ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

A MULTI-FIDELITY PREDICTION FRAMEWORK WITH CONVOLUTIONAL NEURAL NETWORKS USING HIGH-DIMENSIONAL DATA

M.Sc. THESIS

Hüseyin Emre TEKASLAN

Department of Aeronautical and Astronautical Engineering

Aeronautical and Astronautical Engineering Programme

JUNE 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

A MULTI-FIDELITY PREDICTION FRAMEWORK WITH CONVOLUTIONAL NEURAL NETWORKS USING HIGH-DIMENSIONAL DATA

M.Sc. THESIS

Hüseyin Emre TEKASLAN
(511191116)

Department of Aeronautical and Astronautical Engineering

Aeronautical and Astronautical Engineering Programme

Thesis Advisor: Prof. Dr. Melike Nikbay

JUNE 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

YÜKSEK BOYUTLU VERİLER İLE ÇOK-DOĞRULUKLU EVRİŞİMSEL SİNİR AĞI TABANLI KESTİRİM

YüKSEK LİSANS TEZİ

Hüseyin Emre TEKASLAN
(511191116)

Uçak ve Uzay Mühendisliği Anabilim Dalı

Uçak ve Uzay Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Melike Nikbay

HAZİRAN 2022

Hüseyin Emre TEKASLAN, a M.Sc. student of ITU Graduate School student ID 511191116, successfully defended the thesis entitled "A MULTI-FIDELITY PREDICTION FRAMEWORK WITH CONVOLUTIONAL NEURAL NETWORKS USING HIGH-DIMENSIONAL DATA", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Melike Nikbay** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Nazım Kemal Üre** Istanbul Technical University	.....
	Dr. Domenico Quagliarella Italian Aerospace Research Center	.....

Date of Submission : June 3, 2022
Date of Defense : June 20, 2022

To my family,

FOREWORD

I would like to express my sincere gratitude to my supervisor Prof. Dr. Melike Nikbay for her tremendous support during my Master's degree. I believe that we accomplished so much together in AeroMDO Multidisciplinary Optimization Laboratory in such a short period. I take pride in all the work I have done with her guidance. I also would like to thank my colleagues in AeroMDO Lab for their technical help and friendships. Finally, I would like to extend my gratitude to Umut Başak Uluğ for the constant and unconditional mental support that she gave during these days.

June 2022 Hüseyin Emre TEKASLAN

TABLE OF CONTENTS

Page

ABBREVIATIONS

ANN : Artificial Neural Network
CPU : Central Processing Unit
CNN : Convolutional Neural Network
GPU : Graphics Processing Unit
LF : Low-fidelity
HF : High-fidelity
MF : Multi-fidelity
MFDNN : Multi-fidelity Deep Neural Network
MFCNN : Multi-fidelity Convolutional Neural Network
FCNN : Fully-connected Neural Network
MLP : Multilayer Perceptrons
ReLU : Rectified Linear Unit
PReLU : Parametric Rectified Linear Unit
POD : Proper Orthohonal Decomposition
NN : Neural Network
RANS : Reynolds Averaged Navier-Stokes
SA : Spallart-Allmaras
MAE : Mean Absolute Error
LFDNN : Low-fidelity Deep Neural Network
FGMRES : Flexible Generalized Minimum Residual
JST : Jameson-Schmidt-Turkel Scheme
CFL : Courant–Friedrichs–Lewy Condition
RMSE : Root Mean Squared Error
LF-RMSE : Low-fidelity Root Mean Squared Error
MF-RMSE : Multi-fidelity Root Mean Squared Error

SYMBOLS

α	: Angle of attack
$\mathbf{b}$	: Learnable bias
β	: Learnable additive batch normalization parameter
C_D	: Drag coefficient
C_L	: Lift coefficient
C_N	: Normal force coefficient
C_p	: Pressure coefficient
δ	: Additive surrogate model
ε	: Learning rate
f	: Layer function
f_H	: High-fidelity function
f_L	: Low-fidelity function
$\mathscr{F}_l$	: Linear correlation network
$\mathscr{F}_{nl}$	: Nonlinear correlation network
g	: Gradient
γ	: Learnable multiplicative batch normalization parameter
$\mathbf{k}$	: Learnable kernel
$\mathscr{L}$	: Loss function
L_1	: Mean absolute error loss function
L_2	: Mean squared error loss function
λ	: Regularization strength
M	: Mach number
m_p	: Dimension of low- and high-fidelity predictions
μ_x	: Mean of an input
N	: Total number of data in a dataset
$N_{\mathbf{train}}$	: Number of data points in the training set
$N_{\mathbf{val}}$	: Number of data points in the validation set
$N_{\mathbf{test}}$	: Number of data points in the test set
n_H	: High-fidelity input dimension
$n_{H,E}$	: Encoded input dimension
n_θ	: Number of learnable parameters
NS_H	: Number of high-fidelity samples
NS_L	: Number of low-fidelity samples
Ω	: Regularization function
ω	: Correlation weight
ω_0	: Initial correlation weight
ω_f	: Optimized correlation weight
P	: List containing integers
p	: Type of norm

Pr	: Probability
ρ	: Multiplicative surrogate model
Re	: Reynolds number
$r_{\textbf{train}}$	: Ratio of the training set
$r_{\textbf{val}}$	: Ratio of the validation set
σ	: Sigmoid function
σ_x	: Standard deviation of an input
θ	: Learnable parameter set of a neural network
t_h	: Computation time of a single high-fidelity simulation
t_l	: Computation time of a single low-fidelity simulation
T_{hf}	: Total computation time of high-fidelity predictions
T_{mf}	: Total computation time of multi-fidelity predictions
tanh	: Hyperbolic tangent function
$\mathscr{U}$	: Uniform distribution
$\mathbf{W}$	: Learnable weight
W_i	: Dimensions of a learnable weight
$\mathscr{X}$	: Input domain
$\mathbf{X}$	: Dataset
$\mathbf{x}$	: Scalar input or multi-dimensional input array
$\bar{\mathbf{x}}$	: Normalized input array
$\tilde{\mathbf{x}}$	: Shuffled input array
$\mathbf{X}_b$	: A batch of a dataset
$\mathbf{x}_C$	: Input of the correlation networks
$\mathbf{x}_L$	: Low-fidelity input array
$\vec{\mathbf{x}}_L$	: Vectorized low-fidelity input array
$\mathbf{x}_H$	: High-fidelity input array
$\vec{\mathbf{x}}_H$	: Vectorized high-fidelity input array
$\mathscr{Y}$	: Output range
$\mathbf{y}$	: Scalar target or multi-dimensional target array
$\hat{y}$	: Predicted value
$\tilde{\mathbf{y}}$	: Shuffled target array
y_L	: Low-fidelity target
$\hat{y}_L$	: Low-fidelity prediction
y_H	: High-fidelity target
$\hat{y}_H$	: High-fidelity prediction

LIST OF TABLES

Page

LIST OF FIGURES

Page

A MULTI-FIDELITY PREDICTION FRAMEWORK WITH CONVOLUTIONAL NEURAL NETWORKS USING HIGH-DIMENSIONAL DATA

SUMMARY

The multi-fidelity analysis is getting more attention day by day and has become a significant research and application area in both academy and industry, especially in the computational sciences. The multi-fidelity analysis, as its name reveals, leverages different fidelity of data. The foremost objective is to attenuate the prohibitive computational or experimental cost of simulations while keeping the accuracy as high as possible. A high number of cheap-to-generate low-fidelity data are corrected by using a low number of expensive-to-generate high-fidelity data. Thus, a notable efficiency is accomplished in comparison to using high-fidelity data solely. The literature offers different multi-fidelity methods such as hierarchical or surrogate-based multi-fidelity schemes. These schemes, in the literature, are classified into three classes: adaptation, fusion, and filtering. In this thesis, the multi-fidelity artificial neural networks, surrogate-based multi-fidelity methods using the fusion principle, are studied for processing high-dimensional data to make multi-fidelity estimations.

Additionally, the multi-fidelity deep neural network was first proposed in 2019. Literature provides a limited number of research on this method which has limitations on the usage of problems with high-dimensional inputs. The issues associated with these limitations occur especially in the case of the usage of high-dimensional inputs and low-dimensional predictions simultaneously. Therefore, the dimensional discrepancy impedes the learning of linear and nonlinear correlations between high-fidelity and low-fidelity data sources. On the other hand, the multi-fidelity deep neural networks compose fully-connected networks which can only process scalars or vectors. Thus, 2-dimensional data with locality must be vectorized to make them suitable for neural network processing. Vectorization brings about a loss of information due to the dislocation of highly-correlated regions. Two novel multi-fidelity neural network architectures tailored for high-dimensional inputs are proposed to improve multi-fidelity predictions on the problems inherited from these conditions. First, a fully-connected encoder is implemented within the existing multi-fidelity neural network architecture in the literature. The encoder performs dimensionality reduction by mapping its input onto a low-dimensional subspace by only extracting the latent features of a given input vector. Therefore, the dimensional discrepancy is hindered and the training performance of multi-fidelity deep neural networks is enhanced. Additionally, convolutional layers are incorporated into the modified multi-fidelity deep neural networks, named multi-fidelity convolutional neural networks, so as to preclude vectorization so does the loss of correlation information.

Besides, one of the obstacles to using flow fields as neural network inputs is that these spatial domains must be represented in the matrix/tensor notation; thus, a neural network layer can be capable of processing a given input. In this study, the interpolation of flow variables on both structured and unstructured grids onto Cartesian grids is seen as a remedy. A preliminary study investigating the influence of interpolation schemes and Cartesian grid sizes on a convolutional neural network prediction accuracy is also presented where further studies are based on the findings. Moreover, in the multi-fidelity deep neural network, the autoregressive function uses the weighted approximations of the linear and nonlinear correlation between high- and low-fidelity datasets. The tuning of correlation weight in training is not explicitly stated in the literature. This is not surprising since the research on multi-fidelity neural networks is in rudimentary stages. In this study, a brief investigation of how the initial correlation weight affects its tuned value and model prediction performance is also presented.

In the light of the findings, the proposed methods are compared with the multi-fidelity deep neural networks from the literature using a 2-dimensional flow-varying supercritical airfoil problem. Mach number, angle of attack, and Reynolds number are considered flow variables. The main objective of this study is to make a multi-fidelity prediction of aerodynamic coefficients using pressure coefficient fields around the airfoil with an acceptable accuracy level and computational cost. Furthermore, to generate the dataset, a coarsely discretized flow domain around the airfoil is solved using the SU2 Euler solver for low-fidelity data whereas a relatively finer grid is utilized for the high-fidelity data to obtain viscous solutions using the Spallart-Allmaras turbulence model. The difference in the computational cost for each high-fidelity and low-fidelity computational fluid dynamics analysis is around 31.2 seconds where the high-fidelity solver is costlier more than 19 times relative to the low-fidelity solver in terms of computation time. In total, the used dataset contains 200 high-fidelity and 300 low-fidelity simulation results. Each considered multi-fidelity neural network method is trained with an increasing high-fidelity sample size using pressure coefficient fields which are interpolated onto 64-by-64 Cartesian grids. The performance metrics to compare the methods are determined as the test accuracy, physical training time, and the size of the high-fidelity samples. Results demonstrate that the proposed multi-fidelity neural network architectures outperform the multi-fidelity deep neural networks in predictive modeling using high dimensional inputs by improving the multi-fidelity prediction accuracy up to 78.7%.

YÜKSEK BOYUTLU VERİLER İLE ÇOK-DOĞRULUKLU EVRİŞİMSEL SİNİR AĞI TABANLI KESTİRİM

ÖZET

Günümüzde bilgisayar işlemci performansları, geçmişe kıyasla çok daha yüksek olsa da hesaplamalı bilimler alanında kullanılan ve yüksek doğrulukta sonuçlar verebilen nümerik analiz yöntemleri hala hesaplama süresi açısından oldukça maliyetlidir. Bu durum havacılık ve uzay alanında, tasarım uzayında birçok hesaplama gerektiren optimizasyon, belirsizlik analizi ve hassasiyet analizi gibi uygulamalarda yüksek doğruluk ve hassasiyette sonuçlar veren yöntemlerin kullanımını ciddi ölçüde kısıtlamaktadır. Hesaplamalı akışkanlar dinamiği özelinde, günümüzdeki uygulamalarda on milyonlarca çözüm ağı elemanı içeren akış alanlarını sayısal yöntemler ile çözmek gerekmektedir. Bu çözüm ağının birçok işlemciye bölüştürülüp işlemci başına düşen yükün azaltıldığı ve çözüm süresinin hızlandırıldığı paralel hesaplama ile bile ancak saatler mertebesinde akış çözümü sağlanabilmektedir. Dolayısıyla yüksek doğrulukta sonuçlar üretebilen ancak hesaplama maliyetleri de bir o kadar yüksek olan sonlu elemanlar ve sonlu hacimler yöntemleri yerine panel yöntemleri gibi hesaplama açısından yük oluşturmayan ve daha düşük doğruluğa sahip yöntemler kullanılmaktadır. Bu noktada farklı fiziksel doğruluk seviyesine sahip yöntemleri birleştiren çok-doğruluklu analiz metotları, son yıllarda tasarım optimizasyonunda oldukça yaygın şekilde kullanılmaktadır.

Çok-doğruluklu analiz yöntemleri, hesaplama açısından verimli olan çok sayıda düşük-doğruluklu analiz verisi ile az sayıda yüksek-doğruluklu ancak üretimi pahalı olan veriyi belirli matematiksel metotlarla birleştiren yöntemlerdir. Çok-doğruluklu analiz metotlarının ana hedefi sadece yüksek-doğruluklu yöntemlerin kullanımına kıyasla hesaplama maliyetini düşürmek ve aynı zamanda da fiziksel doğruluğu yüksek-doğruluklu yöntem seviyesinde tutmaktır. Bu yöntemler, sağladıkları hesaplama verimliliği sayesinde hem akademi hem de endüstride birçok tasarım optimizasyon çalışmasında kullanılmaktır. Literatürde çok-doğruluklu analiz yöntemleri 3 sınıfa ayrılmıştır: adaptasyon, birleşim ve filtreme. Yöntemler, her sınıf içerisinde hiyerarşik ve temsili model tabanlı olmak üzere iki alt sınıfa ayrılabilir. Bu tez kapsamında birleşim prensibini kullanan temsili model tabanlı çok-doğruluklu derin öğrenme metotları incelenmektedir.

Derin öğrenme yöntemlerinin ana fikri 1960'lı yıllara dayanmaktadır. Fakat uygulama alanları, artan hesaplama gücü ve veri toplama hızı ile son yıllarda artmıştır. Günümüzde derin öğrenme uygulamalarına hemen hemen her alanda rastlamak mümkündür. Bu yöntemler, havacılık ve uzay alanında, otonom uçuş, hava trafik yönetimi, optimizasyon ve belirsizlik analizi gibi uygulamalarda kullanılmaktadır.

Yapay sinir ağlarının en önemli avantajlarından biri klasik makine öğrenmesi yöntemlerinde kullanılan öznitelik çıkarımının olmamasıdır. Öznitelik çıkarımı alan uzmanlığı gerektirdiği gibi model performansını da yüksek oranda etkilemektedir. Derin öğrenmenin diğer bir önemli getirisi ise büyük veri setlerine uygun bir yöntem olmasıdır. Klasik makine öğrenmesi yöntemleri artan veri miktarına her zaman artan bir model performansıyla cevap veremez ve bu anlamda doygunluğa ulaşırken, sinir ağları büyük veri setleri ile modelin kestirim gücünü artırabilmektedir. Çok-doğruluklu derin sinir ağları ise literatürde paylaşılmış oldukça yeni bir metottur. Diğer çok-doğruluklu yöntemlere benzer şekilde az sayıda yüksek-doğruluklu ve çok sayıda düşük-doğruluklu verinin kullanıldığı bu metot ile ilgili geniş bir literatüre rastlamak mümkün değildir. Dolayısıyla açık ve güncel bir araştırma konusudur. Çok-doğruluklu derin sinir ağları otoregresif bir model kullanır ve tam bağlantılı katmanlardan oluşan 3 alt ağdan oluşur. Bu ağlardan ilki verilen girdiler ile düşük-doğruluklu kestirimler yapar. İkinci ve üçüncü ağlar ise düşük- ve yüksek-doğruluklu veriler arasındaki doğrusal ve doğrusal olmayan korelasyonu modellemek için kullanılır. Literatürde az sayıda yapılan çalışmalar, çok-doğruluklu sinir ağlarının analitik ve düşük boyutlu vektörel girdi ve çıktıların kullanıldığı 2 boyutlu aerodinamik problemlerde etkin sonuçlar verdiğini göstermiştir. Mevcut çalışmada, literatürde henüz bulunmayan, yüksek boyutlu verilerin çok-doğruluklu sinir ağları ile birlikte kullanımı ele alınmıştır. Bu kapsamda 2 boyutlu akış alanları yüksek boyutlu girdi olarak kullanılmıştır.

Literatürde önerilen çok-doğruluklu sinir ağlarının yüksek veriler ile kullanımında iki önemli eksik göze çarpmakta ve bu çalışmada, yöntemin eksikliklerin iyileştirilmesi için iki farklı çok-doğruluklu sinir ağı mimarisi önerilmektedir. Bu eksiklerden birincisi girdi ve çıktı arasındaki boyut farkının yüksek olduğu durumlarda ortaya çıkar. Daha önce bahsedilen korelasyon ağları, tahminlerin ve girdilerin birleşim vektöründen korelasyon kestirimi yapar. Bu iki vektör arasındaki boyut farkının fazla olduğu durumlarda, veri setleri arasındaki korelasyonun öğrenilmesi zorlaşmakta ve çok-doğruluklu kestirimler yapmak mümkün olmamaktadır. Bu tez çalışmasında, söz konusu probleme çözüm olarak çok-doğruluklu sinir ağı yapısına doğrusal kodlayıcı (encoder) veya kod çözücünün (decoder) eklenmesi önerilmektedir. Kodlayıcılar verilen girdiyi yüksek-boyutlu uzaydan daha düşük boyutlu uzaya taşırken kod çözücüler tam tersini yapar. Böylece bu derin öğrenme elemanları, girdi ve çıktı arasındaki boyut farkını azaltıp çok-doğruluklu kestirim performansını artırmak için kullanılabilir. Bu çalışmada süperkritik bir kanat profili etrafındaki yüksek-boyutlu akış alanlarının girdi, düşük boyutlu aerodinamik katsayıların çıktı olarak kullanıldığı bir problem ele alınmıştır. Dolayısıyla girdi ve çıktı arasındaki boyut farkını azaltmak için yüksek eleman sayısına sahip girdi vektörlerinin kodlayıcılar ile çok daha düşük boyutlu vektörlere dönüştürüldüğü ağ yapıları kullanılmıştır. Bu tez kapsamında, bir kodlayıcı ağının entegre edildiği çok-doğruluklu sinir ağlarına, değiştirilmiş çok-doğruluklu sinir ağları adı verilmiştir. Bunlara ek olarak, literatürdeki mevcut yöntemin ikinci dezavantajı ise kullandığı tam bağlaşımlı katmanların getirdiği skaler ya da vektörel girdi zorunluluğudur. Dolayısıyla sinir ağlarının eğitiminde ve kestirim için kullanılmasında, birden fazla boyuta sahip girdilerin vektörleştirilmesi gerekmektedir. Fakat, normalde birden fazla boyuta sahip olup vektör haline getirilen verilerde bilgi kaybı yaşanmakta, sinir ağı modelinin kestirim performansını düşürmektedir. Bu kaybın ana sebebi, yüksek korelasyon içeren bölgelerin vektör içerisinde birbirinden uzak olacak şekilde konumlanmasından kaynaklanır. Benzer bir problem daha önce derin öğrenme yöntemleri ile görüntü işlemede karşılaşılmış

ve evrişimsel sinir ağları kullanılarak çözülmüştür. Akış alanları da birbiri ile yüksek korelasyonlu birçok noktadan oluşan 2 boyutlu bir yüzey ya da 3 boyutlu bir hacimdir. Bu çalışmada, aerodinamik çözüm alanlarının, bilgi kaybı yaşanmadan sinir ağları ile işlenebilmesi için evrişimsel sinir ağları, çok-doğruluklu analiz yapısına entegre edilmiştir.

Üzerinde düşünülmesi gereken bir diğer konu ise akış alanlarının nasıl matris ya da tensör notasyonu ile temsil edilebileceğidir. Çünkü 2 ve daha yüksek boyutlu evrişimsel sinir ağları matris/tensör formatında veri işleyebilmektedir. Bu çalışmada, literatürde daha önce kullanılmış bir yöntem olan, akış alanlarının kartezyen bir ağın üzerine interpolasyonu kullanılmıştır. İnterpolasyon hem yapısal hem de yapısal olmayan çözüm ağlarına kolayca uygulanabilir bir yöntemdir. Ayrıca veri noktası sayısını ciddi ölçüde azaltmaktadır. Bu yöntem literatürde detaylı şekilde irdelenmediği için bu tezde bir ön çalışma olarak, interpolasyon yaklaşımının bir evrişimsel sinir ağının kestirim performansına olan etkisi incelenmiştir. Bu uygulamada, farklı kartezyen ağ boyutları ve interpolasyon metotları, akış ve geometri değişimli bir kanat profiline uygulanmıştır. Daha sonra interpole edilmiş akış alanları kullanılarak, taşıma ve sürükleme katsayılarının bir evrişimsel sinir ağı ile tahmini gerçekleştirilmiştir. Başka bir ön çalışma ise çok-doğruluklu otoregresif modelde kullanılan korelasyon ağırlığını incelemek için yapılmıştır. Bu çalışmada ise korelasyon ağırlığının başlangıç değerinin optimizasyon sonundaki değerine olan etkisi, literatürde bulunan analitik çok-doğruluklu problemler üzerinde irdelenmiştir.

Yapılan ön iki çalışmanın bulguları, bu tezin asıl uygulama konusu olan süperkritik bir kanat profiline ait basınç katsayısı alanları kullanılarak aerodinamik katsayıların çok-doğruluklu kestiriminde kullanılmıştır. Bu uygulamada, kestirim doğruluğu ve eğitim süresi gibi performans ölçüleri göz önünde bulundurularak, yeni önerilen çok-doğruluklu sinir ağı mimarilerinin etkinliği, literatürdeki çok-doğruluklu sinir ağları ile kıyaslanmıştır. Mach sayısı, hücum açısı ve Reynolds sayısı akış değişkenleri olarak kabul edilmiştir. Veri seti üretimi için açık-kaynaklı SU2 akış çözücüleri kullanılmıştır. Düşük-doğruluklu veri seti, kaba bir çözüm ağı üzerinde Euler denklemleri çözülerek elde edilmiştir. Yüksek-doğruluklu veri seti ise Reynolds Ortalamalı Navier-Stokes ve Spallart-Allmaras türbülans denklemlerinin daha sık bir akış ağı üzerinde çözümü ile üretilmiştir. Benzer yaklaşım literatürde farklı çalışmalarda da kullanılmıştır. 36 Intel(R) Xeon(R) Gold 6148 CPU 2.40GHz işlemcinin paralel olarak kullanıldığı veri seti üretiminde, tek bir yüksek-doğruluklu analizin çözümü, düşük-doğruluklu analiz çözümünden yaklaşık 19 kat daha uzun bir sürede tamamlandığı gözlemlenmiştir. Üretilen veri seti, toplamda 200 tane yüksek-doğruluklu ve 300 tane düşük-doğruluklu analiz sonucu içermektedir. Kullanılan her bir yöntem, kademeli olarak artırılan yüksek-doğruluklu veri seti ile test edilmiştir. Literatürdeki çok-doğruluklu yönteme kıyasla, önerilen yeni sinir ağı mimarileri ile elde edilen çok-doğruluklu tahminlerde %78.7'e varan daha yüksek doğruluk seviyelerine ulaşılmıştır.

1. INTRODUCTION

Artificial neural networks (ANNs) have emerged a new era in computer and computational sciences. Practical conundrums back then a few decades such as image, video & audio recognition, and machine translation can now be resolved with an adequate size of data and neural networks in the tasks of classification, clustering, and estimation. However, the idea of the neural network is not newly fashionable, and the concept lies back around the 1960s. Even though this method has a history nowadays, it has recently become renowned as central processing units (CPU) and graphics processing units (GPU) offer adequate computing power. Besides, data accumulation is getting easier and cheaper. Thus, this outstanding data-driven metamodeling technique has been disseminated over time and become ubiquitous across different disciplines such as bioinformatics [2–4] and geosciences [5–7].

In fact, data-driven metamodeling is used in computational sciences for a few decades. In this field, it is mostly known as surrogate or reduced-order modeling. In the literature, numerous data-driven surrogate modeling methods exploiting different mathematical approaches such as mapping and regression are available. Principal component analysis, proper orthogonal decomposition, dynamic mode decomposition, polynomial chaos expansion, and Kriging can be given as examples of surrogate modeling techniques that are studied in computational sciences for complex and expensive-to-solve problems. Recently, ANNs gained a considerable amount of content in aerospace. Guidance [8, 9], air traffic management [10, 11], optimization, uncertainty quantification, and inverse design are some of the fields that incorporate deep learning methods in research. For instance, in computational aerodynamics and multidisciplinary studies involving aerodynamics, data are scarce due to the computational expense of high-fidelity computational fluid dynamics simulations and the high economic cost of experimental test setups. Data acquisition is not an option the entire time which makes available data precious. Therefore, the main purpose of the usage of neural networks is to establish a surrogate model using an available

physical solver output or experimental data in design optimization and uncertainty quantification studies. Some of the research applying ANNs in aerodynamic problems which are highly nonlinear, costly, complex, and generally black-box can be found in the literature [12–19].

Moreover, even with increased computational resources, solving highly nonlinear and involved physical problems with high accuracy does still pose a problem in engineering applications. In most cases, using a cheaper numerical method yields results with lower accuracy when compared to more expensive tools. Albeit the practicality of low-fidelity tools in preliminary conceptual design, detailed design studies necessitate more accurate and precise frameworks. At this point, multi-fidelity analysis that leverages the computational efficiency of low-fidelity and the accuracy of high-fidelity models simultaneously within the same framework is an outstanding option; accordingly, multi-fidelity methods have been highly appealing in computational sciences and engineering. Moreover, the multi-fidelity analysis leverages at least two different fidelity of data where one is superior to the other in terms of physical accuracy. The main motivation in the application of multi-fidelity methods is to decrease the computational burden of high-fidelity solvers or the expense of experimental tests. Since some of the engineering applications such as design optimization and uncertainty analysis require numerous function evaluations, even up to several million, it is not viable to use only high-fidelity (HF) partial differential equation solvers within a design process. On the other hand, designers must sacrifice a considerable amount of accuracy using only low-fidelity (LF) solvers, especially in complex problems where low-fidelity tools cannot sufficiently capture the underlying physics. Multi-fidelity methods combine a small number of HF and a large number of LF analyses' data to remarkably reduce the computational expenses while keeping the accuracy as high as in HF solutions. In the literature, multi-fidelity polynomial chaos expansion with orthogonal polynomials and co-Kriging (Multi-fidelity Gaussian regression) that uses maximum likelihood approach are extensively appealing in design optimization [20–25].

Moreover, a novel multi-fidelity approach with the assistance of deep neural networks was first proposed in the literature [26] in 2019. This method leveraged 3 fully-connected neural networks to estimate low-fidelity output and linear and

nonlinear correlations between a high-fidelity and a low-fidelity dataset with an autoregressive scheme. Thus, both low- and high-fidelity predictions could be made with a single multi-fidelity deep neural network (MFDNN). This approach has been embraced and applied in the aerospace community. In [27], the computational efficiency of gradient enhanced MFDNN was compared with the efficiencies of gradient enhanced neural networks and multi-fidelity neural networks using analytical functions and an airfoil optimization problem. Authors of [28] investigated the efficacy of MFDNN in the prediction of aerodynamic coefficients using LF inviscid and HF viscous airfoil flows and shared a comparison with the results of coKriging. A comprehensive shape optimization study of an airfoil and a wing-body configuration in the transonic regime was provided in [29]. A multi-fidelity Bayesian neural network was presented in [30] that applied to noisy data with different fidelity levels and was also able to solve inverse problems using partial differential equations.

One of the drawbacks of MFDNN is that it only consists of affine layers and activation functions which makes it fully-connected neural networks or known as multilayer perceptrons. Thus, the input of an MFDNN architecture must be provided in a vector form. However, vectorization of input data may cause loss of information for a certain type of data due to the dislocation of highly correlated data points. This problem was experienced in image processing studies in the past. We can elaborate this issue: each image is a tensor with a combination of pixels. Each pixel has a unique location on an image such that they form a visual representation of an object by surrounding each other. Nevertheless, taking each row of pixels and concatenating them side by side to obtain a vector of pixels deteriorates the quality and meaning of the visual representation; in other words, it brings about a loss of information on the image. For computational sciences, 2- and 3-dimensional differentiable physical solution domains can be thought of as images. A vertex of a grid on a 2- or 3-dimensional spatial domain, let's say a flow field, is mostly expected to be highly correlated with the surrounding vertices. Therefore, the representation of a flow field with a vector notation disarrays high correlation regions and leads to the loss of gradient information in a quantity of interest. This issue may engender a reduction in neural network performance due to the lack of features. In addition, the representation of an involved flow field may require a large number of matrix/tensor elements; thus, fully-connected networks that

are embedded in MFDNN suffer the curse of dimensionality with the necessity of a large learnable parameter set to process these high-dimensional vectorized flow fields. As a result, both memory requirement and training time increase regardless of the loss of data information.

This thesis addresses the drawbacks of the MFDNN method stated above in the application of computational flow fields by proposing a multi-fidelity convolutional neural network (MFCNN). Convolutional neural networks (CNNs) have been vastly put into practice in image and video processing due to their ability to extract local features. CNNs are expected to perform well with 2-dimensional flow field data and incorporated into multi-fidelity analysis with an alike network architecture in MFDNN. Instead of vectorizing 2-dimensional flow fields, each flow field is processed in a high-dimensional tensor form by an MFCNN. Therefore, a loss of information is prevented. Moreover, sparse interactions and parameter sharing of CNNs allow us to establish deep convolutional neural networks with a small number of learnable parameters which alleviates the curse of dimensionality so they are preferable for high-dimensional data such as computational flow fields. Furthermore, an intermediate form between MFDNN and MFCNN architectures, named modified-MFDNN, is also proposed and investigated in predictions using flow fields. The modified-MFDNN leverages the MFDNN architecture with a combination of the dimensionality reduction feature of MFCNNs.

In the context of the thesis, a multi-fidelity supercritical airfoil problem is addressed to evaluate the performances of three different multi-fidelity neural network architectures using computational flow fields; multi-fidelity deep neural networks presented in the literature, the modified multi-fidelity deep neural networks, and multi-fidelity convolutional neural networks. The generated dataset is used to make multi-fidelity predictions of aerodynamic coefficients. The performance of each method is evaluated according to their test accuracy and computational needs. The obtained results prove that the proposed MFCNN architecture outclasses both MFDNN and modified-MFDNN in the prediction of aerodynamic coefficients using flow fields.

The layout of the thesis is described as follows. In Section 2, deep learning, deep learning methods exploited in this study, multi-fidelity analysis and different multi-fidelity neural network architectures are introduced. In Section 3, the dataset

generation of the considered 2-dimensional airfoil problem is presented as well as the investigation of all three methods on the multi-fidelity prediction of aerodynamic coefficients. Eventually, Section 4 compares the obtained results of each method and proposes a future work.

2. METHODOLOGY

2.1 Deep Learning

Deep learning is a set of enhanced machine learning methods. Its main principles lie on linear algebra, statistics, optimization, and computer sciences. The most outstanding convenience that it provides is that it eliminates the dependency of machine learning model performance in feature extraction which requires human intervention and domain expertise. These features and even more that a human may not consider are automatically extracted by means of hidden layers. Hence, it offers end-to-end learning. In addition, the prediction performance of classical machine learning algorithms saturate at a point even the dataset size is increased. In opposition, deep learning methods get a part of their strength from big data which means that it performs better with an increasing data size. This adaptation of deep learning methods comes from the concept of layers. More layers can be stacked on top of each other to increase the learning potential of a neural network.

Despite the strong mathematical foundations, deep learning applications ground on heuristic approaches such as constructing network architecture and hyperparameter tuning [31]. Due to the stochastic nature of the training process, the artificial neural networks which use the same dataset, the same network architecture, and the same training algorithm is expected to yield different model performances for even consecutive runs. This is mainly caused by the random network initialization and dataset shuffling. At this point, random network initialization helps to prevent local minimums and saddle points. Furthermore, deep learning methods are highly contingent upon datasets. For example, a network that is suitable for a pressure dataset may not achieve the same performance for the velocity data even the datasets belong to the same flow field solution. Hence, dataset pre-processing, construction, and training of CNNs would require user intervention for satisfactory model prediction performance.

2.1.1 Fully-connected neural networks

Fully-connected neural networks (FCNN) or also known as multilayer perceptrons (MLP) can be thought of a composition of functions with parameters where an output of a function is the input of the next function. As any other surrogate modeling methods such as Kriging, proper orthogonal decomposition, and polynomial chaos expansion, FCNNs are utilized for data-driven function approximations; besides, they are quite useful surrogate modeling tools that has been broadly appealed in classification and prediction tasks in both academic and industrial applications.

To elaborate the mathematical structure, the universal approximation theorem states that any smooth function on a closed and bounded subset of $\mathbb{R}^n$ can be approximated using an FCNN with at least a hidden layer [32]. The approximation is mostly achieved with gradient-based optimization algorithms by optimizing the parameter set of an FCNN, which are called learnable parameters, based on the provided dataset. This process is called training and elucidated in detail under Section 2.1.3. Besides, an FCNN can involve solely linear functions which may be useful for linear function approximations. However, in the most of the cases, the inclusion of nonlinear functions is required due to the nonlinear behaviour of the engineering problems.

In the literature, each function within an FCNN is named layer. Specifically, the very first function that takes the input provided by the user is called the input layer where the last function that yields the prediction is known as the output layer. The functions that are stacked on top of each other between the input and output layers are named hidden layers. A mathematical representation of an FCNN, $f(\mathbf{x})$, with an input $\mathbf{x}$ is given in equation **(2.1)**.

$$f(\mathbf{x}) = (f_n \circ f_{n-1} \circ \cdots \circ f_2 \circ f_1)(\mathbf{x}) \tag{2.1}$$

where $f_1(\mathbf{x})$ and $f_n(.)$ respectively correspond to the input and output layers whereas the rest are hidden layers. The total number of layers refers to the deepness of an FCNN. This terminology brings about the name "deep learning" [1]. As it is aforementioned, FCNNs may consist linear and nonlinear layers. The linear layer that makes an affine transformation to a given input is mathematically expressed as

in equation **(2.2)**. In this equation, **W** refers to weight and **b** is the bias. These multidimensional arrays are learnable parameters of a linear layer.

$$f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{W} + \mathbf{b} \tag{2.2}$$

Furthermore, nonlinear layers are called activation layers in the deep learning terminology. Numerous tailored nonlinear layers are available in the literature. However, sigmoid function, hyperbolic tangent function, and rectified linear unit (ReLU) [33] are the most outstanding activation layers. None of these stated activation functions has learnable parameters; nonetheless, some variations of ReLU such as Parametric ReLU (PReLU) [34] possess parameters optimized during a training. Sigmoid function maps its inputs in the range of (0,1) and is given in equation **(2.3)**.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.3}$$

Additionally, hyperbolic tangent function behaves similar to sigmoid function yet it transforms a given input in the range of (-1,1). Hyperbolic tangent function is provided with the equation **(2.4)**.

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2.4}$$

Unlike the sigmoid and hyperbolic tangent activation functions, ReLU is a piecewise function as in equation **(2.5)**.

$$\mathrm{ReLU}(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{2.5}$$

In the recent deep learning applications, ReLU is recommended as default activation layer [1]. Sigmoid, hyperbolic tangent, and ReLU functions for input $\mathbf{x} \in [-10, 10]$ are depicted in Fig. 2.1.

2.1.2 Convolutional neural networks

Convolutional neural network (CNN) is an adapted deep learning algorithm to process data that have local correlations [1]. A convolution operation uses at least a weight matrix known as kernel or filter, **k**, to extract the features of a given input. The expression for convolution is given in equation **(2.6)**. In the literature, the symbols $\otimes$ and $*$ are widely used to indicate convolution operation. These element-wise

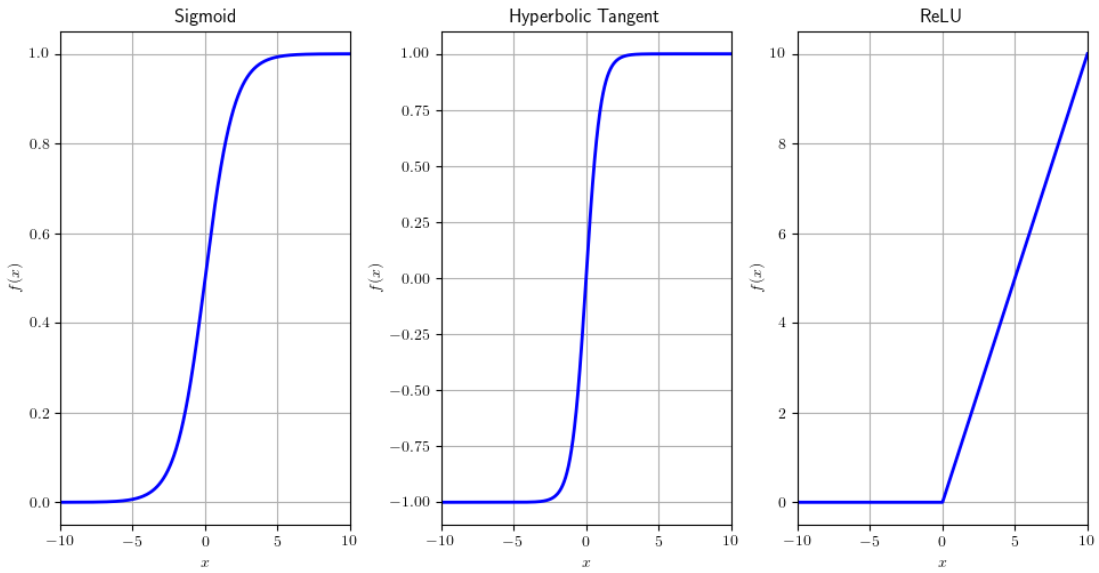

Figure 2.1 : Sigmoid, hyperbolic tangent, ReLU activation layers.

multiplication operators are also known as point-wise multiplication operator or Hadamard product.

$$f(\mathbf{x}) = \mathbf{x} \otimes \mathbf{k} \tag{2.6}$$

In addition, kernels are multidimensional arrays and learnable parameters of a convolution layer that are adapted during training based on data. In the convolution operation, kernels stride over an input and are used to compute the weighted summation of an overlapped submatrix that belongs to the input. The obtained weighted summation is called the output of a convolution operation or feature map. For the sake of clarity, the convolution terminology is elucidated with an example input, $\mathbf{x} \in \mathbb{R}^{5\times5}$, and a kernel, $\mathbf{k} \in \mathbb{R}^{3\times3}$ in Fig. 2.2.

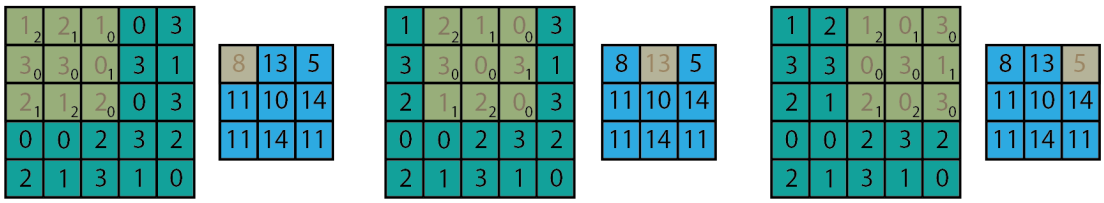

Figure 2.2 : A convolution operation over the input (green) with the kernel (yellow), and a resulting feature map (blue).

As seen in Fig. 2.2, the kernel moves over the input with a unit pace means that passes one column or row at a time. The pace of the kernel between two consecutive convolution operation is called stride. The same demonstration of the convolution is

provided in Fig. 2.3 with the stride of 2. Increasing the stride value yields a feature map with a smaller size.

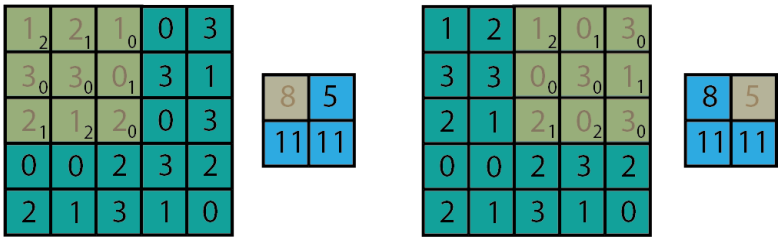

Figure 2.3 : A demonstration of a convolution operation over the input with the kernel with the stride of 2.

In addition, a convolution layer maps the input data by using learnable kernels into a different low-dimensional hyperplane. The input is downsampled based on the layer parameters such as the kernel size and stride. Thus, CNN is a preferable data-driven method for especially high-dimensional problems like flow solutions where the number of grid points may go up to tens of millions. Besides, this method is analogous to proper orthogonal decomposition (POD) where an input is shrunk by projecting it onto an optimal lower-dimensional subspace. Moreover, CNNs mostly do not only consist of convolution layers, yet a certain combination of linear, nonlinear, pooling, and regularization layers [18, 35–37].

To begin with, pooling layers make a neural network (NN) less susceptible to input variance [1]. Likewise to convolution layer, they use striding kernels to yield statistics of an overlapping input, such as the mean or the maximum value. Some of the commonly implemented pooling layers are maximum pooling, average pooling, and power-average pooling. The maximum pooling results to the maximum value of the given input as mathematically given below.

$$f(\mathbf{x}) = \max(\mathbf{x}) \tag{2.7}$$

The average pooling yields the average of the input with the total number of elements of N.

$$f(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.8}$$

Lastly, the power-average pooling is

$$f(\mathbf{x}) = \sqrt[p]{\sum_{i=1}^{N} x_i^p} \tag{2.9}$$

where p denotes the type of norm. Furthermore, regularization layers are used to improve the performance of network by preventing overfitting and increasing the generalization capability. In this thesis, batch normalization and dropout are utilized as regularization layers. Batch normalization is first introduced in [38], and it enables an NN to learn faster with higher learning rates in optimization and makes less dependent about weight, kernel, and bias initialization. It also acts as a regularizer which increases the generalization capability of an NN and prevents overfitting. Correspondingly, dropout [39] is a regularizer which omits randomly selected learnable parameters during an optimization iteration by temporarily setting them as 0 based on a probability. Thus, it improves the generalization capability of an NN. In some cases, using batch normalization layers eliminates the need for dropout. The batch normalization layer performs the operation given below.

$$f(\mathbf{x}) = \gamma \frac{\mathbf{x} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}} + \beta \tag{2.10}$$

Here, the mean and standard deviation of the given input are respectively symbolized with $\mu_{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$. Besides, γ and β are learnable parameters and linearly transform the normalized input with standard Gaussian distribution. Additionally, the dropout operation is illustrated in Fig. 2.4.

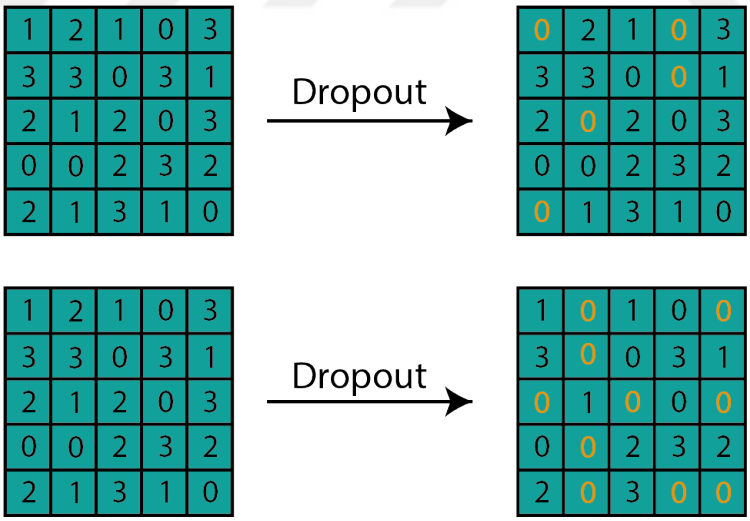

Figure 2.4 : A demonstration of a dropout with the probabilities of an element to be zeroed of Pr = 0.2 (top) and Pr = 0.4 (bottom).

2.1.3 Optimization for training convolutional neural networks

The learning algorithms can be classified as unsupervised learning and supervised learning. Unsupervised learning refers to a training where the algorithm explores the

information within a data without data labels. On the other hand, supervised learning algorithms performs with not only a data but also a corresponding target set. Therefore, each input of a neural network has a matching ground truth value, also known as target value. Training of a neural network with a supervised learning algorithm is achieved with the minimization of discrepancies between the network prediction and the target value. In the context of this thesis, only supervised learning is applied to train neural networks.

Training or learning in deep learning terminology corresponds to optimization of network parameters with a dataset in a way that the network is capable of approximating the function to which dataset belongs. Let θ is a learnable parameter set including networks weights, $\mathbf{W}$, biases, $\mathbf{b}$, kernels $\mathbf{k}$, and any other learnable parameters. Then, a neural network can be defined from a domain $\mathscr{X}$ to a range $\mathscr{Y}$ as $f(\mathbf{x};\theta) : \mathscr{X} \rightarrow \mathscr{Y}$. Let the target of an input vector $\mathbf{x}$ be $\mathbf{y}$ and the NN prediction to a given input vector $\mathbf{x}$ be $\hat{\mathbf{y}}$. Then, an error function that computes the difference between the NN prediction and the target can be defined as $\mathscr{L}(\mathbf{y},\hat{\mathbf{y}}) : \mathscr{Y} \rightarrow \mathbb{R}$. The function $\mathscr{L}(\mathbf{y},\hat{\mathbf{y}})$ is named loss function which (i.e. cost function or objective function in optimization terminology). Therefore, the training of an NN with a parameter set θ can be mathematically expressed as an unconstrained optimization problem given in equation **(2.11)**. The usage of equation **(2.11)** is limited due the lack of constraints; however, this will be explained after the introduction of overfitting term for the convenience of the reader.

$$\min_{\theta} \ \mathscr{L}(\mathbf{y},\hat{\mathbf{y}}) \tag{2.11}$$

Furthermore, the optimization problem given in equation **(2.11)** is usually solved with a gradient-based algorithm (e.g. gradient descent); however, gradient-free optimization schemes are also employed in some of research [40, 41]. Additionally, since each of layers has an exact analytical derivative expression, numerical gradient computation schemes such as finite differencing are not needed. The information of gradient is computed by applying the chain rule of calculus. This method is known as backpropagation [42]. Consider equation **(2.1)** as a simple neural network. Let the prediction of the network, output of each layer except the last layer and the target

vector be $\hat{\mathbf{y}}$, $\hat{\mathbf{y}}_i$ and $\mathbf{y}$, respectively. Then, the loss value of this prediction becomes $\mathscr{L}$ that is the short notation of $\mathscr{L}(\mathbf{y},\hat{\mathbf{y}})$. Thus, the backpropagation of the loss with respect to the input vector is given as in equation **(2.12)**.

$$\frac{\partial \mathscr{L}}{\partial \mathbf{x}} = \frac{\partial \mathscr{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \hat{\mathbf{y}}_{n-1}} \frac{\partial \hat{\mathbf{y}}_{n-1}}{\partial \hat{\mathbf{y}}_{n-2}} \cdots \frac{\partial \hat{\mathbf{y}}_2}{\partial \hat{\mathbf{y}}_1} \frac{\partial \hat{\mathbf{y}}_1}{\partial \mathbf{x}} \tag{2.12}$$

Besides, the gradient of weight, bias, and kernel of an arbitrary j^{th} layer can be respectively expressed as provided in equations **(2.13)**, **(2.14)** and **(2.15)**.

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}_j} = \frac{\partial \mathscr{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \hat{\mathbf{y}}_{n-1}} \frac{\partial \hat{\mathbf{y}}_{n-1}}{\partial \hat{\mathbf{y}}_{n-2}} \cdots \frac{\partial \hat{\mathbf{y}}_{j+1}}{\partial \hat{\mathbf{y}}_j} \frac{\partial \hat{\mathbf{y}}_j}{\partial \mathbf{W}_j} \tag{2.13}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{b}_j} = \frac{\partial \mathscr{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \hat{\mathbf{y}}_{n-1}} \frac{\partial \hat{\mathbf{y}}_{n-1}}{\partial \hat{\mathbf{y}}_{n-2}} \cdots \frac{\partial \hat{\mathbf{y}}_{j+1}}{\partial \hat{\mathbf{y}}_j} \frac{\partial \hat{\mathbf{y}}_j}{\partial \mathbf{b}_j} \tag{2.14}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{k}_j} = \frac{\partial \mathscr{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \hat{\mathbf{y}}_{n-1}} \frac{\partial \hat{\mathbf{y}}_{n-1}}{\partial \hat{\mathbf{y}}_{n-2}} \cdots \frac{\partial \hat{\mathbf{y}}_{j+1}}{\partial \hat{\mathbf{y}}_j} \frac{\partial \hat{\mathbf{y}}_j}{\partial \mathbf{k}_j} \tag{2.15}$$

As a result, a learnable parameter set is updated with the gradient, $\mathbf{g}$, and a learning rate (i.e. step size in optimization terminology), ε, as provided in equation **(2.16)**.

$$\theta \leftarrow \theta - \varepsilon \mathbf{g} \tag{2.16}$$

Besides, some of the substantial considerations in training are overfitting and underfitting. Overfitting is a significant problem that reduces the generalization capability of a neural network. Generalization capability of a network can be evaluated with the generalization gap which is the difference between training and test errors. Thus, a neural network with a small training error and a small generalization gap can be said to be trained well. Furthermore, overfitting can be intuitively thought as the memorization of a training dataset. Low generalization capability restricts a neural network to make accurate predictions on data other than the training data. On the other hand, underfitting happens when a neural network is not capable of learning the features of a given data, adequately. In that case, the network cannot only perform accurately on test set but on training set as well. Moreover, model capacity is a crucial consideration in training to hinder fitting problems. To oversimplify, the capacity of

a network model is positively commensurate with the number of learnable parameters which has an impact on the model performance [43]. The selection of functions in a model has also an influence on the model capacity [1]. A demonstration of model capacity vs. model error is shared with Fig. 2.5. As it is seen, there is an optimum spot between the underfitting and overfitting zones. Hence, finding the neural network architecture with the optimal or near-optimal capacity makes the training process heuristic.

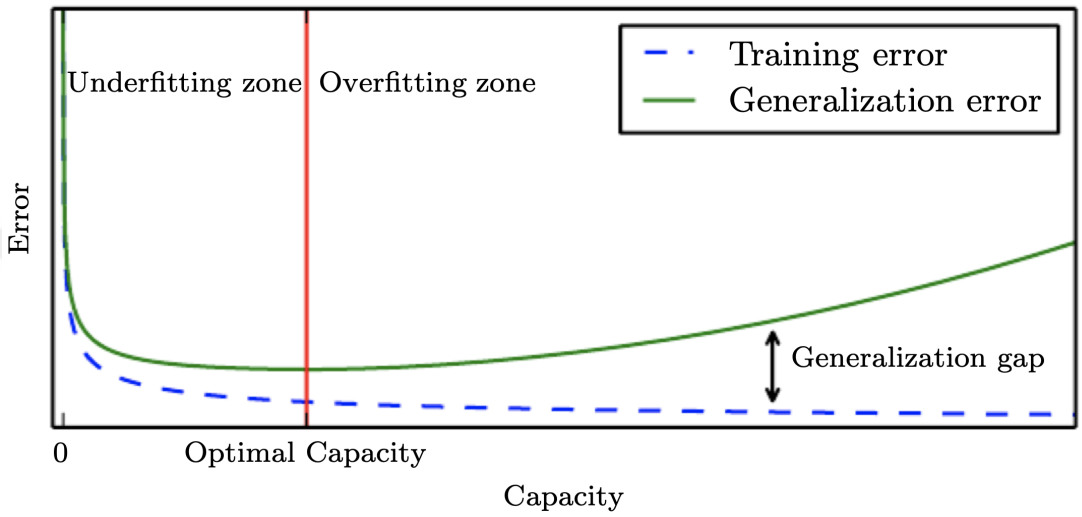

Figure 2.5 : Demonstration of underfitting and overfitting with error curves vs. model capacity [1].

As overfitting is defined, the limitation on equation **(2.11)** can be elaborated. Since it is an unconstrained problem, elements of learnable parameters are updated to perfectly fit on a dataset which makes a neural network prone to overfitting. In order to impede overfitting, a penalty norm namely parameter regularization is introduced. In this thesis, L_1 and L_2 parameter regularization schemes are utilized. L_1 and L_2 regularization respectively given in equations **(2.17)** and **(2.18)** forces learnable parameters to have elements with small values; thus, a perfect fit is prevented. These loss functions are also known as mean absolute error (MAE) and mean squared error (MSE), respectively.

$$\Omega(\theta) = ||\mathbf{W}|| + ||\mathbf{b}|| + ||\mathbf{k}|| \tag{2.17}$$

$$\Omega(\theta) = ||\mathbf{W}||_2^2 + ||\mathbf{b}||_2^2 + ||\mathbf{k}||_2^2 \tag{2.18}$$

Then, the optimization problem in equation **(2.11)** becomes a constrained problem by adding a penalty term as specified in equation **(2.19)**.

$$\min_{\theta} \ \mathscr{L}(\mathbf{y},\hat{\mathbf{y}}) + \lambda\Omega(\theta) \tag{2.19}$$

where λ is the regularization strength (i.e. penalty constant). Here, the regularization strength is a hyperparameter that controls the trade-off between getting low learnable parameters and fitting on data.

Moreover, several gradient-based algorithms have been being employed in neural network applications to solve the optimization problem in equation **(2.19)**. However, Adam [44] first-order stochastic gradient-descent optimization algorithm is the most appealed among all available methods by far. The reason is that Adam optimizer requires low computational effort and memory requirement. On top of that, optimization problems with a high number of parameters and/or data can be effectively solved by using Adam. The algorithm and mathematical foundations of Adam can be found in [44].

2.2 Data Pre-processing

This section elucidates the fundamentals of data pre-processing used in the thesis.

2.2.1 Normalization

In order to increase numerical stability during training, data standardization is a crucial step [45–47]. In this thesis, min-max, one of the most implemented normalization scheme, is used. Min-max normalization maps a given input, $\mathbf{x}$, to the range of [0,1] and given with equation **(2.20)**.

$$\bar{\mathbf{x}} = \frac{\mathbf{x} - \mathbf{x}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}} \tag{2.20}$$

where $\bar{\mathbf{x}}$ is the normalized dataset. Subscripts min and max respectively correspond to the minimum and the maximum value within the data array. A different variation of min-max normalization in equation **(2.21)** can also be used to transform an input vector to the range of [-1,1].

$$\bar{\mathbf{x}} = 2\frac{\mathbf{x} - \mathbf{x}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}} - 1 \tag{2.21}$$

For multivariable datasets, each variable in the data and target vectors is normalized in itself.

2.2.2 Dataset shuffling

Data shuffling is used to reorder a given data and target vectors based on the first dimension without disordering the data-target pair. There are research that show shuffling yields a better model performance [48, 49]. In [50], it is proved that shuffling provides convergence guarantee in non-convex optimization problems. Furthermore, it is stated that it speeds up the training process. The pseudo-code used in this thesis is provided in Algorithm 1 where N is the total number of data points, and $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ correspond to shuffled input and target arrays.

Algorithm 1 Data shuffling

Require: Data vector $\mathbf{x}$, Target array $\mathbf{y}$
  $P \leftarrow \{1, \cdots, N\}$
  $P \leftarrow \text{permute}(P)$
  $\tilde{\mathbf{x}} \leftarrow \mathbf{0}$
  $\tilde{\mathbf{y}} \leftarrow \mathbf{0}$
  for i $\leftarrow$ 1 to N **do**
    $j \leftarrow P_i$
    $\tilde{\mathbf{x}}_i \leftarrow \mathbf{x}_j$
    $\tilde{\mathbf{y}}_i \leftarrow \mathbf{y}_j$
  end for
  return $\tilde{\mathbf{x}}, \tilde{\mathbf{y}}$

2.2.3 Dataset splitting

Dataset splitting is a standard step in data pre-processing where the entire data is split into training, validation, and/or test sets with a certain ratio. Further information on the determination of splitting ratio can be found on [51]. The pseudo-code is provided in Algorithm 2.

Algorithm 2 Data splitting

Require: Data array $\mathbf{x}$, Target array $\mathbf{y}$, Training ratio $r_{train} \in \mathbb{R}^+$, Validation ratio $r_{val} \in \mathbb{R}^+$ where $r_{train} + r_{val} \leq 1$

$N_{train} \leftarrow \text{round}(r_{train} \times N)$

$N_{val} \leftarrow \text{round}(r_{train} + r_{val}) \times N)$

$N_{test} \leftarrow N - (N_{train} + N_{val})$

$\mathbf{x}_{\text{train}} \leftarrow \mathbf{x}_{1,\cdots,N_{train}}$

$\mathbf{x}_{\text{val}} \leftarrow \mathbf{x}_{N_{train},\cdots,N_{val+1}}$

$\mathbf{x}_{\text{test}} \leftarrow \mathbf{x}_{N_{val+1},\cdots,N}$

$\mathbf{y}_{\text{train}} \leftarrow \mathbf{y}_{1,\cdots,N_{train}}$

$\mathbf{y}_{\text{val}} \leftarrow \mathbf{y}_{N_{train},\cdots,N_{val}}$

$\mathbf{y}_{\text{test}} \leftarrow \mathbf{y}_{N_{val},\cdots,N}$

return $\mathbf{x}_{\text{train}}, \mathbf{x}_{\text{val}}, \mathbf{x}_{\text{test}}, \mathbf{y}_{\text{train}}, \mathbf{y}_{\text{val}}, \mathbf{y}_{\text{test}}$

2.2.4 Dataset minibatching

A dataset may contain millions of data points; hence, it is not always to process entire dataset in a training iteration due to memory limitations. In order to overcome this impediment, a subset, $\mathbf{X_b}$, of a dataset, $\mathbf{X}$, is processed in a training iteration where $\mathbf{X_b} \subset \mathbf{X}$. Feeding all minibatches to the network during a training is called an epoch. Then, it is clear that learnable network parameters of a network are optimized with a small number of data in each training iteration. Thus, each gradient computation based on a small batch is an estimation of the gradient of the entire dataset. Under this circumstance, a gradient descent optimization becomes stochastic gradient descent optimization process. A mathematical representation on gradient estimation is given in equation **(2.22)** with a small adjustment to equation **(2.12)**. As it is seen the derivative of the loss with respect to all dataset (lefthand side of the equation) is computed by using a small partition of the dataset $\mathbf{X}_b$ (righthand side of the equation). It can be asserted that what percentage of an entire dataset that a single batch of data corresponds to is an important parameter in gradient computation so does in training. At this point, batch size which is the number of data points in a batch can be considered as a hyperparameter. The higher the batch size is, the more accurate gradients are computed in each iteration yet the more memory is required to process that amount of data as well.

$$\frac{\partial \mathscr{L}}{\partial \mathbf{X}} \approx \frac{\partial \mathscr{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \hat{\mathbf{y}}_{n-1}} \frac{\partial \hat{\mathbf{y}}_{n-1}}{\partial \hat{\mathbf{y}}_{n-2}} \cdots \frac{\partial \hat{\mathbf{y}}_2}{\partial \hat{\mathbf{y}}_1} \frac{\partial \hat{\mathbf{y}}_1}{\partial \mathbf{X}_b} \tag{2.22}$$

2.2.5 Representation of physical flow fields for neural networks

A solution field of a flow problem is composed of either node or cell data. These data points might belong to a structured or an unstructured grid and must be represented with matrix/tensor form to enable them to be processed by a convolutional neural network. Regardless of the problem, grid type and shape, solution variables of a 2-dimensional case can be transferred on a Cartesian grid to be represented with the matrix notation. This approach is used in [19, 52]. Each node on the Cartesian grid can be thought as a matrix element. For instance, a 2-dimensional pressure coefficient field interpolated on a 2-dimensional 64-by-64 Cartesian grid can be represented with a matrix, $\mathbf{x} \in \mathbb{R}^{64\times64}$. To be specific, it may not be necessary to interpolate an entire flow field but considering only a substantial part that can be sufficient as provided in Fig. 2.6. In the figure, the bounded interpolated domain is located in a way to encompass most of the gradient/valuable information as seen.

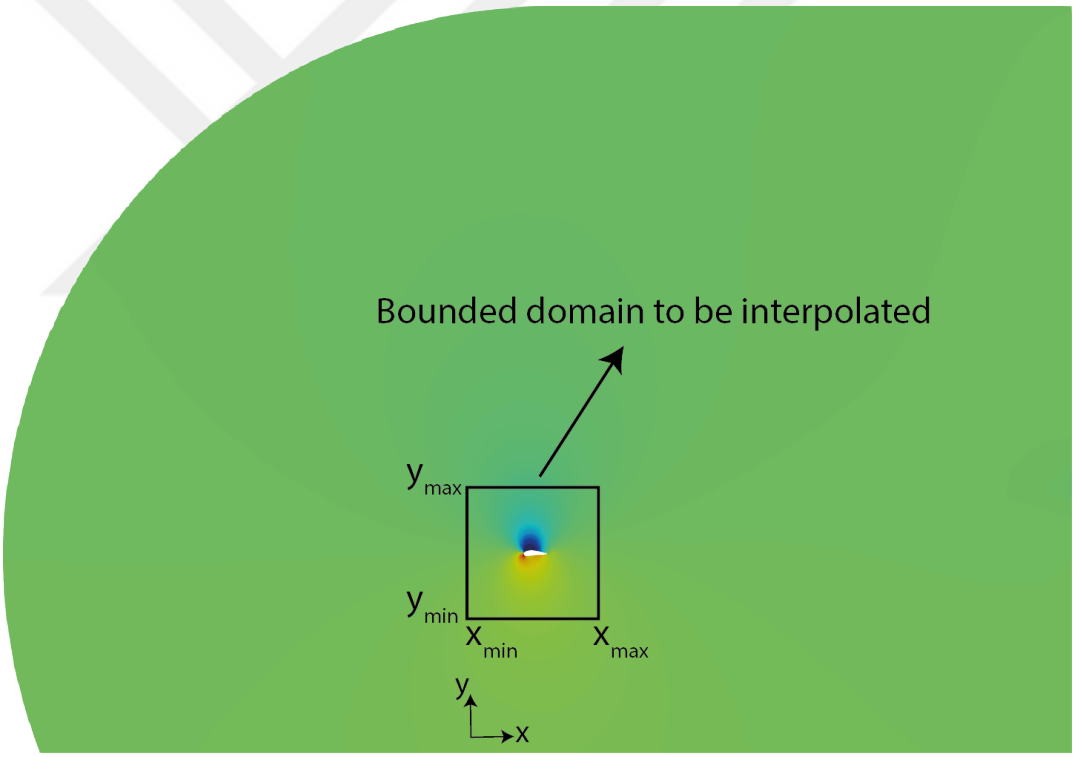

Figure 2.6 : A bounded interpolation domain around an airfoil.

Furthermore, one of the other advantages of interpolating 2-dimensional outputs is that it can dramatically reduce the number of data points that constitutes a solution domain. Thus, tens of thousands of grid data points can be represented with a few thousands of data that can be thought as a low-fidelity version of the original data. In summary, interpolation approach is applicable for a wide range of problems and

it offers dimensionality reduction which might be quite useful for high-dimensional datasets. To illustrate, interpolation of a flow field on a Cartesian grid is given in Fig. 2.7. As it can be seen on the figure, it embodies apparent pressure gradients such as at the stagnation point on the leading edge, and supersonic region over the upper surface. However, the geometry gets obscure after the interpolation. In fact, it is expected not to be an issue to lose geometrical features as gradients on a flow field are directly associated with geometries; thus, they are some kind of representations of shapes within a flow field.

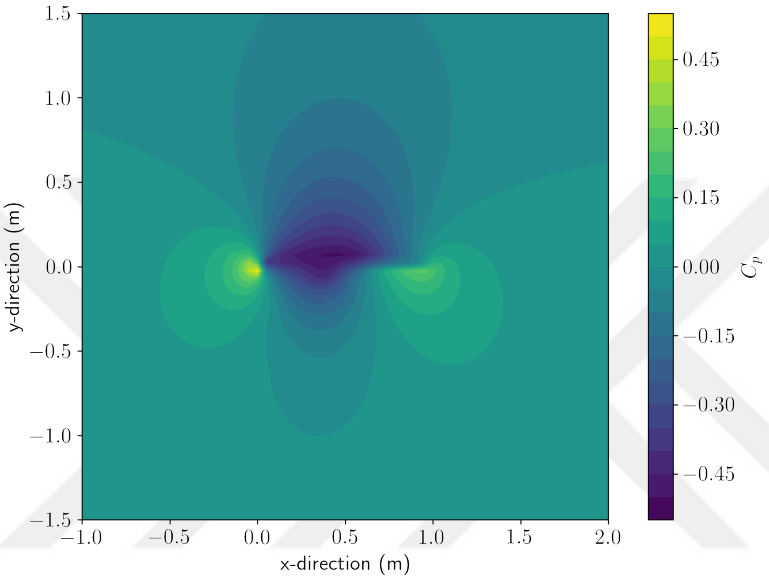

Figure 2.7 : An interpolated pressure coefficients field around a supercritical airfoil.

Additionally, a pre-processing tool is developed in Python environment to make 2-dimensional solver outputs suitable for training and test of convolutional neural networks. It is intended to extract, read, and convert data from solution files using open-source ParaView [53] post-processing tool in the data extraction part. This is achieved with a combination of different functions. The process of flow field representation with the developed tool is depicted in Fig. 2.8.

Firstly, a set of *.csv* files, each corresponds to a certain solution file, and contains solution variables and x, y, z coordinates, are created using ParaView. Then, a Python module reads variables and coordinates from *.csv* files, and interpolates on a user-defined Cartesian grid using one of the interpolation schemes which are nearest neighbour, linear, and cubic spline. In addition, dataset normalization with min-max, z-score, and decimal normalization schemes are also available within the tool. Besides,

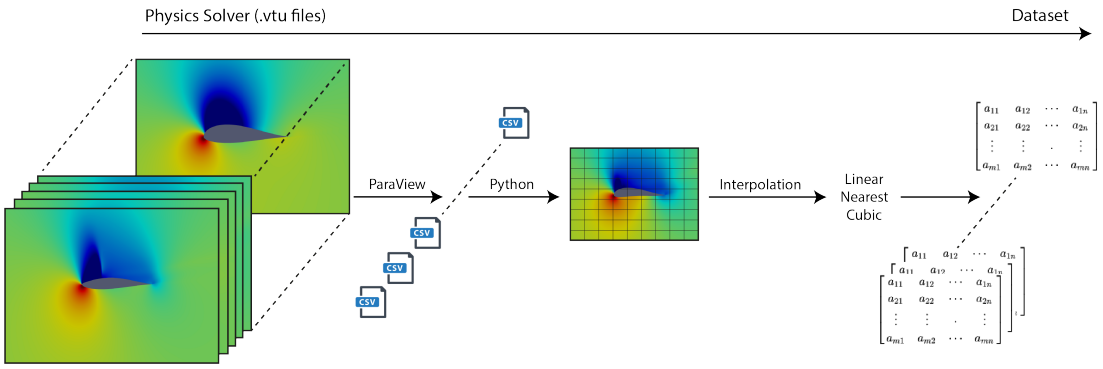

Figure 2.8 : The flow of the dataset preparation.

dataset arrangement algorithms, shuffling, splitting, and minibatching, are embedded in a different Python class. Lastly, a visualization class is developed to visualize the data during pre-processing so as to prevent any numerical errors before the training. Hence, the developed tool offers end-to-end pre-processing for representation of 2-dimensional solver outputs.

The efficacy of interpolated 2-dimensional physical solver outputs in use of convolutional neural networks is investigated with a parameter-varying 2-dimensional flow problem using the developed pre-processing tool as is presented in the following section.

2.2.5.1 The investigation of interpolated flow fields in convolutional neural networks

An informative preliminary study is performed to investigate the effects of different interpolation methods and grid sizes on a CNN prediction performance. This study focuses on a geometry and flow dependent airfoil problem where the objective is to predict drag and lift coefficients of airfoils using flow fields. The considered problem is 8-dimensional where 6 of them define an airfoil geometry as given in Fig. 2.9 and the rest are flow parameters, namely, Mach number and angle of attack. The flow parameters are respectively taken as $M \in [0.4, 0.7]$ and $\alpha \in [8°, 16°]$. 4 equidistant Mach numbers and 3 equidistant angle of attack values are sampled. Thus, in total, 12 different flow conditions are determined. Moreover, an in-house parametric airfoil generation tool developed in AeroMDO Multidisciplinary Design Optimization Laboratory is employed to obtain 64 unique airfoils. SU2 is utilized to solve Reynolds-Averaged-Navier-Stokes (RANS) equations with Spallart-Allmaras

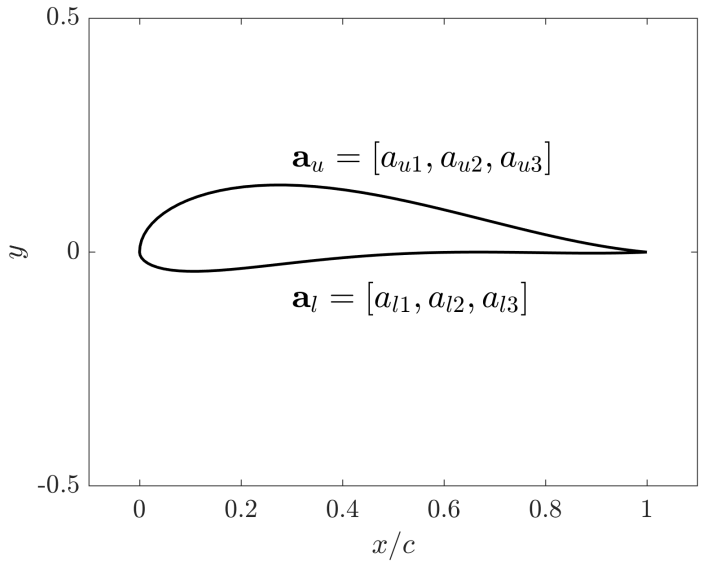

Figure 2.9 : Airfoil defining parameters for upper and lower surfaces.

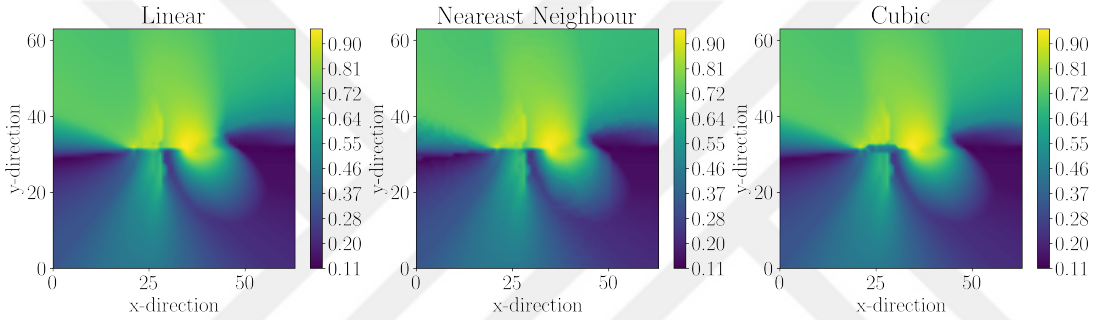

Figure 2.10 : Normalized C_p distributions of the same flow field obtained with different interpolation methods.

(SA) turbulence model. Each airfoil is analyzed under all 12 flow conditions which yields 768 viscous flow solutions. 85%-15% ratio is used to split the flow solutions as training and test sets. Furthermore, Cartesian grid sizes of 64×64, 128×128, and 256×256 are considered with 3 different interpolation methods and min-max data and target normalization. Normalization results with all considered interpolation methods on a 64-by-64 Cartesian grid of a randomly selected airfoil within the dataset is depicted in Fig. 2.10. It is experienced that the computational cost of each method in an increasing order is the nearest neighbour, linear, and cubic interpolation where the interpolation times of a single flow field are respectively 0.03, 1, and 1.9 seconds.

The established neural network architecture consists of 4 convolution blocks where each block involves a convolution layer, ReLU, and batch normalization. A dense block with output size of 2 is used in order to obtain C_D and C_L values. The convolution

blocks and the dense block are connected with an average pooling layer. The detailed network architecture for 64×64 input size is tabulated in Table 2.1.

Table 2.1 : The CNN architecture for input data with the size of 64×64.

Layer	**Filter/Kernel/Stride**
Conv. - ReLU - Batch Norm.	4/4/1
Conv. - ReLU - Batch Norm.	8/4/2
Conv. - ReLU - Batch Norm.	16/4/2
Conv. - ReLU - Batch Norm.	32/4/3
Average Pooling	-/4/3
Output Layer	**Input dimension/Output dimension**
Linear - ReLU - Linear	(32/8), - , (8/2)

For training, Adam optimizer is implemented with L_1 loss function. A constant learning rate of 10^{-5} is used for total 5000 epochs. The network is optimized on Nvidia Quadro P4000 8GB GPU. The model performances in terms of mean absolute error (MAE) for all training conditions are listed in 2.2. Test results show that acceptable drag and lift coefficient predictions obtained for all cases although the grid size of 128×128 gives the best results for all interpolation methods. It can be concluded that the usage interpolated 2-dimensional flow fields can produce satisfactory results in CNN-based predictive modeling. In addition, increasing the flow field resolution does not affect the model performance positively for all cases. Lastly, the linear interpolation is the least variant with changing grid size where test losses are comparable.

Table 2.2 : Training and test loss values of different datasets.

Interpolation Method	**Grid Size**	**Training MAE**	**Test MAE**
Linear	64×64	0.0067	0.0148
	128×128	0.0073	0.0116
	256×256	0.0070	0.0152
Nearest	64×64	0.0065	0.0345
	128×128	0.0060	0.0196
	256×256	0.0123	0.0287
Cubic	64×64	0.0087	0.0330
	128×128	0.0051	0.0196
	256×256	0.0068	0.0230

64-by-64 grid size with linear interpolation is found accurate enough and utilized for the further parts of the study presented in this thesis.

2.3 Multi-fidelity Analysis

Multi-fidelity analysis methods are mostly adopted to diminish the computational burden of a set of high-fidelity simulations. These methods fuse a low number of expensive and physically more accurate data with a high number of cheap and less accurate data. In [54], multi-fidelity methods are categorized as adaptation, fusion and filtering. In the context of the thesis, fusion-based multi-fidelity approach is discussed. Fusion principle requires firstly to obtain low-fidelity and high-fidelity data. Afterwards, data with different fidelity levels are fused with a scheme such as co-Kriging, multi-fidelity polynomial chaos expansion, and multi-fidelity neural networks. A general expression for multi-fidelity comprehensive correction model is given with equation **(2.23)**.

$$y_H(x) = \rho(x) y_L(x) + \delta(x) \tag{2.23}$$

where y_H and y_L respectively correspond to exact HF and LF data points. $\rho(x)$ and $\delta(x)$ respectively denote the surrogate models for multiplicative and additive corrections. It should be noted that this approach can merely handle linearly correlated HF and LF data. A further information on multi-fidelity methods can be found in [55].

2.3.1 Multi-fidelity deep neural networks

Multi-fidelity deep neural networks (MFDNN) have been recently developed data-driven predictive modeling method and first introduced in [26]. In this paper, a generalized autoregressive method for multi-fidelity analysis using scalar input and output is proposed as in equation **(2.24)**.

$$y_H(x, y_L) = \mathscr{F}_l(x, y_L) + \mathscr{F}_{nl}(x, y_L) \tag{2.24}$$

where $\mathscr{F}_l(x, y_L)$ and $\mathscr{F}_{nl}(x, y_L)$ are respectively the linear and nonlinear correlation networks. To the best of the author's knowledge, weighting these correlation networks is first proposed in [43]. The weighted autoregressive scheme is as in equation **(2.25)**.

$$y_H(x, y_L) = \omega \mathscr{F}_l(x, y_L) + (1 - \omega) \mathscr{F}_{nl}(x, y_L) \tag{2.25}$$

where ω is the linear correlation weight. In general, the type and strength of a correlation between two arbitrary engineering dataset may not be known beforehand. Thus, ω is a hyperparameter to be tuned.

The proposed MFDNN architecture in [43] is illustrated in Fig. 2.11. It has 3 distinct

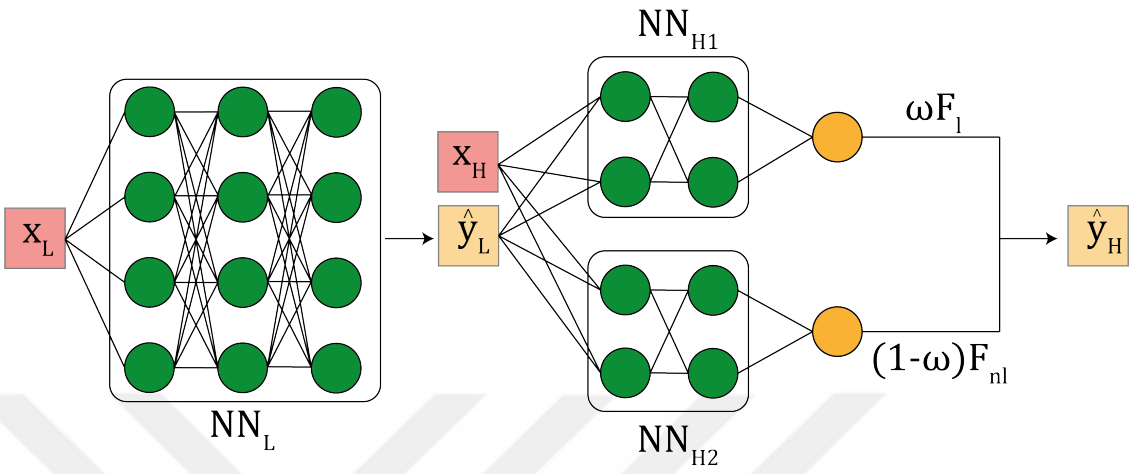

Figure 2.11 : A generic multi-fidelity deep neural network architecture.

neural networks which are low-fidelity deep neural network (LFDNN), NN_L, linear correlation network, NN_{H_1}, and nonlinear correlation network, NN_{H_2}. To clarify, NN_L is used to approximate low-fidelity outputs with a given set of low-fidelity input. In addition, NN_{H_1} and NN_{H_2} are implemented to estimate the correlation between low-fidelity and high-fidelity datasets. These correlation networks accept a combination of high-fidelity inputs and low-fidelity outputs as an input. Let a high-fidelity input be $\mathbf{x_H} \in \mathbb{R}^{n_H}$ and low-fidelity prediction be $\hat{\mathbf{y}}_\mathbf{L} \in \mathbb{R}^{m_p}$. Then the input of correlation networks becomes $\mathbf{x_C} \in \mathbb{R}^{n_H+m_p}$. Hence, high-fidelity inputs and low-fidelity predictions should be in a proper shape to be concatenated and feed to correlation networks.

In the literature, MFDNNs are constructed with only linear layers and hyperbolic tangent functions respectively given with equations **(2.2)** and **(2.4)** [26–28,43,56]. NN_L and NN_{H_2} have similar composition including linear layers and activation functions while NN_{H_1} only consists of linear layers due to the fact that it is supposed to catch the linear correlation between high- and low-fidelity datasets.

The training of MFDNN is based on the both low-fidelity and high-fidelity predictions. The loss function to be minimized is computed as in equation **(2.26)**.

$$\mathscr{L}(y_L, y_H, \hat{y}_L, \hat{y}_H, \theta) = \mathscr{L}(\mathbf{y}_L, \hat{\mathbf{y}}_L) + \mathscr{L}(\mathbf{y}_H, \hat{\mathbf{y}}_H) + \lambda\Omega(\theta) \tag{2.26}$$

For numerical predictions discussed through this section, mean squared error loss function is useful to evaluate the prediction error. Then, equation **(2.26)** becomes the expression in **(2.27)** where m_p is the number of predicted data points, $\hat{\mathbf{y}}_L \in \mathbb{R}^{m_p}$ and $\hat{\mathbf{y}}_H \in \mathbb{R}^{m_p}$.

$$\mathscr{L}(y_L, y_H, \hat{y}_L, \hat{y}_H, \theta) = \frac{1}{m_p}\left(||\mathbf{y}_L - \hat{\mathbf{y}}_L||_2^2 + ||\mathbf{y}_H - \hat{\mathbf{y}}_H||_2^2\right) + \lambda\Omega(\theta) \tag{2.27}$$

For this thesis, an MFDNN code is developed in Python environment using PyTorch deep learning and optimization libraries [57]. So as to validate the developed code and explore the method, analytical multi-fidelity Forrester, Borehole, and Branin functions are taken into account and elaborated in the following sections. In all of the examples 10 HF, 50 LF equidistantly spaced samples are used where both input and target sets are normalized to the range of [-1,1].

2.3.1.1 Application: forrester function

The multi-fidelity Forrester function [58] is a univariate set of functions given in equation **(2.28)**.

$$\begin{aligned} f_H(x) &= (6x-2)^2 \sin(12x-4) \\ f_L(x) &= A f_H(x) + B(x-0.5) - C \end{aligned} \tag{2.28}$$

where f_H and f_L refer to high-fidelity and low-fidelity functions, and $A = 0.5$, $B = 10$, $C = -5$, and $x \in [0,1]$. The created MFDNN architecture is tabulated in Table 2.3 where Tanh is the shorthand notation of hyperbolic tangent activation layer and width of a layer weight is given with number of features. The results are shared with Fig. 2.12.

Table 2.3 : The MFDNN architecture used to predict MF Forrester function.

Network	Layer	Number of Features
NN_L	Linear-Tanh	20
	Linear-Tanh	20
	Linear-Tanh	20
	Linear	20
NN_{H_1}	Linear	10
	Linear	10
NN_{H_2}	Linear-Tanh	20
	Linear-Tanh	20
	Linear-Tanh	20
	Linear	20

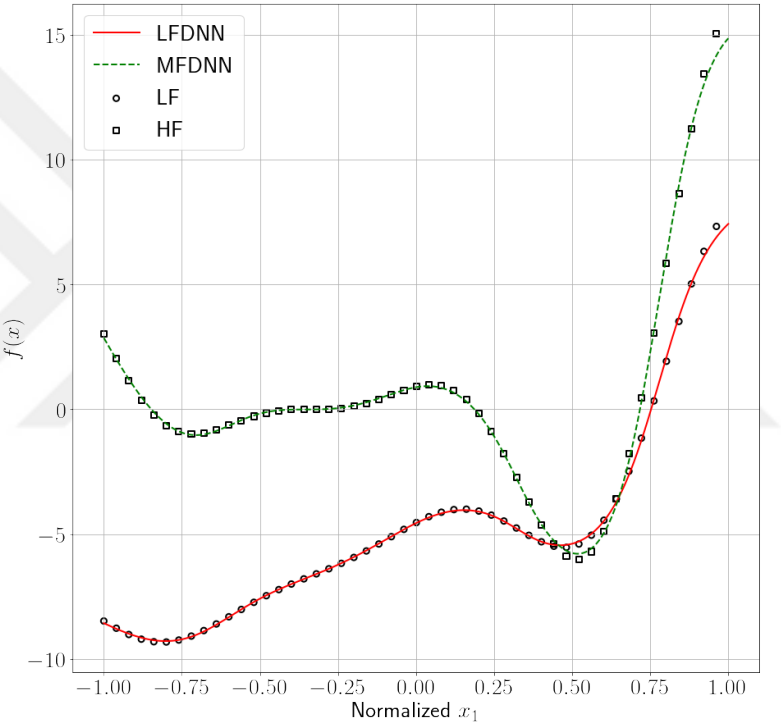

Figure 2.12 : The comparison of MFDNN prediction of MF Forrester function with the exact functions.

Results show that the MFDNN can accurately make low-fidelity and multi-fidelity predictions.

2.3.1.2 Application: branin function

The multi-fidelity Branin function [59] is a bivariate set of functions given in equation **(2.29)**.

$$
\begin{aligned}
f(x_1,x_2) &= 10+\left[x_2-5.1\times\frac{x_1^2}{4\pi^2}+\frac{5x_1}{\pi}-6\right]^2+10\cos(x_1)\left[1-\frac{1}{8\pi}\right]\\
f_H(x_1,x_2) &= f(x_1,x_2)-22.5x_2\\
f_L(x_1,x_2) &= f(0.7x_1,0.7x_2)-15.75x_2+20(0.9+x_1)^2-50
\end{aligned}
\tag{2.29}
$$

where f, f_H and f_L respectively refer to base, high-fidelity, and low-fidelity functions. The input domain of the Branin function is $x_1 \in [-5,10]$ and $x_2 \in [0,15]$. The MFDNN architecture used in prediction of MF Branin function is tabulated in Table 2.4. Fig. 2.13 shows the results. Circle and square markers correspond to exact low-fidelity and high-fidelity functions.

Table 2.4 : The MFDNN architecture used to predict MF Branin function.

Network	Layer	Number of Features
NN_L	Linear-Tanh	40
	Linear-Tanh	40
	Linear	40
NN_{H_1}	Linear	10
	Linear	10
NN_{H_2}	Linear-Tanh	10
	Linear-Tanh	10
	Linear-Tanh	10
	Linear	10

The MFDNN is capable of predicting both fidelity levels of the Branin function.

2.3.1.3 Application: borehole function

The multi-fidelity Borehole function [60], is an 8-dimensional function with a strong linear correlation. High- and low-fidelity Borehole functions are given in equations **(2.30)** and **(2.31)**, respectively.

$$
f_H(x) = \frac{2\pi T_u(H_u-H_l)}{\ln(r/r_w)\left(1+\dfrac{2LT_u}{\ln(r/r_w)\,r_w^2K_w}+\dfrac{T_u}{T_l}\right)} \tag{2.30}
$$

$$
f_L(x) = \frac{5T_u(H_u-H_l)}{\ln(r/r_w)\left(1.5+\dfrac{2LT_u}{\ln(r/r_w)\,r_w^2K_w}+\dfrac{T_u}{T_l}\right)} \tag{2.31}
$$

The domains of input variables are provided in Table 2.5. The comparison of MFDNN predictions with the exact functions are illustrated in Fig. 2.14.

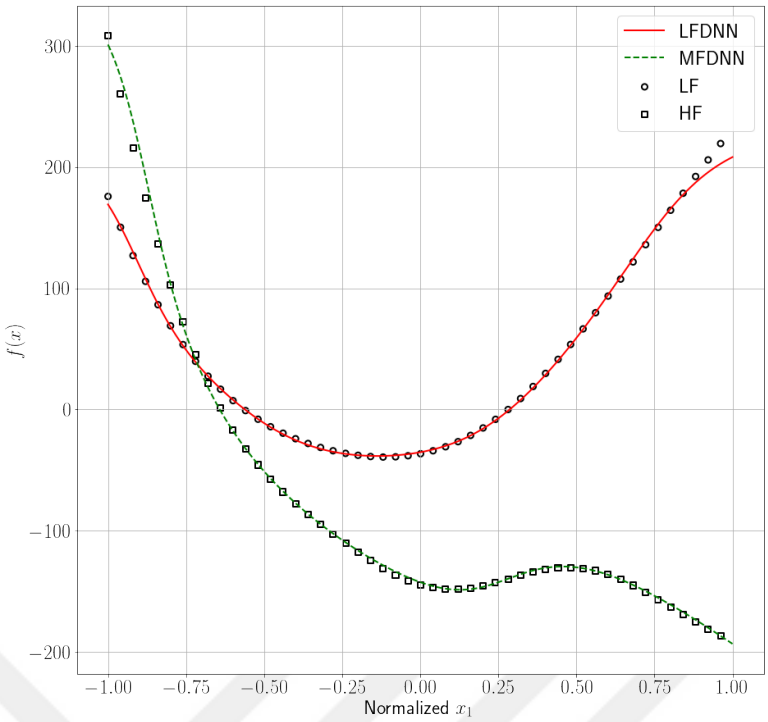

Figure 2.13 : The comparison of MFDNN prediction of MF Branin function with the exact functions.

Table 2.5 : The lower and upper bounds of input variables.

Variable	Domain
r_w	[0.05, 0.15]
r	[100, 5000]
T_u	[3700, 115600]
H_u	[990, 1100]
T_l	[63.1, 116]
H_l	[700, 820]
L	[1120, 1680]
K_w	[9855, 12045]

As a result, the MFDNN proves its efficacy on a relatively higher-dimensional problems in comparison of the previous problems.

2.3.1.4 The investigation of the correlation weight

The correlation weight given in equation **(2.25)** is an optimization parameter that need to be tuned as the correlation between the high- and low-fidelity datasets may not be known in advance. The correlation weight, ω, is not broadly investigated in the literature. Authors of [26–28, 43, 56] did not clearly state the tuning of correlation weight.

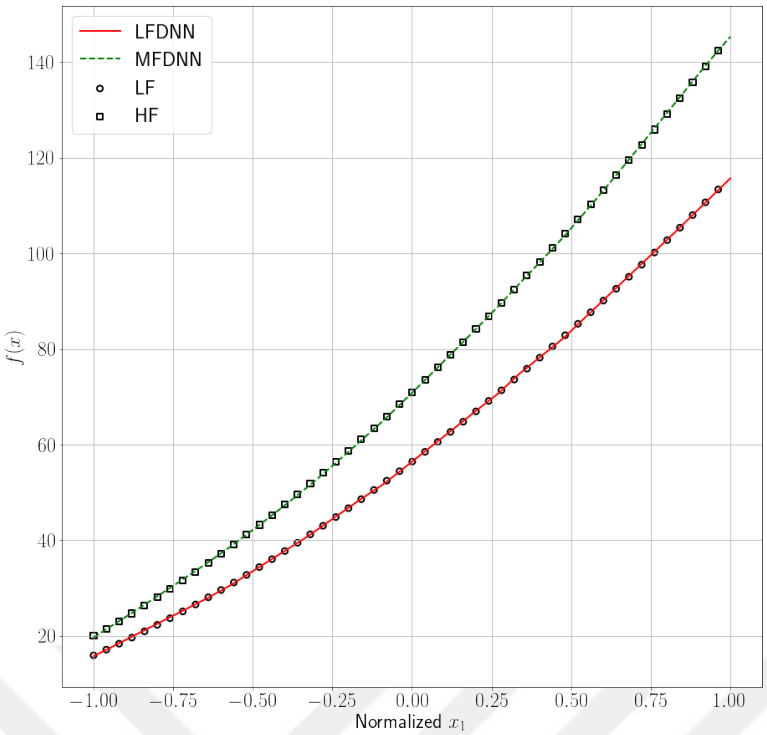

Figure 2.14 : The comparison of MFDNN prediction of MF Borehole function with the exact functions.

Moreover, the learning of correlation weight may be fallacious since training of a multi-fidelity neural network may not guarantee to learn the linear and nonlinear correlation functions of a multi-fidelity dataset and their weights separately. Contrarily, they would be trained such that they try to yield the minimum loss function in a coupled way. In this study, the optimization of correlation weight is investigated with with the same analytical multi-fidelity functions considered before: Forrester function, Borehole function, and Branin function. To be clear, the dependency of the optimized correlation weight to the initial correlation weight is investigated similar to a grid search strategy. The correlation weight is defined as a network parameter like network weights, biases, and kernels where it is optimized simultaneously with the other network parameters. Furthermore, this study requires each multi-fidelity network, and their training process to be the same including dataset, hyperparameters, loss function, and optimization scheme as well as the initial model state (the initial weight and biases of networks). Therefore, the variation in the training process is reduced to the initial correlation weight, only.

The set of initial correlation weight taken into account is $\omega_0 = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. So, there are 5 different initial correlation weight values which corresponds to

5 MFDNN trainings with aforementioned multi-fidelity analytical functions. For multi-fidelity Forrester, Borehole, and Branin functions, the employed MFDNN architectures are respectively as given in Section 2.3.1.1, 2.3.1.2, and 2.3.1.3. In addition to the architectures, the same hyperparameter set, loss function, and optimizer stated in these sections are utilized. The trainings are considered converged when the total RMSE of predictions is lower than 10^{-4}. If the convergence criterion could not be satisfied in 3500 epochs, the training stops and the results obtained at the end are used.

The predicted and exact $\hat{y}_L$ vs. $\hat{y}_H$ curves of all functions for the set of initial correlation weights are depicted in Fig. 2.15, 2.16, 2.17 where ω_f denotes the optimized correlation weight.

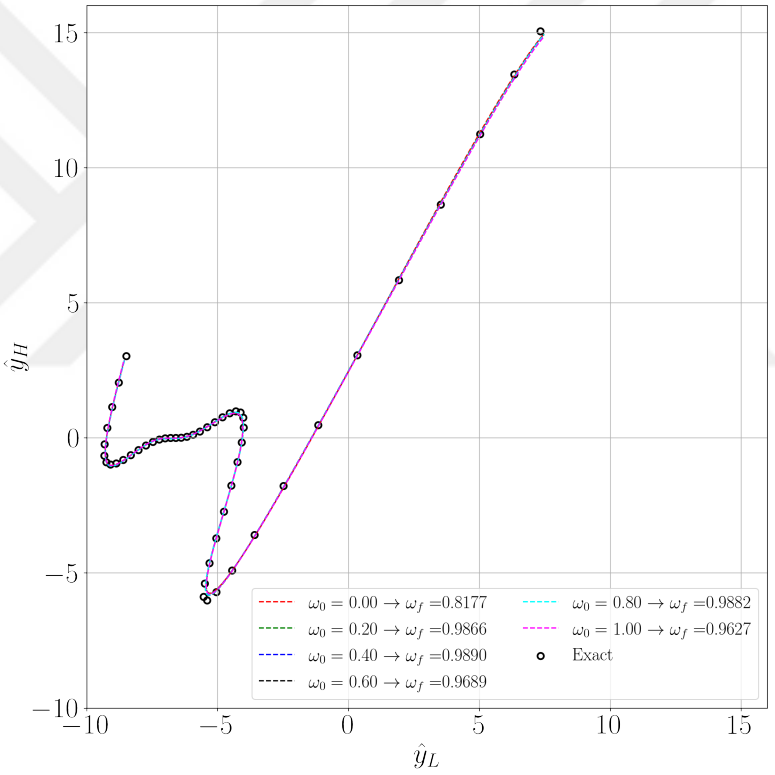

Figure 2.15 : The correlation curves of the multi-fidelity Forrester function.

Interestingly, all of the MFDNNs can make quite robust predictions of both low- and high-fidelity functions excluding the case of $\omega_0 = 1$ for Branin function. Nonetheless, the optimized value of the correlation weight is contingent upon the initial correlation weight and changes dramatically across the trainings especially for the Borehole and Branin functions. The most obvious case is the predictions of Borehole function which has a linear correlation between the high-fidelity and low-fidelity functions as is seen

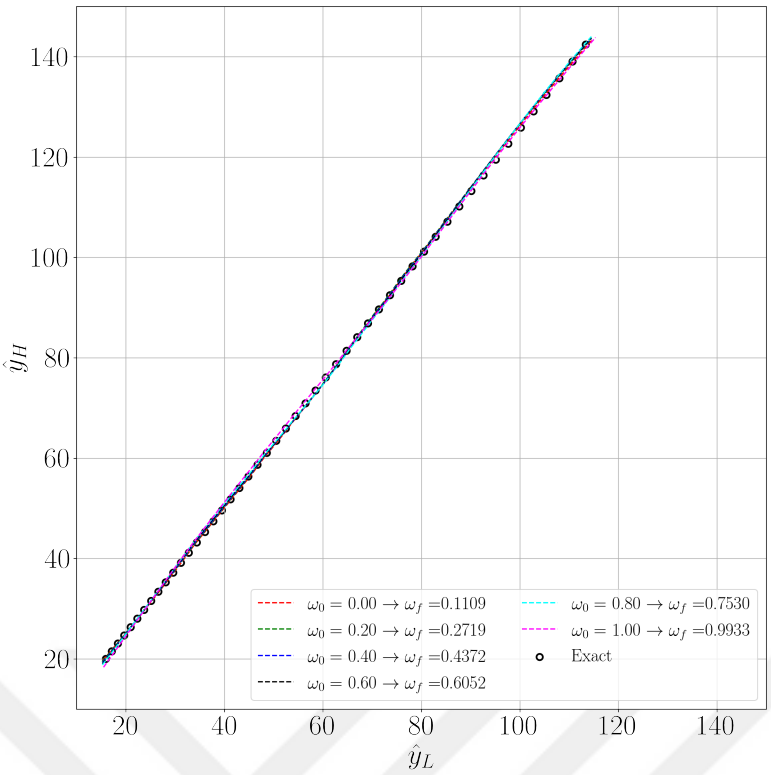

Figure 2.16 : The correlation curves of the multi-fidelity Borehole function.

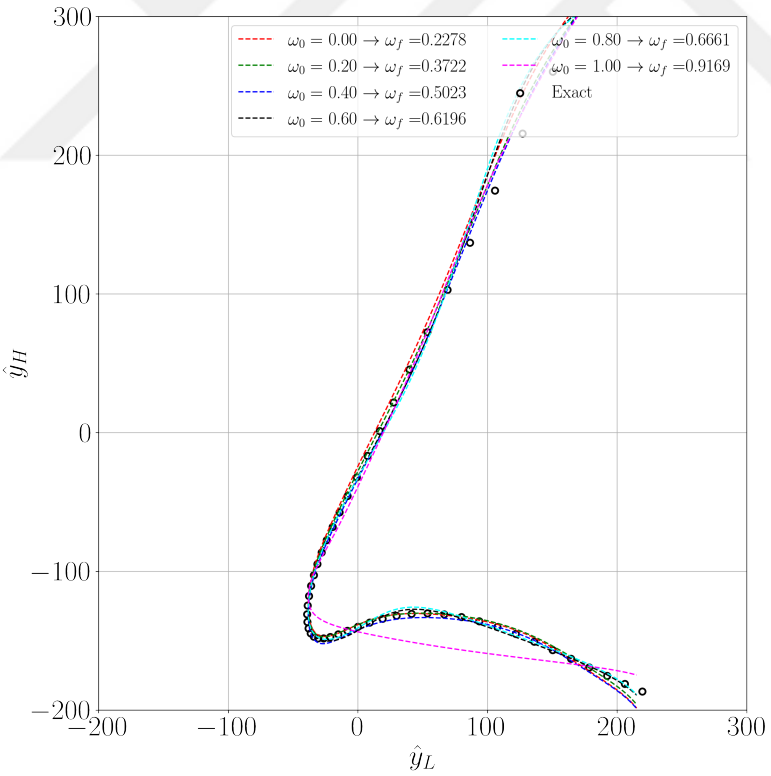

Figure 2.17 : The correlation curves of the multi-fidelity Branin function.

on Fig. 2.16. As ω is the strength of linear correlation, it is expected to have a value closer to 1 if the network approximates the correlation weight accurately. However,

the tuned correlation weight, ω_f, in the case of Borehole function varies between the lower bound of 0.11 and upper bound of 0.99. Therefore, it does not converge to a value near 1.

As a consequence, it can be asserted that, for the considered cases, the optimization of the correlation weight does not make the network to learn the exact correlation weights which are not known in advance. Hence, the correlation weight and the correlator networks are jointly trained to make accurate multi-fidelity predictions instead of training the correlation functions and the correlation weight, separately. Based on the findings, the correlation weight which has no explicit positive effect on learning is omitted from the autoregressive function given in equation **(2.25)** and the form provided in equation **(2.24)** is used for the rest of the study.

2.3.2 The modified multi-fidelity deep neural networks

As it is explained in Section 2.3.1, high-fidelity inputs and low-fidelity predictions should have suitable shapes to be stacked on top of each other so they can be given to the correlation networks. This condition restricts the usage of high-dimensional high-fidelity inputs and low-dimensional low-fidelity predictions. To the best of author's knowledge, research articulating the proper vector sizes for concatenation does not exist in the literature. Mathematically, let a high-fidelity input be $\mathbf{x_H} \in \mathbb{R}^{n_H}$ and a low-fidelity prediction be $\hat{\mathbf{y}}_\mathbf{L} \in \mathbb{R}^{m_p}$ where $m_p << n_H$. It is experienced and presented in the Section 3 that the correlation networks cannot learn the relation between the high-fidelity input and low-fidelity prediction as n_H is a quite higher value than m_p. The main reason of this issue is out of scope of this thesis study.

A workaround for the considered problem, a fully-connected encoder is incorporated into the MFDNN architecture. As it is discussed, the main purpose of the encoder is to map a high-dimensional high-fidelity input to a lower-dimensional subspace such that the original size of the input becomes closer to the size of low-fidelity predictions. The architecture of the modified-MFDNN is given in Fig. 2.18. The only difference of the modified-MFDNN from the MFDNN is a sub-network named NN_E. NN_E is a fully-connected encoder to transform $\mathbf{x_H} \in \mathbb{R}^{n_H}$ to an encoded vector $\mathbf{x_{H,E}} \in \mathbb{R}^{n_{H,E}}$ where $n_{H,E} \approx m_p << n_H$. Thus, the input of correlation networks becomes $\mathbf{x_C} \in \mathbb{R}^{n_{H,E}+m_p}$ instead of $\mathbf{x_C} \in \mathbb{R}^{n_H+m_p}$. As a result, the training and test

experiments presented under Section 3 show that the modified-MFDNN performs better than MFDNN.

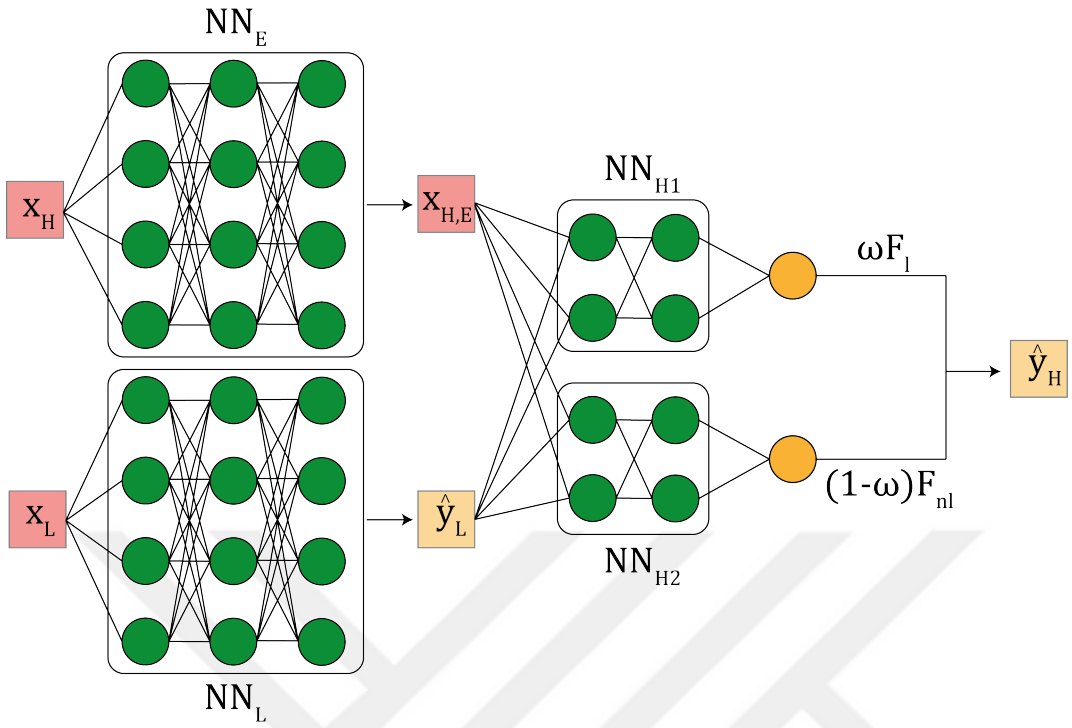

Figure 2.18 : A generic architecture of the modified-MFDNN.

Moreover, the size of the transformed high-fidelity input $n_{H,E}$ is a hyperparameter. An appropriate size should be selected to extract the latent features of a given high-dimensional high-fidelity input without the loss of information. The influence of latent vector size on the modified-MFDNN performance is given under Section 3.

2.3.3 Multi-fidelity convolutional neural networks

As it is mentioned before, the MFDNN uses input data in a scalar or vector form which may not be suitable for processing data with more than 1-dimension. As higher dimensional data must be vectorized to obtain a 1-dimensional data array. This process gives rise to the loss of information on the input data so does on the neural network performance. An illustration of the vectorization of a flow field around an airfoil is given in Fig. 2.19. As seen on the figure, highly correlated regions over the upper surface are bifurcated and placed far away from each other which disrupts the locality of the flow field.

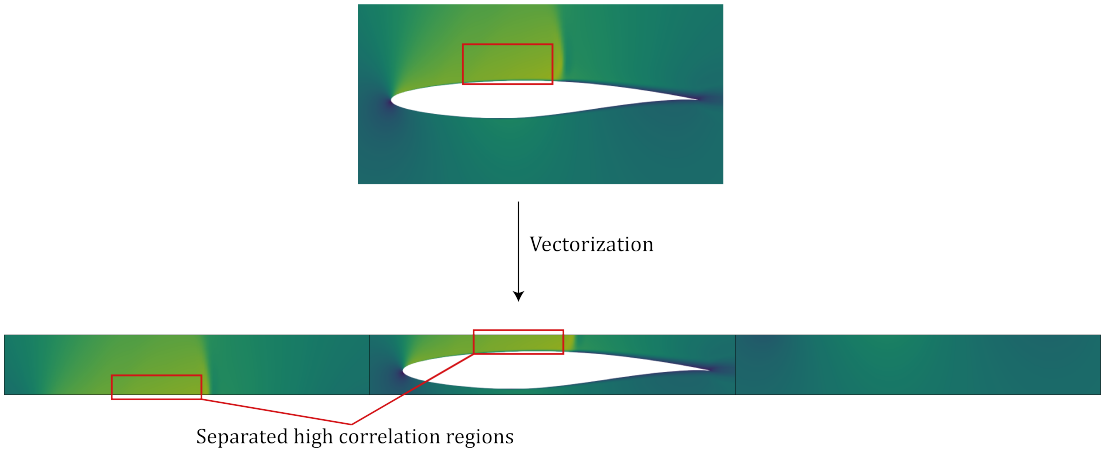

Figure 2.19 : A representation of the vectorization of a flow field around an airfoil.

To address this drawback, a coalescence of existing methods, MFDNN and CNN, is proposed in this thesis specifically for multi-fidelity predictive modeling of 2-dimensional flow fields.

The autoregressive model defined in equation **(2.25)** is valid for this method as well. The multi-fidelity convolutional neural network architecture proposed in this thesis is demonstrated in Fig. 2.20. In the MFDNN, multi-fidelity predictions require the high-fidelity inputs and low-fidelity predictions. Since both inputs and predictions are either a scalar or a vector, they can be concatenated. Thus, these two arrays are stacked and given to the correlation networks to estimate high-fidelity results. On the other hand, a new requirement arises with the usage of 2-dimensional numerical data in MFCNN. Unlike MFDNN, as it is discussed in Section 2.3.2 high-fidelity inputs of MFCNN are flow fields represented with 4-dimensional tensors (including batch, channel, height, and width) and cannot be directly stacked with any arbitrary low-fidelity predictions unless inputs and predictions have the same dimensions and sizes. Thus, high-fidelity inputs should be either downsampled by an encoder or upsampled by a decoder, accordingly.

Alike MFDNN, MFCNN comprises low-fidelity estimator, NN_L, linear correlator, NN_{H_1}, and nonlinear correlator, NN_{H_2}. Instead of the combination of affine and activation layers, NN_L of MFCNN is concatenation of convolution and activation layers. On top of that, MFCNN brings an additional sub-network, namely, NN_E. NN_E is either an encoder with convolution layers or a decoder with transpose convolution

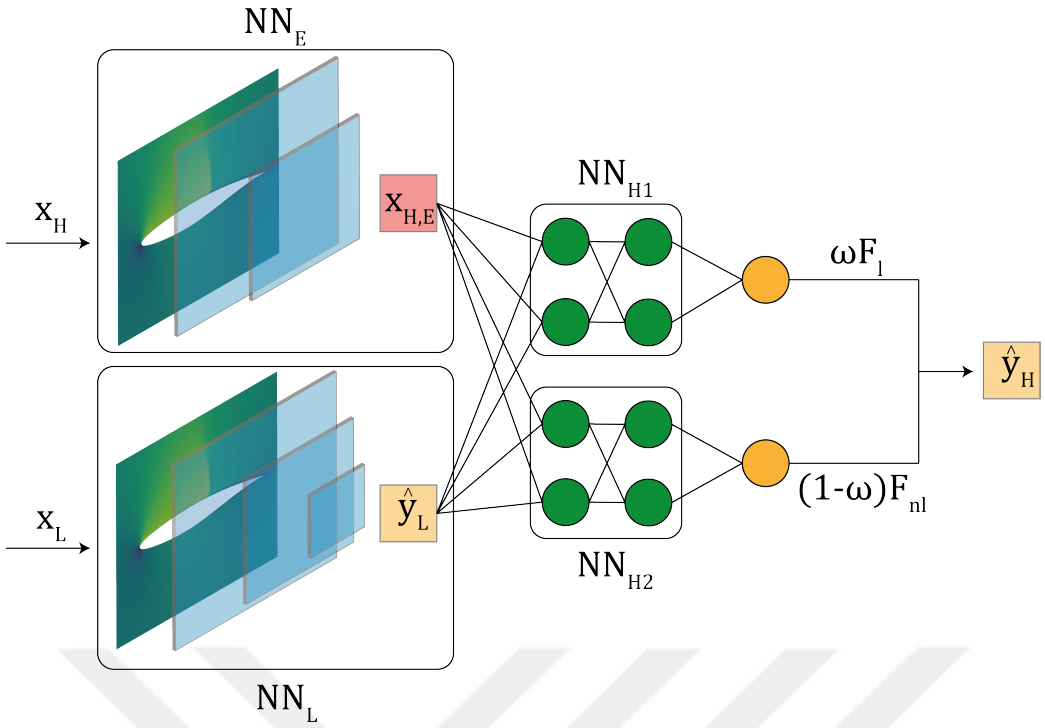

Figure 2.20 : A generic multi-fidelity convolutional neural network architecture.

layers. It aims to extract features of a high-fidelity input and represent them in a different dimension. In this thesis, quantities that are predicted have lower dimensions in comparison of high-fidelity inputs. Considering the data dimensions, an encoder version of NN_E is studied. Briefly, it performs dimensionality reduction for a given high-fidelity input into proper size such that resulting array can be concatenated with the low-fidelity predictions similar to the discussion in the modified-MFDNN.

3. APPLICATION

In the context of the thesis, a multi-fidelity supercritical airfoil problem is considered to evaluate the performances of three different multi-fidelity neural network architectures: multi-fidelity deep neural networks presented in the literature, the modified multi-fidelity deep neural networks, and multi-fidelity convolutional neural networks. The performance of each method is evaluated according to the test accuracy and computational needs.

3.1 Dataset Generation for Flow Around a Supercritical Airfoil

The open-source SU2 multiphysics suite [61] is integrated into this thesis project to generate flow field datasets. RAE2822 airfoil problem with varying flow parameters is studied for multi-fidelity dataset generation. For high-fidelity data, RANS equations are solved employing SA turbulence equation whereas inviscid Euler simulations are run for low-fidelity data generation as in [62, 63].

The generated flow domain around the airfoil has roughly 50 chord length radius. O-grid is used for both type of simulations. y^+ value is set less than 1 over the surface of airfoil for viscous simulations. The discretized flow domains are depicted in Fig. 3.1.

Jameson-Schmidt-Turkel (JST) flux differencing scheme, Flexible Generalized Minimum Residual (FGMRES) linear solver, and weighted least squares gradient computation are employed with adaptive CFL number. The convergence field is set as RMSE of density and convergence criterion of 10^{-8} is used. Firstly, a validation study is performed at Mach 0.729, Reynolds number of 6.5×10^6, and a constant normal force coefficient, C_N, of 0.743. The distinguishing parameters and computation costs experienced in validation study using 36 Intel(R) Xeon(R) Gold 6148 CPU 2.40GHz cores for HF and LF simulations are provided in Table 3.1.

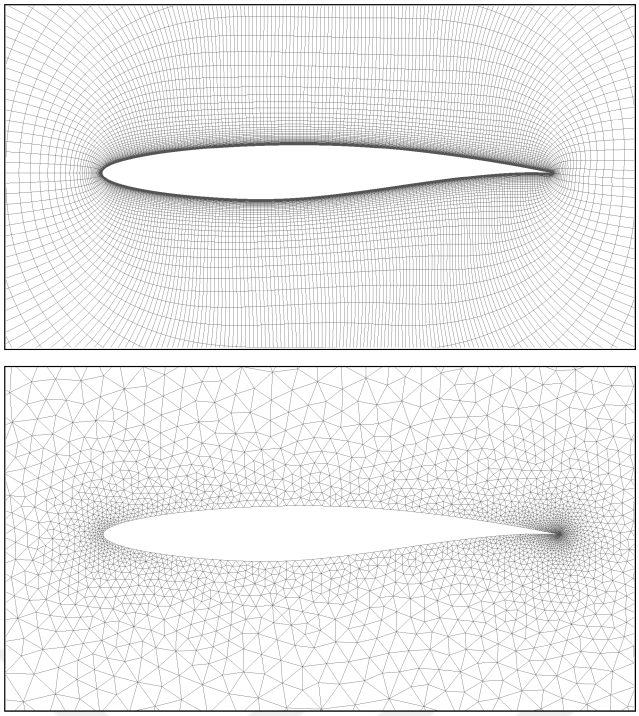

Figure 3.1 : The discretized flow domains: high-fidelity (top), low-fidelity (bottom)
.

Table 3.1 : Numerical differences of high-fidelity and low-fidelity simulations.

Fidelity	**Solver**	**Number of cells**	**Cost** (second/analysis)
High-fidelity	RANS-SA	32k	32.9
Low-fidelity	Euler	10k	1.7

Pressure coefficients over the airfoil obtained with both HF and LF simulations are compared with experimental results [64] shared by NASA. The comparison is depicted in Fig. 3.2. As it is seen, HF pressure coefficient agrees well with the experimental results; however, a remarkable difference occurs in low-fidelity flow field especially at the end of the shock wave on the upper surface of the airfoil. Therefore, the impact of fidelity level is obvious in favor of high-fidelity solutions which comes with a high computational burden. Mach contours of validated solutions are represented in Fig. 3.3.

For dataset generation, Mach number, M, angle of attack, α, and Reynolds number, Re, are considered varying. A broad range is determined for each flow variable to allow complex flow phenomena occur such as flow separation and different shock waves. The lower and upper bounds of the considered variables are given in Table 3.2.

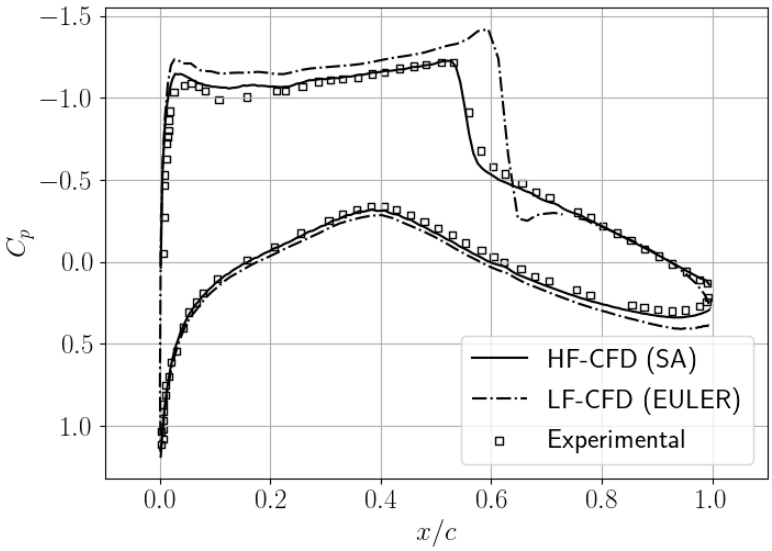

Figure 3.2 : The validation results for RAE2822.

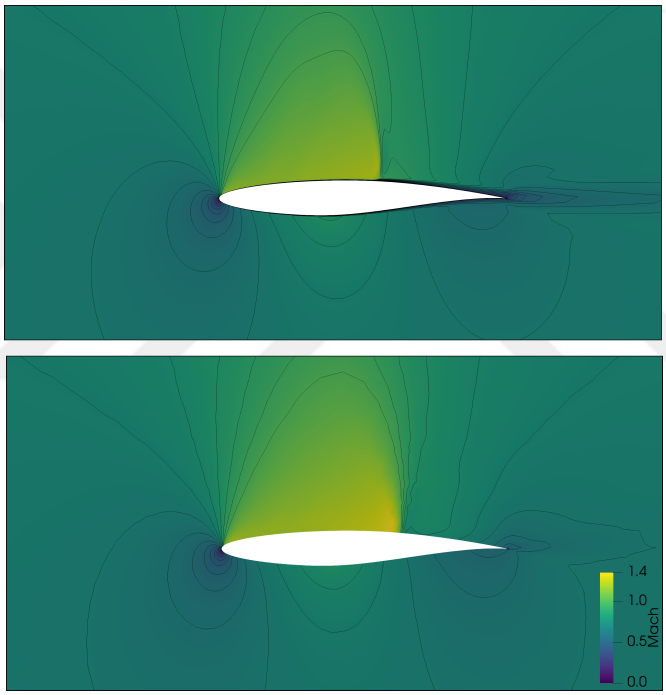

Figure 3.3 : The comparison of Mach field for high-fidelity (top) and low-fidelity (bottom) RAE2822 flow solutions.

Table 3.2 : The lower and upper bounds of flow parameters.

Parameter	**LB**	**UB**
Mach	0.4	0.8
α (°)	-4	8
Re ($\times 10^6$)	5	7.5

Furthermore, Halton quasi-random sampling [65] is employed to obtain high- and low-fidelity samples. Halton sampling uses a deterministic approach to generate sample points in a unit hypercube that seem randomly distributed using coprime numbers and is appealed widely in multi-fidelity analysis. For this application, the dataset contains 200 high-fidelity and 300 low-fidelity simulation results in total are generated. The number of high-fidelity samples are determined heuristically and more than it is needed for the application considered in this thesis. The main purpose of choosing high number of high-fidelity data is to investigate the influence of high-fidelity sample size on the performance of a multi-fidelity neural network by gradually increasing the number of high-fidelity samples.

In order to improve and expedite the heuristic neural network training approach, the inspection of data has a crucial aspect. Especially, the correlation between the low-fidelity and high-fidelity datasets unveils the main concepts of employable neural networks. The correlation charts of drag and lift coefficients are depicted in Fig. 3.4. As it is seen, the viscous effects included in RANS simulations are quite blatant.

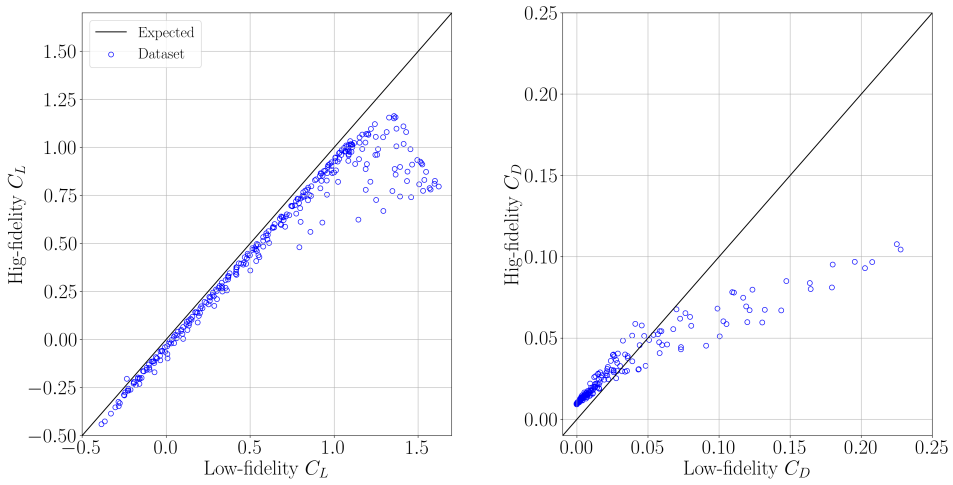

Figure 3.4 : The correlation of drag and lift coefficients of the parameter-varying RAE2822 airfoil dataset.

In addition, high lift conditions create a distinct discrepancy between the inviscid low-fidelity and viscous high-fidelity solutions due to flow separation over the airfoil where low-fidelity solver overestimates the lift coefficients, C_L. A similar effect can also be observed on the drag coefficient, C_D. For transonic regime with high angle of attack values, Euler solutions cannot capture the shock waves accurately under the

given analysis conditions causing high drag coefficient estimations. Furthermore, lift coefficient have a mostly linear relation as it is expected in the lower C_L values. High lift conditions occur at high angle of attack values where flow separation over the airfoil can only be seen in viscous high-fidelity solutions. On the other hand, drag coefficient has a slight nonlinear behaviour. At first sight, shallow linear and nonlinear correlation networks seem to be enough to predict the given aerodynamic coefficient datasets with a low number of high-fidelity data accurately since the correlation is not complex.

Eventually, the dataset is pre-processed to make it suitable for neural network training according to the utilized method. Pressure coefficient fields of the entire simulations are interpolated on a 64-by-64 Cartesian grid with linear interpolation which was found adequate for a 2-dimensional airfoil application as presented under Section 2.2.5.1. In addition, the low- and high-fidelity interpolated flow domains and targets are normalized with the min-max normalization scheme given in equation **(2.20)** to the range of [0,1] to improve the numerical stability of training. The entire dataset is shuffled without disordering the input-target pairs, and then split into training and test sets with the ratio of $85\% - 15\%$. Finally, the constant batch size of 32 is used for partitioning the dataset to shrink the memory requirement during training and test.

3.2 Aerodynamic Prediction From Flow Fields Using Multi-fidelity Neural Networks

In this section, the considered multi-fidelity neural network architectures are evaluated using the generated multi-fidelity supercritical airfoil dataset. For the MFDNN and the modified-MFDNN, three similar architectures with varying learnable parameter set sizes are investigated. Besides, the high-fidelity sample size is gradually increased for each model to observe the effect of the number of high-fidelity data points. The performance metrics for the evaluation of methods are determined as test accuracy in terms of root mean squared error, training time, and the utilized high-fidelity dataset size with a given fixed low-fidelity sample size.

3.2.1 Aerodynamic prediction using multi-fidelity deep neural networks

The generated supercritical airfoil dataset is first used to predict aerodynamic lift and drag coefficients of RAE2822 airfoil under varied flow conditions using the

MFDNN. The low-fidelity pressure coefficient input dataset $\mathbf{x}_L \in \mathbb{R}^{300\times64\times64}$ and the high-fidelity pressure coefficient input dataset $\mathbf{x}_H \in \mathbb{R}^{200\times64\times64}$ are respectively vectorized into $\vec{\mathbf{x}}_L \in \mathbb{R}^{300\times4096}$ and $\vec{\mathbf{x}}_H \in \mathbb{R}^{200\times4096}$ such that they can be processed using affine layers given in equation **(2.2)**. The constructed MFDNN architectures are elaborated in Table 3.3 and 3.4 where n_θ is the total number of learnable parameters that each multi-fidelity neural network possesses. All of the MFDNNs uses the same linear, NN_{H_1}, and nonlinear correlation, NN_{H_2}, networks but varying low-fidelity estimators, NN_L.

Table 3.3 : Detailed MFDNN low-fidelity estimator architectures used to predict aerodynamic coefficients.

n_θ	Sub-network	Layers	Out Features
2.4×10^6	NN_L	Linear-ReLU	512
		Linear-ReLU	128
		Linear-ReLU	16
		Linear-ReLU	2
4.6×10^6	NN_L	Linear-ReLU	1024
		Linear-ReLU	128
		Linear-ReLU	16
		Linear-ReLU	2
10.9×10^6	NN_L	Linear-ReLU	2048
		Linear-ReLU	1024
		Linear-ReLU	128
		Linear-ReLU	2

Table 3.4 : Detailed MFDNN correlator network architectures used to predict aerodynamic coefficients.

Sub-network	Layers	Out Features
NN_{H_1}	Linear	32
	Linear	2
NN_{H_2}	Linear-ReLU	32
	Linear-ReLU	32
	Linear-ReLU	2

As Table 3.3 reveals, MFDNNs with 3 different learnable parameter sizes are used. All the models have the same correlation networks NN_{H_1} and NN_{H_2}. The parameter size changes with the alteration of the low-fidelity estimator NN_L. The initial learnable parameters are uniformly sampled where the bounds are determined as given in

equation **(3.1)**.

$$\text{bound} = \frac{1}{\sqrt{W_2}} \tag{3.1}$$

where $\mathbf{W} \in \mathbb{R}^{W_1 \times W_2}$ is a layer weight tensor. Thus, any model parameter in the MFDNN is initialized with the samples from $\mathbf{W} \sim \mathscr{U}(-\text{bound}, \text{bound})$ and $\mathbf{b} \sim \mathscr{U}(-\text{bound}, \text{bound})$ where $\mathbf{b}$ is the bias array of a layer.

Two convergence criteria are set for training the networks. Training continues if the total number of epochs is less than 3500 or training MSE loss is more than 10^{-4}. These criteria are determined after a couple of training trials. In general, convergence criterion is observed over the test loss; nevertheless, training and test losses follow a similar pattern during the training in this application. Thus, the observation of convergence using the training loss can be seen as the same that is done using the test loss. In addition, a single Nvidia Quadro P4000 GPU is used to speed up trainings. A constant learning rate of 10^{-4} and weight decay of 10^{-5} are used to update the model parameters using Adam stochastic optimization method. The same training setup is used for 5 cases of increasing high-fidelity sample size and fixed low-fidelity sample size of 300 to investigate the effect of the number of high-fidelity data. The RMSE of low-fidelity and multi-fidelity predictions of aerodynamic coefficients are tabulated in Table 3.5. The average physical training time of MFDNNs is approximately 150 seconds. The results show that MFDNN does not provide acceptable multi-fidelity predictions of aerodynamic coefficients due to incapability of learning the correlation. This is because a high-fidelity input $\mathbf{x_H} \in \mathbb{R}^{4096}$ has much more higher data points when compared to a low-fidelity prediction, $\hat{\mathbf{y}}_\mathbf{L} \in \mathbb{R}^2$. Intuitively, the input is the dominant information in the concatenated vector that is given to correlator networks which brings about the neural network to ignore low-fidelity predictions. Therefore, a remedy for this issue is addressed with the modified-MFDNN.

Table 3.5 : The prediction results of the aerodynamic coefficients using MFDNN.

n_θ	NS_H	**LF-RMSE**	**MF-RMSE**
2.4×10^6	10	0.0177	0.0458
	50	0.0177	0.0483
	100	0.0177	0.0477
	150	0.0177	0.0296
	200	0.0177	0.0473
4.6×10^6	10	0.0142	0.0447
	50	0.0142	0.0481
	100	0.0142	0.0475
	150	0.0142	0.0330
	200	0.0142	0.0480
10.9×10^6	10	0.0177	0.0449
	50	0.0177	0.0484
	100	0.0177	0.0484
	150	0.0177	0.0322
	200	0.0177	0.0486

3.2.2 Aerodynamic prediction using the modified multi-fidelity deep neural networks

The modified-MFDNN is an enhanced version of the MFDNN for processing high-dimensional inputs. It brings an additional sub-network to transform a high-fidelity input to a different dimension; hence, it does not dominate low-fidelity predictions in correlation computations. Similar to the MFDNN application, the low-fidelity pressure coefficient input dataset $\mathbf{x}_L \in \mathbb{R}^{300\times64\times64}$ and the high-fidelity pressure coefficient input dataset $\mathbf{x}_H \in \mathbb{R}^{200\times64\times64}$ are respectively vectorized into $\vec{\mathbf{x}}_L \in \mathbb{R}^{300\times4096}$ and $\vec{\mathbf{x}}_H \in \mathbb{R}^{200\times4096}$. We investigated the transformed high-fidelity input sizes of 4, 8, and 16 to observe the effect of extracted feature sizes on the multi-fidelity predictions. As a low-fidelity prediction is in shape of $\hat{\mathbf{y}}_\mathbf{L} \in \mathbb{R}^2$, the concatenated high-fidelity input and low-fidelity prediction vector that is given to the correlator networks has shapes of $\mathbf{x}_\mathbf{C} \in \mathbb{R}^6$, $\mathbf{x}_\mathbf{C} \in \mathbb{R}^{10}$, and $\mathbf{x}_\mathbf{C} \in \mathbb{R}^{18}$. When compared to the MFDNN application where the concatenated vector is $\mathbf{x}_\mathbf{C} \in \mathbb{R}^{4098}$, we could represent the information of high-fidelity input and low-fidelity predictions more uniformly. As it is seen in Table 3.5, models with different parameter sizes give similar outcomes in terms of MF-RMSE. Thus, we used the same NN_L, NN_{H_1}, and NN_{H_2} of the model with $n_\theta = 2.4 \times 10^6$ in the modified-MFDNN to save computation time. The employed NN_E peculiar to the modified-MFDNN architecture is elaborated on Table

3.6. At the end of the day, the modified-MFDNN models have approximately 6.6×10^6 learnable parameters. Thus, NN_E brings an additional computation cost where the modified-MFDNN has roughly 2.75 times higher number of learnable parameters in comparison to the MFDNN with the identical architecture except the NN_E.

Table 3.6 : Employed NN_E architecture used in the modified-MFDNN application.

Sub-network	Layers	Out Features
NN_E	Linear	1024
	Linear	256
	Linear	64
	Linear	4/8/16

The same hyperparameter set is used as in the MFDNN application. The average physical training time is 240 seconds. The 60% increment in the computational cost, when compared to MFDNN, is caused by the additional learnable parameters brought by NN_E. The RMSE results of the prediction of normalized aerodynamic coefficients and the comparison of MF-RMSE obtained with MFDNN are listed in Table 3.7. The given RMSE values are obtained almost the same for different high-fidelity sample sizes. The table reveals that the modified-MFDNN ameliorates MF-RMSE results up to 66% compared to the MFDNN for predictions using flow fields. However, increasing the high-fidelity sample size does not affect the multi-fidelity predictions; therefore, we conclude that latent vectors, lower-dimensional representations of the high-fidelity inputs, cause loss of information during downsampling. Therefore, the learning of the correlation function is impeded.

Table 3.7 : The prediction results of the aerodynamic coefficients using the modified-MFDNN.

$n_{H,E}$	NS_H	LF-RMSE	MF-RMSE	MF-RMSE Improvement (%)
4	[10,200]	0.0155	0.0162	66.0
8	[10,200]	0.0208	0.0215	50.8
16	[10,200]	0.0170	0.0240	45.1

3.2.3 Aerodynamic prediction using multi-fidelity convolutional neural networks

Lastly, the proposed multi-fidelity framework, MFCNN, is employed in the prediction of multi-fidelity aerodynamic coefficients. The low-fidelity pressure coefficient input dataset of $\mathbf{x}_L \in \mathbb{R}^{300\times64\times64}$ and the high-fidelity pressure coefficient input dataset of

$\mathbf{x}_H \in \mathbb{R}^{200\times64\times64}$ are used. The linear and nonlinear correlator networks, NN_{H_1}, NN_{H_2}, are used in the MFCNN model as given in Table 3.4. The low-fidelity estimator, NN_L, is constructed with the combination of 2-dimensional convolution layers and ReLU activation functions where the layer parameters are shared in Table 3.8 with the architecture NN_E. Moreover, NN_E is a convolutional encoder and does not consist of activation functions. Therefore, it is a linear function from the input domain to a latent range $f : \mathbb{R}^{1\times64\times64} \rightarrow \mathbb{R}^{16}$ where the latent vector size of 16 is used. Eventually, the MFCNN architecture composes of 5143 learnable parameters. The number of learnable parameters diminishes by about %99.8 when compared to MFDNN by employing CNN instead of FCNN.

Table 3.8 : Employed NN_L and NN_E architecture used in the MFCNN application.

Sub-network	**Layers**	**Filter/Kernel/Stride**
NN_L	Conv2d-ReLU	2/4/2
	Conv2d-ReLU	4/4/2
	Conv2d-ReLU	8/3/2
	Conv2d-ReLU	16/3/2
	Linear-ReLU-Linear	16^*/-/2^*
NN_E	Conv2d	1/4/4
	Conv2d	1/4/4

* The number of output features

In order to investigate the efficacy of the multi-fidelity convolutional neural network architecture, the same learning rate and weight decay as in the previous applications are employed with the L_1 loss function which is minimized with the Adam optimizer. The average physical training time of MFCNN is 235 seconds for the considered problem. The obtained RMSE values are provided in Table 3.9. In this table, the improvements of MF-RMSE obtained by using MFCNN are also given for MFDNN vs. MFCNN and the modified-MFDNN vs. MFCNN.

Table 3.9 : The prediction results of the aerodynamic coefficients using the MFCNN architecture.

			MF-RMSE Improvement (%)	
NS_H	**LF-RMSE**	**MF-RMSE**	MFDNN vs. MFCNN	Modified-MFDNN vs. MFCNN
10	0.0075	0.0225	50.9	-42.4
50	0.0059	0.0113	76.5	30.0
100	0.0060	0.0105	77.9	35.0
150	0.0065	0.0100	66.2	38.3
200	0.0075	0.0101	78.7	37.7

As a result, MFCNN architecture outperforms both the MFDNN and the modified-MFDNN in data-driven predictive modeling using flow fields as inputs. Additionally, unlike MFDNN and the modified-MFDNN, increasing the high-fidelity sample size enhances the multi-fidelity predictions. Hence, it can be asserted that MFCNN learns the correlation better than the previously presented methods.

4. RESULTS and DISCUSSION

Firstly, an overall comparison of the presented methods for predictive modeling using flow fields is provided in this section. The best neural network model of each method in terms of MF-RMSE is considered for comparison and visualization of the results. In addition, a comprehensive discussion on the proposed methods and future work are presented.

4.1 Results and Discussion

To begin with, the learning curves of each method with respect to the high-fidelity sample sizes are illustrated in Fig. 4.1. As the figure shows, the fastest learning method

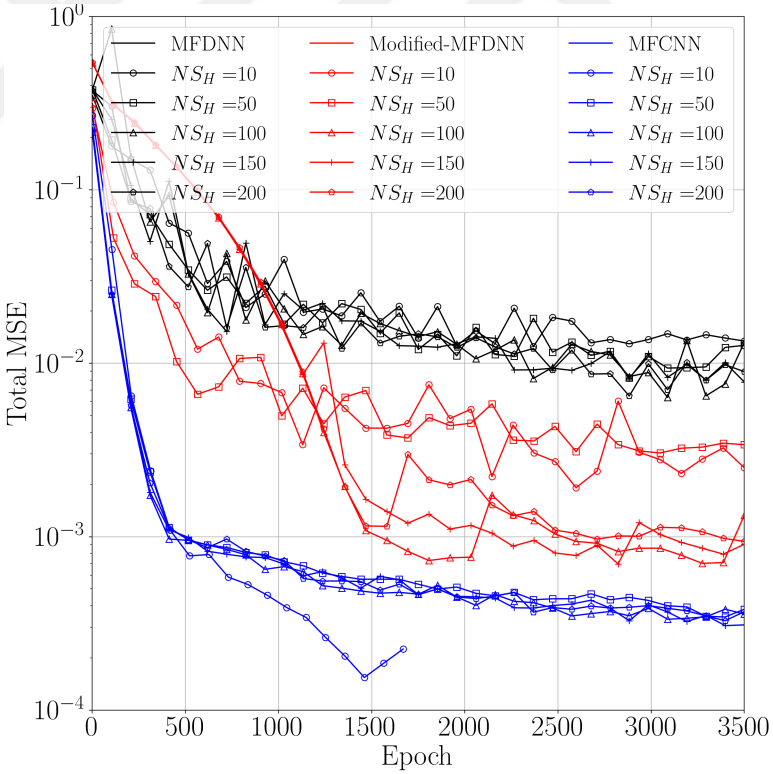

Figure 4.1 : Learning curves of multi-fidelity neural networks with respect to the high-fidelity sample size.

is the MFCNN with a steeper learning pattern based on the first 500 epochs. In 3500 epochs, the minimum training loss is achieved with the MFCNN models. Furthermore,

increasing the number of high-fidelity samples within the training dataset improves the training losses of the modified-MFDNN. However, it has not that much influence on the MFDNN and MFCNN.

The test performances of each method are compared in Fig. 4.2 with low-fidelity and multi-fidelity RMSE values. The impact of training performances is seen on the RMSE values where the MFCNN yields more accurate predictions in both low-fidelity and high-fidelity test datasets. On the multi-fidelity prediction side with the given results, a higher number of high-fidelity sample size only improves the multi-fidelity predictions obtained with the MFCNN which can be interpreted as a higher correlation learning capability. Thus, other methods are not able to distinguish the differences of add on high-fidelity samples. Even though the training performance of the modified-MFDNN is better with the increasing number of high-fidelity samples, the same behavior is not seen in the test accuracy. This is a sign of overfitting and low generalization capability which may be improved with a higher value of weight decay or early stopping.

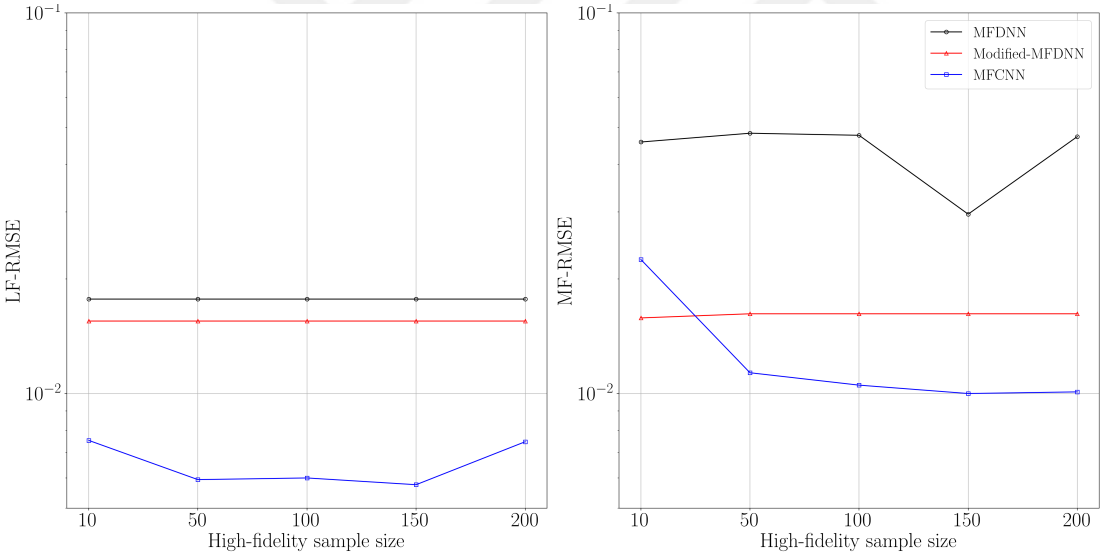

Figure 4.2 : Low-fidelity and multi-fidelity RMSE values of each method obtained using test dataset.

On top of that, the predictions of drag and lift coefficients for all 3 methods are demonstrated in Fig. 4.3 to provide a better visual understanding. It is conspicuous that the MFDNN, prevailing in the literature, is not capable of dealing with the combination of high dimensional inputs and low dimensional predictions. On the other hand, processing high-dimensional inputs to represent them with lower-dimensional latent vectors significantly enhances the multi-fidelity prediction performances. Besides,

since flow fields are composed of highly correlated spatial variables, the MFCNN outperforms the modified-MFDNN, a combination fully-connected neural networks, due to the prevention of vectorized flow fields so does the loss of information.

Finally, the computational savings using the modified-MFDNN and the MFCNN for different high-fidelity sample sizes are given in Table 4.1. The percentage savings are computed with the equations **(4.1)** and **(4.2)**.

$$\begin{aligned} T_{mf} &= t_l \times NS_L + t_h \times NS_H \\ T_{hf} &= t_h \times (NS_L + NS_H) \end{aligned} \tag{4.1}$$

Here, T is the total computation time needed, and subscripts mf and hf respectively correspond to multi-fidelity and high-fidelity. The computational time of each low-fidelity and high-fidelity simulations is given with t_l and t_h, respectively. For the presented supercritical airfoil application computational simulation times are $t_l = 1.7$ and $t_h = 32.9$ seconds as given in Table 3.1. To remind, NS_L is the number of low-fidelity analyses and NS_H is the number of high-fidelity analyses.

$$\text{Saving}\% = \frac{T_{hf} - T_{mf}}{T_{hf}} \times 100 \tag{4.2}$$

Table 4.1 : The computational savings using the modified-MFDNN and MFCNN.

NS_H	**Savings** (%)
10	91.8
50	81.3
100	71.1
150	63.2
200	56.7

As concluding remarks, two novel multi-fidelity neural network architectures in predictive modeling using high-dimensional inputs are proposed. In total, 3 different neural network models are compared: the multi-fidelity deep neural network, the modified multi-fidelity deep neural network, and the multi-fidelity convolutional neural network. In order to get a better understanding, interpolation of flow fields onto a Cartesian grid and the optimization of the correlation weight defined in the autoregressive scheme are scrutinized where the results are presented. The findings of these preliminary investigations are leveraged within the following parts of the study. Furthermore, a multi-fidelity 2-dimensional aerodynamic problem is taken into

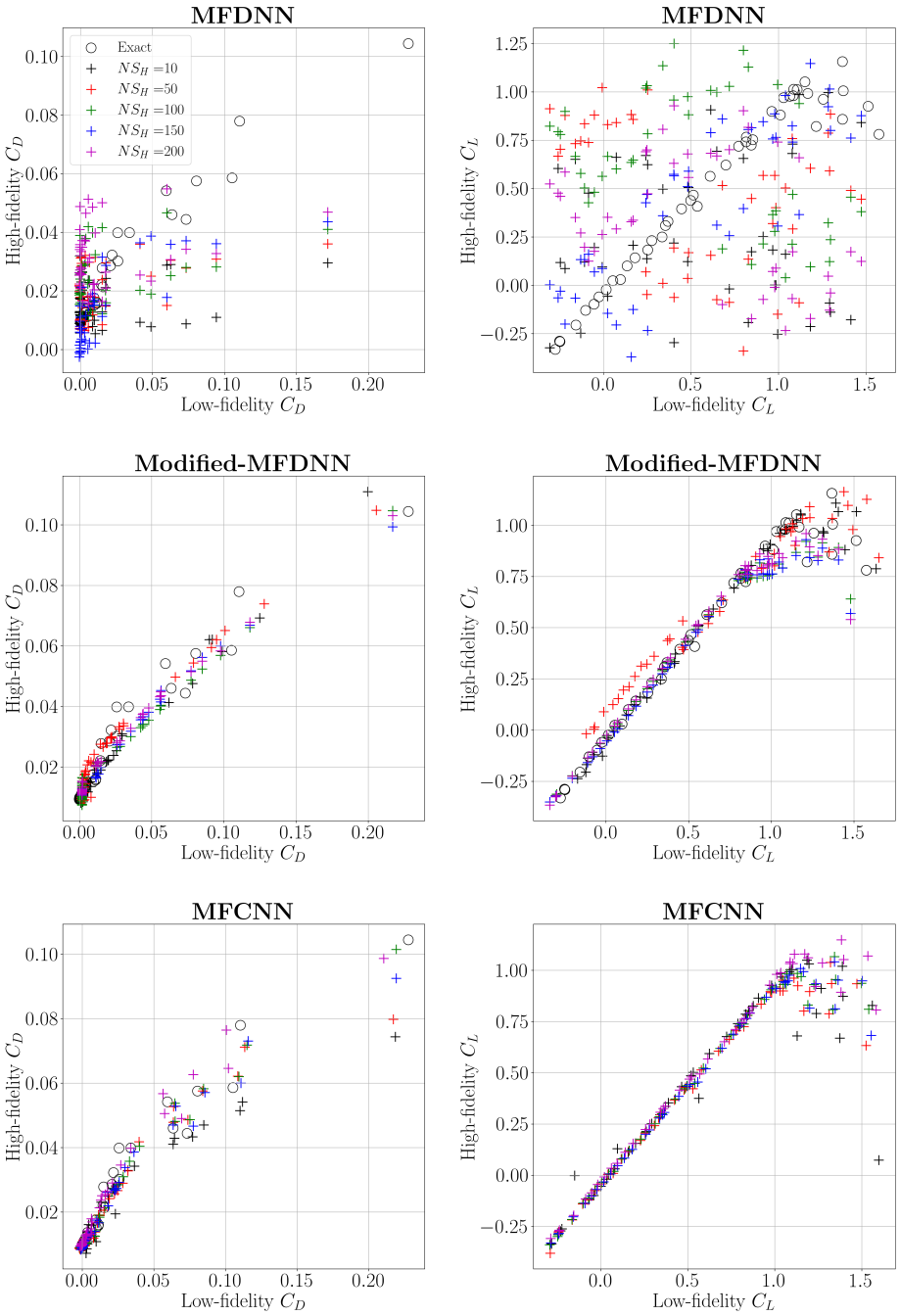

Figure 4.3 : Multi-fidelity aerodynamic predictions of the MFDNN, the modified-MFDNN, and the MFCNN using pressure coefficient fields.

account. RAE2822 supercritical airfoil is analyzed using a low-fidelity Euler solver with a coarse grid and a high-fidelity RANS solver using the SA turbulence model with a relatively finer grid. Different flow conditions with varying Mach number, angle of attack, and Reynolds number are used to obtain flow fields around the airfoil. The main objective is determined as the prediction of drag and lift coefficients under varying flow conditions using pressure coefficient fields around the airfoil. Halton sampling is utilized to generate varying flow parameters; in addition, 200 high-fidelity and 300 low-fidelity simulations are performed using the SU2 flow solver. The obtained flow fields are pre-processed to transform them into a suitable form for neural network training and testing. Furthermore, 3 different MFDNN models with different learnable parameter sizes are investigated. Likewise, for the modified-MFDNN, 3 different latent vector sizes are analyzed. In addition, training of all cases is run on the Python environment using PyTorch deep learning and optimization libraries. For the prediction of lift and drag coefficients, L_1 known as the mean absolute error loss function is employed with the renowned Adam stochastic gradient optimizer. The same hyperparameter set is used for all cases. Test accuracy, physical training time, and the computational savings are evaluated to quantify the performances of each method. The results obtained with the 2-dimensional airfoil problem reveal that the multi-fidelity deep neural network is not capable predicting aerodynamic coefficients using pressure coefficient fields where the modified-MFDNN substantially ameliorates the multi-fidelity prediction; besides, the multi-fidelity convolutional neural network outclasses the modified multi-fidelity deep neural network in terms of prediction accuracy on test dataset with a given high-fidelity sample size except for a case.

The results presented in this thesis are preliminary outcomes of ongoing research. A set of hyperparameters, loss function, and optimizer are considered only for the entire investigated cases. Since heuristic approaches are essential for deep learning methods, there may be room for improvement in all of the results. In future works, the two proposed multi-fidelity neural network architectures, the modified-MFDNN, and MFCNN should be scrutinized using not only flow fields but any high dimensional input with low dimensional predictions as well. On top of that, the correlation between the low- and high-fidelity cases in the presented application is mostly linear and

slightly nonlinear. The prediction performances of the considered architectures should be analyzed with more complex cases where the correlation is highly nonlinear.

REFERENCES

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

[2] **Min, S.**, **Lee, B. and Yoon, S.**, (2016), Deep Learning in Bioinformatics, arXiv:1603.06430.

[3] **Spencer, M.**, **Eickholt, J. and Cheng, J.** (2015). A Deep Learning Network Approach to italicab initio/italic Protein Secondary Structure Prediction, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *12*(1), 103–112.

[4] **Cao, R.**, **Bhattacharya, D.**, **Hou, J. and Cheng, J.** (2016). DeepQA: improving the estimation of single protein model quality with deep belief networks, *BMC Bioinformatics*, *17*(1).

[5] **Toms, B.A.**, **Barnes, E.A. and Ebert-Uphoff, I.** (2020). Physically Interpretable Neural Networks for the Geosciences: Applications to Earth System Variability, *Journal of Advances in Modeling Earth Systems*, *12*(9).

[6] **Mamalakis, A.**, **Ebert-Uphoff, I. and Barnes, E.A.**, (2021), Neural Network Attribution Methods for Problems in Geoscience: A Novel Synthetic Benchmark Dataset, arXiv:2103.10005.

[7] **Lary, D.J.**, **Alavi, A.H.**, **Gandomi, A.H. and Walker, A.L.** (2016). Machine learning in geosciences and remote sensing, *Geoscience Frontiers*, *7*(1), 3–10.

[8] **Julian, K.D. and Kochenderfer, M.J.** (2021). Reachability Analysis for Neural Network Aircraft Collision Avoidance Systems, *Journal of Guidance, Control, and Dynamics*, *44*(6), 1132–1142, https://doi.org/10.2514/1.G005233.

[9] **Hovell, K. and Ulrich, S.** (2021). Deep Reinforcement Learning for Spacecraft Proximity Operations Guidance, *Journal of Spacecraft and Rockets*, *58*(2), 254–264.

[10] **Brittain, M.W.**, **Yang, X. and Wei, P.** (2021). Autonomous Separation Assurance with Deep Multi-Agent Reinforcement Learning, *Journal of Aerospace Information Systems*, *18*(12), 890–905, https://doi.org/10.2514/1.I010973.

[11] **van Rooijen, S.J.**, **Ellerbroek, J.**, **Borst, C. and van Kampen, E.** (2020). Toward Individual-Sensitive Automation for Air Traffic Control Using Convolutional Neural Networks, *Journal of Air Transportation*, *28*(3), 105–113, https://doi.org/10.2514/1.D0180.

[12] **Sekar, V.**, **Zhang, M.**, **Shu, C. and Khoo, B.C.** (2019). Inverse Design of Airfoil Using a Deep Convolutional Neural Network, *AIAA Journal*, *57*(3), 993–1003.

[13] **Yu, J. and Hesthaven, J.S.** (2019). Flowfield Reconstruction Method Using Artificial Neural Network, *AIAA Journal*, *57*(2), 482–498.

[14] **Peng, W.**, **Zhang, Y. and Desmarais, M.** (2021). Spatial Convolution Neural Network for Efficient Prediction of Aerodynamic Coefficients, *AIAA Scitech 2021 Forum*, American Institute of Aeronautics and Astronautics.

[15] **Wang, Q.**, **Medeiros, R.R.**, **Cesnik, C.E.**, **Fidkowski, K.**, **Brezillon, J. and Bleecke, H.M.** (2019). Techniques for Improving Neural Network-based Aerodynamics Reduced-order Models, *AIAA Scitech 2019 Forum*, American Institute of Aeronautics and Astronautics.

[16] **Du, X.**, **He, P. and Martins, J.R.** (2021). Rapid airfoil design optimization via neural networks-based parameterization and surrogate modeling, *Aerospace Science and Technology*, *113*, 106701.

[17] **Achour, G.**, **Sung, W.J.**, **Pinon-Fischer, O.J. and Mavris, D.N.** (2020). Development of a Conditional Generative Adversarial Network for Airfoil Shape Optimization, *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics.

[18] **Tekaslan, H.E.**, **Imrak, R. and Nikbay, M.** (2021). Reliability Based Design Optimization of a Supersonic Engine Inlet, *AIAA Propulsion and Energy 2021 Forum*, American Institute of Aeronautics and Astronautics.

[19] **Tekaslan, H.E.**, **Demiroglu, Y. and Nikbay, M.** (2022). Surrogate Unsteady Aerodynamic Modeling with Autoencoders and LSTM Networks, *AIAA SCITECH 2022 Forum*, American Institute of Aeronautics and Astronautics.

[20] **Liu, Y.**, **Chen, S.**, **Wang, F. and Xiong, F.** (2018). Sequential optimization using multi-level cokriging and extended expected improvement criterion, *Structural and Multidisciplinary Optimization*, *58*(3), 1155–1173.

[21] **Keane, A.J.** (2012). Cokriging for Robust Design Optimization, *AIAA Journal*, *50*(11), 2351–2364.

[22] **Nagawkar, J.**, **Leifsson, L.T. and Du, X.** (2020). Applications of Polynomial Chaos-Based Cokriging to Aerodynamic Design Optimization Benchmark Problems, *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics.

[23] **March, A.**, **Willcox, K. and Wang, Q.** (2011). Gradient-based multifidelity optimisation for aircraft design using Bayesian model calibration, *The Aeronautical Journal (1968)*, *115*(1174), 729–738.

[24] **Cheng, K.**, **Lu, Z. and Zhen, Y.** (2019). Multi-level multi-fidelity sparse polynomial chaos expansion based on Gaussian process regression, *Computer Methods in Applied Mechanics and Engineering*, *349*, 360–377.

[25] **West, T.K. and Phillips, B.D.** (2020). Multifidelity Uncertainty Quantification of a Commercial Supersonic Transport, *Journal of Aircraft*, *57*(3), 491–500.

[26] **Meng, X. and Karniadakis, G.E.** (2020). A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems, *Journal of Computational Physics*, *401*, 109020.

[27] **Nagawkar, J.R.**, **Leifsson, L.T. and He, P.** (2022). Aerodynamic Shape Optimization Using Gradient-Enhanced Multifidelity Neural Networks, *AIAA SCITECH 2022 Forum*, American Institute of Aeronautics and Astronautics.

[28] **He, L.**, **Qian, W.**, **Zhao, T. and Wang, Q.** (2020). Multi-Fidelity Aerodynamic Data Fusion with a Deep Neural Network Modeling Method, *Entropy*, *22*(9), 1022.

[29] **Zhang, X.**, **Xie, F.**, **Ji, T.**, **Zhu, Z. and Zheng, Y.** (2021). Multi-fidelity deep neural network surrogate model for aerodynamic shape optimization, *Computer Methods in Applied Mechanics and Engineering*, *373*, 113485.

[30] **Meng, X.**, **Babaee, H. and Karniadakis, G.E.** (2021). Multi-fidelity Bayesian neural networks: Algorithms and applications, *Journal of Computational Physics*, *438*, 110361.

[31] **Gotmare, A.D.**, **Keskar, N.S.**, **Xiong, C. and Socher, R.** (2019). A Closer Look at Deep Learning Heuristics: Learning rate restarts, Warmup and Distillation, *ArXiv*, *abs/1810.13243*.

[32] **Hornik, K.**, **Stinchcombe, M. and White, H.** (1989). Multilayer feedforward networks are universal approximators, *Neural Networks*, *2*(5), 359–366.

[33] **Nair, V. and Hinton, G.E.** (2010). Rectified Linear Units Improve Restricted Boltzmann Machines, *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, Omnipress, Madison, WI, USA, p.807–814.

[34] **He, K.**, **Zhang, X.**, **Ren, S. and Sun, J.** (2016). Deep Residual Learning for Image Recognition, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.770–778.

[35] **Bukka, S.R.**, **Magee, A.R. and Jaiman, R.K.** (2020). Deep Convolutional Recurrent Autoencoders for Flow Field Prediction, *Volume 8: CFD and FSI*, American Society of Mechanical Engineers.

[36] **Yu, J. and Hesthaven, J.S.** (2019). Flowfield Reconstruction Method Using Artificial Neural Network, *AIAA Journal*, *57*(2), 482–498.

[37] **Li, Y.**, **Chang, J.**, **Kong, C. and Wang, Z.** (2020). Flow field reconstruction and prediction of the supersonic cascade channel based on a symmetry neural network under complex and variable conditions, *AIP Advances*, *10*(6), 065116–1–065116–19.

[38] **Ioffe, S. and Szegedy, C.** (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, JMLR.org, p.448–456.

[39] **Srivastava, N.**, **Hinton, G.**, **Krizhevsky, A.**, **Sutskever, I. and Salakhutdinov, R.** (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *Journal of Machine Learning Research*, *15*(56), 1929–1958.

[40] **Aly, A.**, **Guadagni, G. and Dugan, J.B.** (2019). Derivative-Free Optimization of Neural Networks using Local Search, *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, pp.0293–0299.

[41] **Chen, Y.**, **Chang, H.**, **Meng, J. and Zhang, D.** (2019). Ensemble Neural Networks (ENN): A gradient-free stochastic method, *Neural Networks*, *110*, 170–185.

[42] **Rumelhart, D.E.**, **Hinton, G.E. and Williams, R.J.** (1986). Learning representations by back-propagating errors, *Nature*, *323*(6088), 533–536, `https://doi.org/10.1038/323533a0`.

[43] **Raissi, M.**, **Perdikaris, P. and Karniadakis, G.** (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *Journal of Computational Physics*, *378*, 686–707.

[44] **Kingma, D.P. and Ba, J.**, (2014), Adam: A Method for Stochastic Optimization, `arXiv:1412.6980`.

[45] **Shanker, M.**, **Hu, M. and Hung, M.** (1996). Effect of data standardization on neural network training, *Omega*, *24*(4), 385–397.

[46] **Sola, J. and Sevilla, J.** (1997). Importance of input data normalization for the application of neural networks to complex industrial problems, *IEEE Transactions on Nuclear Science*, *44*(3), 1464–1468.

[47] **Bhanja, S. and Das, A.**, (2018), Impact of Data Normalization on Deep Neural Network for Time Series Forecasting.

[48] **Gürbüzbalaban, M.**, **Ozdaglar, A. and Parrilo, P.A.** (2019). Why random reshuffling beats stochastic gradient descent, *Mathematical Programming*, *186*(1-2), 49–84.

[49] **Recht, B. and Re, C.** (2012). Toward a Noncommutative Arithmetic-geometric Mean Inequality: Conjectures, Case-studies, and Consequences, ***S. Mannor, N. Srebro and R.C. Williamson**, editors, Proceedings of the 25th Annual Conference on Learning Theory*, volume 23 of *Proceedings of Machine Learning Research*, PMLR, Edinburgh, Scotland, pp.11.1–11.24.

[50] **Meng, Q.**, **Chen, W.**, **Wang, Y.**, **Ma, Z.M. and Liu, T.Y.** (2019). Convergence analysis of distributed stochastic gradient descent with shuffling, *Neurocomputing*, *337*, 46–57.

[51] **Joseph, V.R.**, (2022), Optimal Ratio for Data Splitting.

[52] **Bukka, S.R.**, **Magee, A.R. and Jaiman, R.K.**, (2020), Deep Convolutional Recurrent Autoencoders for Flow Field Prediction, `arXiv:2003.12147`.

[53] **Ahrens, J.P.**, **Geveci, B. and Law, C.C.** (2005). ParaView: An End-User Tool for Large-Data Visualization, *The Visualization Handbook*.

[54] **Peherstorfer, B.**, **Willcox, K. and Gunzburger, M.** (2018). Survey of Multifidelity Methods in Uncertainty Propagation, Inference, and Optimization, *SIAM Review*, *60*(3), 550–591.

[55] **Fernández-Godino, M.G.**, **Park, C.**, **Kim, N.H. and Haftka, R.T.** (2019). Issues in Deciding Whether to Use Multifidelity Surrogates, *AIAA Journal*, *57*(5), 2039–2054.

[56] **Chakraborty, S.** (2021). Transfer learning based multi-fidelity physics informed deep neural network, *Journal of Computational Physics*, *426*, 109942.

[57] **Paszke, A.**, **Gross, S.**, **Massa, F.**, **Lerer, A.**, **Bradbury, J.**, **Chanan, G.**, **Killeen, T.**, **Lin, Z.**, **Gimelshein, N.**, **Antiga, L.**, **Desmaison, A.**, **Kopf, A.**, **Yang, E.**, **DeVito, Z.**, **Raison, M.**, **Tejani, A.**, **Chilamkurthy, S.**, **Steiner, B.**, **Fang, L.**, **Bai, J. and Chintala, S.**, (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library, **H. Wallach**, **H. Larochelle**, **A. Beygelzimer**, **F. d'Alché-Buc**, **E. Fox and R. Garnett**, editors, Advances in Neural Information Processing Systems 32, Curran Associates, Inc., pp.8024–8035.

[58] **Forrester, A.I.**, **Sóbester, A. and Keane, A.J.** (2007). Multi-fidelity optimization via surrogate modelling, *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *463*(2088), 3251–3269.

[59] **Dong, H.**, **Song, B.**, **Wang, P. and Huang, S.** (2014). Multi-fidelity information fusion based on prediction of kriging, *Structural and Multidisciplinary Optimization*, *51*(6), 1267–1280.

[60] **Palar, P.S.**, **Tsuchiya, T. and Parks, G.T.** (2016). Multi-fidelity non-intrusive polynomial chaos based on regression, *Computer Methods in Applied Mechanics and Engineering*, *305*, 579–606.

[61] **Economon, T.D.**, **Palacios, F.**, **Copeland, S.R.**, **Lukaczyk, T.W. and Alonso, J.J.** (2016). SU2: An open-source suite for multiphysics simulation and design, *AIAA Journal*, *54*(3), 828–846.

[62] **Ren, J.**, **Leifsson, L.T.**, **Koziel, S. and Tesfahunegn, Y.** (2016). Multi-Fidelity Aerodynamic Shape Optimization Using Manifold Mapping, *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, American Institute of Aeronautics and Astronautics.

[63] **Padron, A.S.**, **Alonso, J.J. and Eldred, M.S.** (2016). Multi-fidelity Methods in Aerodynamic Robust Optimization, *18th AIAA Non-Deterministic Approaches Conference*, American Institute of Aeronautics and Astronautics.

[64] **Cook, P.**, **Firmin, M.**, **McDonald, M. and Establishment, R.A.** (1977). *Aerofoil RAE 2822: Pressure Distributions, and Boundary Layer and Wake Measurements*, Technical memorandum / Royal Aircraft Establishment, RAE.

[65] **Halton, J.H.** (1960). On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals, *Numerische Mathematik*, *2*(1), 84–90.

CURRICULUM VITAE

Name SURNAME: Hüseyin Emre Tekaslan

EDUCATION:

- **B.Sc.:** 2019, Istanbul Technical University, Faculty of Aeronautics and Astronautics, Department of Aeronautical Engineering
- **M.Sc.:** 2022, Istanbul Technical University, Faculty of Aeronautics and Astronautics, Department of Aeronautical and Astronautical Engineering

RESEARCH EXPERIENCE:

- 2018-2019, Undergraduate Researcher, Aerospace Research Center, Istanbul Technical University.
- 2019-2022, Graduate Researcher, AeroMDO Multidisciplinary Design Optimization Laboratory, Istanbul Technical University.

PUBLICATIONS:

- **Tekaslan H.E.**, Nikbay M. "A Multi-fidelity Prediction Framework with Convolutional Neural Networks Using High-Dimensional Data", *AIAA Journal of Aerospace Information Systems* (**Under Review**).
- **Tekaslan H.E.**, Yildiz S., Demiroglu Y., Nikbay M. "Implementation of Multidisciplinary Multi-Fidelity Uncertainty Quantification Methods in Sonic Boom Prediction", *AIAA Journal of Aircraft* (**Accepted**).
- Nikbay M., Kilic D., Cakmak E., **Tekaslan H.E.**, Yildiz S., Demiroglu Y. "Multi-Fidelity and Multi-Disciplinary Design Optimization of A Low-Boom Supersonic Transport Aircraft ", *AIAA SciTech 2023*. (**Abstract submitted**).
- **Tekaslan H.E.**, Nikbay M. "A Multi-fidelity Prediction with Convolutional Neural Networks Using High-Dimensional Data", *AIAA Aviation Forum 2022*. Virtual Event. June 27 - July 1, 2022.

- **Tekaslan H.E.**, Demiroglu Y., Nikbay M. "Surrogate Unsteady Aerodynamic Modeling with Autoencoders and LSTM Networks" *AIAA SciTech 2022*. January 3-7, 2022.

- **Tekaslan H.E.**, Imrak R., Nikbay M. "Reliability Based Design Optimization of a Supersonic Engine Inlet" *AIAA Propulsion and Energy Forum 2021*. Virtual Event. August 9-11, 2021.

- **Tekaslan H.E.**, Yildiz S., Demiroglu Y., Nikbay M. "Implementation of Multidisciplinary Multi-Fidelity Uncertainty Quantification Methods in Sonic Boom Prediction" *AIAA Aviation Forum 2021*. Virtual Event. August 2-6, 2021.

PROJECTS:

- Development of Convolutional Neural Network Based Predictive Modeling Code
  Funded by General Electric Gas Power, UK

- Enhanced Computational Performance and Stability & Control Prediction for NATO Military Vehicle
  NATO STO AVT-351 Research Task Group

- Development of Multi-fidelity and Multidisciplinary Methodologies Integrating Sonic Boom, Aeroelasticity and Propulsion System for Supersonic Aircraft Design
  Funded by TUBITAK, 218M471

AWARDS:

- 2018-2019, Boeing Undergraduate Program Scholarship