If this shock wave interacts with a downstream portion of the vehicle, the resulting aerothermal loads can cause damage to the vehicle. If a candidate vehicle configuration may cause a shock impingement to occur, that vehicle will likely not be as desirable as a configuration without shock impingements. Thus one objective of the developed ML models is to predict if a shock impingement will occur for a given vehicle configuration.

The models utilized were an implementation of k-Nearest Neighbors, a regression kriging, U-Network, and the Multiscale Network (MS-Net). The latter two are based

on convolutional neural networks and utilize gradient descent algorithms in their training. These models were trained to predict the static pressure on a Cartesian grid (in the form of a 128×128 image). It was found that the log base-ten of the static pressure produced better training results. The k-NN and RK models were trained to explicitly minimize the Mean Squared Error between their predictions and the target responses (the CNN-based models utilized a similar loss function as well). The raw static pressure distributions contained very large ranges both within each sample and between samples on the order of five order of magnitude. The static pressure distributions also had large fluctuations in spatially concentrated regions (such as those near a shocks and their interactions). All four models utilize some form of superposition to generate predictions and the large ranges in fluctuations in the dataset were not able to be properly fit or adequately encoded using these models. Utilizing the log transform reduced the range of the data into a more well behaved distribution with a much smaller range of response values within and between samples. This transformed dataset enabled better performance for the models.

The vehicle in this research was a parameterized sharp leading edge double cone. The lengths and half-angle magnitudes of both constituent cones are allowed to vary independently to generate the vehicle configurations. The flight conditions of each configuration were defined by the freestream altitude and Mach number for a total of six design variables. The dataset is comprised of 1024 samples generated using a Sobol sequence to ensure a low-discrepancy filling of the design space. The mesh for each configuration was generated and subsequently simulated using a RANS solver to produce the static pressure distribution within the flowfield and along the double cone surface. These distributions were utilized to generate the 128×128 images which comprise the dataset. Additionally, the minimum distance from the wall and the freestream dynamic pressure were calculated to generate the known features for

the dataset. The k-NN and RK models utilized the length six design variable vectors to predict the flowfield while the CNN-based models, U-Net and MS-Net, utilized the distance and dynamic pressure image inputs to generate their predictions.

Hyperparameter optimization was performed to find the construction settings for each model which produced the lowest loss. For the k-NN model, it was found that combining the eight closest samples in the training set by Manhattan distance produced the best results. For this model, utilizing fewer neighbors allows for the averaged shocks to be more discernible. The boundaries of these shocks become less sharp as they are combined with additional flowfield samples from the training dataset.

It was found that these models are not capable of accurately predicting the magnitude of the surface pressure along the hypersonic double cone. The log transformed fit can deviate from the target response by about one unit in the median case. Translating the predictions from the log space to the original static pressure space would result in large error values as the predictions could deviate from the target distribution by about one order of magnitude. Such consistently large deviations severely limit the applicability of using these models for accurately predicting the static pressure magnitude around the vehicle. These results indicate that these models, in the current configuration, are not an appropriate model for the pressure results obtained from the RANS simulations.

The second use case investigated for these models was for their use as a computationally cheap method to predict if shock impingement is probable for the design configuration. It was found that the interpolation models (k-NN and RK) experienced a steep performance drop between the training and test datasets. Both of these models had no modeling uncertainty, but had high predictive uncertainty. This difference indicates that these models were unsuccessful in capturing the underlying physics in the problem. On the other hand, the NN models experienced little change

in uncertainty between the modeling and predictive problem in exchange for a greater base uncertainty. In contrast to the first two models, the relatively small increase in uncertainty between the two problems indicate that the NN models have more robustly understood the underlying flow physics. While the modeling of the actual physics does not occur, the reduced order model enables the NN models to generate predictions of comparable quality throughout the design space. The U-Net and MS-Net experienced drops in their classification accuracy of 1% and 5%, respectively between the modeling and prediction problems. Of the U-Net predictions, the true positive and true negative rates were 68.3% and 83.7%, respectively. For the MS-Net, the true positive and true negative rates were 60.9%, and 57.5%, respectively. Therefore, for a novel double cone configuration, these models could be queried for a prediction of whether or not an impinging shock is likely to form. Despite their limited applicability in predicting the target pressure response, these models could be used in conjunction with other metrics to better focus computational resources on more promising candidates designs.

All models produced surface pressure predictions which exhibited large oscillations in value between adjacent points. Similar oscillation is not present in the training data, thus it must arise from the internal operations of the models. None of the models are trained with a loss function that accounts for oscillations in surface pressure. The k-NN and RK models are interpolative models and similar double cone configurations tend to have similar, but not identical, surface geometries. The pressure inside of the geometry were assigned a response value of zero while the pixels corresponding to the surface were assigned some non-zero value. It is likely that for samples which are near each other in the design space, there are both pixels with zero and non-zero values at the same point in different nearby training points. Consider a single pixel which encompasses the local double cone surface for a queried configuration. The value

of that specific pixel is influenced by the value of the same pixel from neighboring samples in the training set. If some of these pixels have a value of zero and others are non-zero, there will exist competing influences for that pixel. Because all of the double cone configurations are slightly different, the exact number of zero and non-zero influences are different for each pixel. In the pixels with more or stronger zero valued influences, the predicted value is likely to be smaller than if the non-zero influences are stronger. This relative location of the geometry surface is less predictable where the double cone geometry physically changes the most i.e. over the first and second cones. The horizontal face downstream of both cones always passes through the same row of pixels. There is never a situation where some pixels are zero and others in the line are not, it is always all of the pixels or none of the pixels. In the predictions of all four models, the section over the horizontal face has the most stable value. If a smoother prediction is desired, then the jitter present in pixels near the surface (where sometimes a particular pixel has a value of zero and sometimes it has a non-zero value in the responses of nearby samples) must be reduced. One avenue of addressing this would be to assign pixel values which are slightly inside the geometry the value of the surface pressure instead of only doing so for the pixels which the surface passes through. This would serve to reduce the jitter in the pixels of interest and would likely result in smoother surface pressure predictions as the relevant surface pressure data from adjacent samples would be more readily available.

The U-Net model produced results with some irregularities not present in the predictions from the other models. These include the small-scale grid patters which emerge in the predicted flowfields and the larger-scale bands which tend to appear near the borders of the same. These effects are not present in the MS-Net or the other model predictions. Both of the NN models were trained until the validation error stopped improving. This was a measure used to prevent overfitting the models to

the training data and should not have an effect on the structure of the kernels which could result in the observed U-Net artifacts. Both the U-Net and MS-Net utilize upsampling and padding, however similar artifacts are not present in MS-Net predictions. The two parameters which are appreciably different in the two models are the kernel initialization method and the convolutional padding method. The MS-Net utilizes the standard random initialization scheme for the kernel weights while the U-Net initializes the weights with normally distributed values. This should have no functional effect on the predictions of the U-Net as these weights are modified during model training. The U-Net architecture utilizes zero padding in its convolutional layers. This augments the input to a layer such that the convolution operation produces an output of the same size as the original input to the layer. With zero padding, the augmentation is completed by adding rows/columns of zeros around the input. The MS-Net architecture utilizes "same" padding where the values at the edges of the input are repeated in order to augment the input. This formulation mimics the zero-gradient boundary condition enforced by the solver at the far field flow boundaries. The zero padding introduces undue changes in value near at boundaries. This is likely the mechanism which promotes the formation of the unexpected bands around the output images.

The neural network models demonstrated the ability to capture some flow physics and generalize those relationships to the prediction problem. One attractive characteristic of fully convolutional networks (such as the U-Net and MS-Net) is that they can operate on arbitrarily sized inputs. Additionally, the use of the NN models can generate predictions without the need for grid convergence studies or simulation convergence studies as opposed to traditional, mesh-based CFD techniques. The requirements for the NN models (in their current implementation) are that the vehicle configuration geometry has a definition such that the points of the Cartesian grid can

be determined to be inside or outside the vehicle and the relevant input features and known conditions can be calculated and applied to the grid. If these two conditions can be satisfied, then these NN models can be used to generate rapid predictions of the flowfield response. The training data for these models need only consist of examples of expected flow phenomena. Thus, a training set comprised of relatively small and computationally cheap examples can be used to generate predictions about more complex shapes. If the prediction accuracy and performance of these models can be improved, they have the potential to become powerful tools for rapid low-fidelity design evaluation.

These models would be used relatively early in the design cycle. In their current state, it is not feasible to use them to outright dismiss or accept a candidate vehicle configuration. However, these models can be utilized to generate Bayesian priors for candidate configurations. The NN-based models have consistently predicted the overall trend of the static pressure field, especially along the walls. In their current state, the model predictions can serve to produce data-based prior distributions for the flow field and/or surface pressure values. In combination with other low-fidelity flow field estimation procedures, these priors could highlight configurations which warrant additional investigation using higher-fidelity methods.

6.1 Future Work

Future work for this research should include investigating the applicability of transfer learning. The NN-based models have the potential to be trained on large samples of computationally cheap samples and subsequently adapted to a relevant problem using a few higher-fidelity simulations in the specific design space. The effect of using arbitrarily sized inputs during evaluation could also be investigated as the patterns learned by the models are translationally invariant. These models can be

expanded to operate on three-dimensional data with very little modification, as such their effectiveness on 3D flowfields could also be investigated. Additionally, because of the fast evaluation times of these models, they could be integrated as a low or medium fidelity model into a larger meta-model such as the embedded emulators in [19].

In the implementation and training of the U-Net and MS-Net, this work utilized the loss functions described by the respective works, [24] and [25]. Neither of these works attempted to reconstruct highly compressible fluid flow field. In this research, the accurate reconstruction of structures specific to the realm of highly compressible flows (e.g. shock waves) is of interest. In theory, the greedy nature of the gradient descent training algorithm would seek to reduce the largest differences between the predicted and target responses first. The theorized result of this process was to greedily match the peak response values first and subsequently improve the predictions within the bulk flow. This was not the realized result, the peak responses were not very large compared to the rest of the response values in a sample and constituted a significantly smaller proportion of the response than the freestream values. Thus, the loss reduction offered by accurately predicting the bulk flow was greater than that offered by accurate prediction of the peak responses, resulting in the former driving the direction of training. An additional effect of this training progression was the diffusion of the shock wave. In future work, a loss function could be developed which drives training to better meet the objective of accurately reconstructing shocks and their interactions. Such a function could penalize the diffusion of shock waves to drive reconstructions to generate sharp gradients and reduce diffusion. Additionally, the overall physical accuracy of the flow field reconstruction may be increased by utilizing residuals calculated using the governing equations of fluid flow. With minor modifications to the inputs of the models, the residuals of the Navier Stokes equations can

be calculated for a reconstructed flow field on the Cartesian grid. As these residuals can be used to inform the gradient descent algorithm and drive reconstructed and predicted flow fields to be more accurate with respect to the flow physics.

Bibliography

[1] E. C. C. Team, "Air superiority 2030 flight plan," *R.(Washington DC US Air Force, 2016)*, 2016.

[2] J. D. Anderson, "Hypersonic and high-temperature gas dynamics second edition," *American Institute of Aeronautics and Astronautics*, 2006.

[3] H. Cheng, R. Grandhi, W. Hankey, and P. Belcher, "Takeoff and landing analysis methodology for an airbreathing space booster," *Acta Astronautica*, vol. 29, no. 5, pp. 325–332, 1993.

[4] H. Kauffman, R. Grandhi, W. Hankey, and P. Belcher, "Control strategies for space boosters using air collection systems," *Journal of Spacecraft and Rockets*, vol. 31, no. 2, pp. 243–248, 1994.

[5] ——, "Improved airbreathing launch vehicle performance with the use of rocket propulsion," *Journal of Spacecraft and Rockets*, vol. 28, no. 2, pp. 172–178, 1991.

[6] N. Venugopal, R. Grandhi, W. Hankey, and P. Belcher, "Combined energy management and calculus of variations approach for optimizing hypersonic vehicle trajectories," *Computing Systems in Engineering*, vol. 1, no. 2-4, pp. 591–600, 1990.

[7] J. Tiley, R. V. Grandhi, W. L. Hankey, and P. Belcher, "Optimization of tactical hypersonic intercept performance features," *Engineering Optimization*, vol. 20, no. 1, pp. 1–19, 1992.

[8] H. Kauffman, R. Grandhi, W. Hankey, and P. Belcher, "Optimum design of transitions in climb/cruise/descent for hypersonic cruise vehicles," *Engineering Optimization*, vol. 19, no. 2, pp. 153–170, 1992.

[9] R. H. Speier, G. Nacouzi, C. Lee, and R. M. Moore, *Hypersonic missile nonproliferation: hindering the spread of a new class of weapons*. Rand Corporation, 2017.

[10] D. Knight and M. R. Youssefi, "Assessment of cfd capability for hypersonic shock wave laminar boundary layer interactions," *Aerospace*, vol. 4, p. 25, 2 Apr. 2017, ISSN: 2226-4310. DOI: 10.3390/aerospace4020025. [Online]. Available: https://www.mdpi.com/2226-4310/4/2/25.

[11] J. D. Watts, "Flight experience with shock impingement and interference heating on the x-15-2 research airplane," *NASA TM X-1669*, 1968.

[12] C. A. I. Board, "Report volume 1, august 2003," *US Government Printing Office, Washington*, 2003.

[13] D. C. Gochenaur, D. M. Baier, and A. J. Zakrajsek, "Validation of conceptual hypersonic design tool framework using x-15 case study," in *AIAA Scitech 2020 Forum*, 2020, p. 0321.

[14] K. R. Quinlan, J. Movva, E. V. Stein, and A. Kupresanin, "Leveraging multi-fidelity aerodynamic databasing to efficiently represent a hypersonic design space," in *ASCEND 2021*, 2021, p. 4245.

[15] C. C. Fischer and R. V. Grandhi, "Utilizing an adjustment factor to scale between multiple fidelities within a design process: A stepping stone to dialable fidelity design," in *16th AIAA Non-Deterministic Approaches Conference*, 2014, p. 1011.

[16] ——, "A surrogate-based adjustment factor approach to multi-fidelity design optimization," in *17th AIAA Non-Deterministic Approaches Conference*, 2015, p. 1375.

[17] ——, "Multi-fidelity design optimization via low-fidelity correction technique," in *17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2016, p. 4293.

[18] C. C. Fischer, R. V. Grandhi, and P. S. Beran, "Bayesian-enhanced low-fidelity correction approach to multifidelity aerospace design," *AIAA Journal*, vol. 56, no. 8, pp. 3295–3306, 2018.

[19] A. Beachy, H. Bae, I. Boyd, and R. Grandhi, "Emulator embedded neural networks for multi-fidelity conceptual design exploration of hypersonic vehicles," *Structural and Multidisciplinary Optimization*, vol. 64, no. 5, pp. 2999–3016, 2021.

[20] D. Zettl, E. Dreyer, B. Grier, J. J. McNamara, and C. L. Pasiliao, "Rapid steady-state pressure prediction for ultra high-speed vehicles," in *15th Dynamics Specialists Conference*, 2016, p. 1323.

[21] E. Dreyer, R. Klock, B. Grier, J. J. McNamara, and C. E. Cesnik, "Multi-discipline modeling of complete hypersonic vehicles using cfd surrogates," in *58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2017, p. 0182.

[22] E. R. Dreyer, B. J. Grier, and J. J. McNamara, "Towards characterization of relevant fidelity modeling of loads for maneuvering hypersonic vehicles," in *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2018, p. 1207.

[23] E. R. Dreyer, B. J. Grier, J. J. McNamara, and B. C. Orr, "Rapid steady-state hypersonic aerothermodynamic loads prediction using reduced fidelity models," *Journal of Aircraft*, vol. 58, no. 3, pp. 663–676, 2021.

[24] 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, 1 2020, ISSN: 0001-1452. DOI: 10.2514/1.j058291.

[25] J. E. Santos, Y. Yin, H. Jo, *et al.*, "Computationally efficient multiscale neural networks applied to fluid flow in complex 3d porous media," *Transport in Porous Media*, 2021, ISSN: 15731634. DOI: 10.1007/s11242-021-01617-y.

[26] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer school on machine learning*, Springer, 2003, pp. 63–71.

[27] I. Odeha, A. McBratney, and D. Chittleborough, "Spatial prediction of soil properties from landform attributes derived from a digital elevation model," *Geoderma*, vol. 63, no. 3-4, pp. 197–214, 1994.

[28] T. Hengl, G. B. Heuvelink, and A. Stein, "A generic framework for spatial prediction of soil variables based on regression-kriging," *Geoderma*, vol. 120, no. 1-2, pp. 75–93, 2004.

[29] ——, "Comparison of kriging with external drift and regression kriging," ITC Enschede, Netherlands, 2003.

[30] T. Hengl, G. B. Heuvelink, and D. G. Rossiter, "About regression-kriging: From equations to case studies," *Computers & geosciences*, vol. 33, no. 10, pp. 1301–1315, 2007.

[31] W. Sun, B. Minasny, and A. McBratney, "Analysis and prediction of soil properties using local regression-kriging," *Geoderma*, vol. 171, pp. 16–23, 2012.

[32] H. Keskin and S. Grunwald, "Regression kriging as a workhorse in the digital soil mapper's toolbox," *Geoderma*, vol. 326, pp. 22–41, 2018.

[33] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[34] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

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

[36] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[37] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[38] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*, Springer, 1982, pp. 267–285.

[39] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[40] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[41] J. Chen, J. Viquerat, and E. Hachem, "U-net architectures for fast prediction of incompressible laminar flows," *arXiv preprint arXiv:1910.13532*, 2019.

[42] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu, "Towards physics-informed deep learning for turbulent flow prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1457–1466.

[43] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, "Accelerating eulerian fluid simulation with convolutional networks," in *International Conference on Machine Learning*, PMLR, 2017, pp. 3424–3433.

[44] M. Selig, U. of Illinois at Urbana-Champaign. Aeronautical, and A. E. Department, *UIUC Airfoil Data Site*. Department of Aeronautical and Astronautical Engineering University of Illinois at Urbana-Champaign, 1996. [Online]. Available: `https://books.google.com/books?id=dWTTjwEACAAJ`.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[46] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "Repvgg: Making vgg-style convnets great again," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 733–13 742.

[47] J. T. Barron, "Continuously differentiable exponential linear units," *arXiv preprint arXiv:1704.07483*, 2017.

[48] S. Tissera, V. Titarev, and D. Drikakis, "Chemically reacting flows around a double-cone, including ablation effects," in *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2010.

[49] J. Coblish, M. Smith, T. Hand, G. Candler, and I. Nompelis, "Double-cone experiment and numerical analysis at aedc hypervelocity wind tunnel no. 9," in *43rd AIAA Aerospace Sciences Meeting and Exhibit*, 2005, p. 902.

[50] M. S. Holden and T. P. Wadhams, "Code validation study of laminar shock/boundary layer and shock/shock interactions in hypersonic flow, part a: Experimental measurements," *AIAA paper*, vol. 1031, p. 2001, 2001.

[51] J. Harvey, M. Holden, and T. Wadhams, "Code validation study of laminar shockiboundary layer and shock/shock interactions in hypersonic flow part b: Comparison\ith navier-stokes and dsmc solutions," in *39th Aerospace Sciences Meeting and Exhibit*, 2001, p. 1031.

[52] K. R. Holst, R. S. Glasby, J. T. Erwin, D. L. Stefanski, R. B. Bond, and J. D. Schmisseur, "High-order simulations of shock problems using hpcmp create (tm)-av kestrel coffe," in *2018 AIAA Aerospace Sciences Meeting*, 2018, p. 1301.

[53] D. R. McDaniel and T. Tuckey, "Hpcmp createtm-av kestrel new and emerging capabilities," in *AIAA Scitech 2020 Forum*, 2020, p. 1525.

[54] K. R. Holst, R. S. Glasby, J. T. Erwin, *et al.*, "Current status of the coffe solver within hpcmp createtm-av kestrel," in *AIAA Scitech 2020 Forum*, 2020, p. 1530.

[55] R. B. Bond, K. Tatum, G. D. Power, and T. Tuckey, "Capabilities of hpcmp create-av kestrel v11 for hypersonic flight and ground testing with a two-temperature model," in *AIAA Scitech 2021 Forum*, 2021, p. 0236.

[56] I. M. Sobol', "On the distribution of points in a cube and the approximate evaluation of integrals," *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, vol. 7, no. 4, pp. 784–802, 1967.

[57] S. Joe and F. Y. Kuo, "Constructing sobol sequences with better two-dimensional projections," *SIAM Journal on Scientific Computing*, vol. 30, no. 5, pp. 2635–2654, 2008.

[58] I. M. Sobol', D. Asotsky, A. Kreinin, and S. Kucherenko, "Construction and comparison of high-dimensional sobol'generators," *Wilmott*, vol. 2011, no. 56, pp. 64–79, 2011.

[59] B. J. Grier, K. R. Brouwer, E. R. Dreyer, and J. J. McNamara, "Controlling the p-norm function space distribution of linked surrogate parameters," *AIAA Journal*, vol. 57, no. 6, pp. 2659–2662, 2019.

[60] U. S. Atmosphere, "National oceanic and atmospheric administration," *National Aeronautics and Space Administration, United States Air Force, Washington, DC*, 1976.

[61] J. J. Bertin, *Hypersonic aerothermodynamics*. AIAA, 1994.

[62] R. Kimmel, D. Adamczak, K. Berger, and M. Choudhari, "Hifire-5 flight vehicle design," in *40th Fluid Dynamics Conference and Exhibit*, 2010, p. 4985.

[63] D. R. McDaniel, "A summary of new and emerging features in hpcmp createtm-av kestrel," in *AIAA Scitech 2021 Forum*, 2021, p. 0234.

[64] M. MacLean, M. Holden, and A. Dufrene, "Comparison between cfd and measurements for real-gas effects on laminar shock wave boundary layer interaction, i," *Oral Presentation, AIAA Aviation*, vol. 2014, 2014.

[65] M. Vinokur, "On one-dimensional stretching functions for finite-difference calculations," *Journal of Computational Physics*, vol. 50, no. 2, pp. 215–234, 1983.

[66] F. R. Menter, "Two-equation eddy-viscosity turbulence models for engineering applications," *AIAA journal*, vol. 32, no. 8, pp. 1598–1605, 1994.

[67] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[68] A. G. Howard, "Some improvements on deep convolutional neural network based image classification," *arXiv preprint arXiv:1312.5402*, 2013.

[69] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
24-03-2022	Master's Thesis	Sep 2020 – Mar 2022

4. TITLE AND SUBTITLE

Double Cone Flow Field Reconstruction Between Mach 4 and 12 Using Machine Learning Techniques

5a. CONTRACT NUMBER N/A
5b. GRANT NUMBER N/A
5c. PROGRAM ELEMENT NUMBER N/A
5d. PROJECT NUMBER N/A
5e. TASK NUMBER N/A
5f. WORK UNIT NUMBER N/A

6. AUTHOR(S)

Toros, Trevor A., 2d Lt, USAF

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENY)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-ENY-MS-22-M-313

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RQ
11. SPONSOR/MONITOR'S REPORT NUMBER(S) N/A

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution A: Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES
The material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
Analysis of hypersonic vehicle designs are costly due, in part, to the physical phenomena unique to the hypersonic flight regime. It is desirable to consider as many of these phenomena as possible early in vehicle design when computational resources are limited. Reduced order models can provide insight into these phenomena at a low cost by leveraging previous results. The utility of machine learning models for predicting the pressure field around a hypersonic weapon in flight is investigated. A parameterized double cone model is simulated at hypersonic speeds using a steady-state RANS solver, Kestrel. The resulting pressure fields are used to train two neural network (NN) models, the U-Net and the Multiscale Network, as well as two meta models, K-Nearest Neighbors and Regression Kriging. The NN models are designed to extract flow field relationships using distinct methodologies: the U-Net utilizes auto-encoding while the Multiscale Network utilizes a sequential refinement scheme. All models predict the pressure values on a uniform Cartesian grid of much smaller resolution than the unstructured mesh required for CFD simulation. The accuracy, computational complexity, and versatility of the NN are compared against the meta models. Additionally, the ability for each method to accurately predict shock interactions or impingement with downstream vehicle geometry is examined. Such closed-form ML models can provide advantages over traditional CFD solutions as they do not require any meshing of the computational domain and can quickly generate flow field predictions - on the order of seconds. The NN models were found have lacking yet robust performance on this dataset. Additionally, the NN models were shown to be effortlessly applicable to arbitrary geometries that cannot be described using the existing geometric parameterization.

15. SUBJECT TERMS
hypersonics; machine learning; neural networks; flow reconstruction; reduced order modeling

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Ramana V. Grandhi, Ph.D, AFIT/ENY
U	U	U	UU	166	19b. TELEPHONE NUMBER (937) 785-3636, ext. 4723; ramana.grandhi@afit.edu