Deep Residual Neural Network for Predicting Aerodynamic Coefficient Changes with Ablation

Dong Ho Lee[1] , DongUk Lee[2],

Korea Advanced Institute of Science and Technology, Daejeon, 34141, Republic of Korea

Seoum Han[3], Seongil Seo[4], Bok Jik Lee[5] and

Seoul National University, Seoul, 08826, Republic of Korea

Jaemyung Ahn[6]

Korea Advanced Institute of Science and Technology, Daejeon, 34141, Republic of Korea

Data-driven methods for predicting aerodynamic coefficients of arbitrary shapes have received considerable attention due to their flexibility and scalability. This paper introduces a deep-learning framework based on a deep residual neural network and K-fold cross-validation for fast and accurate prediction of aerodynamic coefficients of a three-dimensional cone with shape changes due to ablation. The proposed neural network model is trained to learn the underlying relationship between shape transformations and the corresponding changes in aerodynamic coefficients. The shape transformations due to ablation are expressed as the difference between the nominal and ablated cones, measured in units of mesh coordinates. Multiple (*K*) models constructed based on the training process are combined to reduce the prediction variance effectively. The resulting ensemble model shows an improved prediction performance for various aerodynamic coefficients. To validate our methodology, we compare our model with the generic Multilayer Perceptron (MLP) with a varying number of neurons and the Gaussian process (GP) regression. The test results indicate that the proposed model predicts the aerodynamic coefficients more accurately than the baseline model (MLP/GP), indicating an improved generalization to unseen data.

[1] Research Assistant, Department of Aerospace Engineering, leedh0124@kaist.ac.kr, AIAA Student Member
[2] Graduate Research Assistant, Department of Aerospace Engineering, dlehddnr52@kaist.ac.kr
[3] Graduate Research Assistant, Department of Aerospace Engineering, seoeum96@snu.ac.kr
[4] Graduate Research Assistant, Department of Aerospace Engineering, manijeon@snu.ac.kr
[5] Associate Professor, Department of Aerospace Engineering, b.lee@snu.ac.kr, AIAA Member
[6] Associate Professor, Department of Aerospace Engineering, jaemyung.ahn@kaist.ac.kr, AIAA Senior Member

Nomenclature

α	=	angle of attack (*deg*)
C_A	=	axial force coefficient (-)
C_N	=	normal force coefficient (-)
C_M	=	pitching moment coefficient (-)
M, M_∞	=	Mach number (-)
R_N	=	nose radius (m)
N_C	=	number of nominal cones
N_A	=	number of ablated cones per nominal cone
$\boldsymbol{x}$	=	input feature vector
$\boldsymbol{p}_l$	=	hidden representation at the l^{th} layer (before activation)
$\boldsymbol{h}_l$	=	hidden representation at the l^{th} layer (after activation)
$\boldsymbol{\theta}_l$ or $(\boldsymbol{W}_l, \boldsymbol{b}_l)$	=	weight and bias parameters of l^{th} layer
$\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{\theta})$	=	learning function of the neural network model
$\hat{y}$	=	predicted output
y	=	true label
η	=	initial learning rate
n_{epoch}	=	number of epochs
ϕ	=	(nonlinear) activation function
B	=	batch size (*rad*)

I. Introduction

The last decade has seen incredible progress in machine learning due to a dramatic increase in the volume of available data and a computational advancement in the graphic processing unit (GPU) for parallel computing. In particular, deep learning (DL), a data-driven methodology within broader machine learning, has received considerable attention from researchers across many engineering domains. The DL has been successful in various applications, particularly those involving high-dimension/unstructured data representations and complex/nonlinear relationships for input-output pairs. Traditionally, approaches in aerodynamics (e.g., aerodynamic shape optimization and flow field analysis) have centered on numerical simulations based on complex computations. Computational fluid dynamics (CFD) relies on heavy calculations because the Navier-Stokes equations embody a system of nonlinear partial differential equations requiring an iterative approach. Given sufficient computational resources and time, modern numerical simulations generate reliable solutions for the areas where analytic solutions are unavailable. However, the capability to create expedited results with acceptable fidelity, which can provide an efficient channel for aerodynamic analysis, is still limited.

The interest in the neural network (NN) models that can learn underlying complex aerodynamic properties is growing. Several studies in the past few years recognized the potential of deep NN models to efficiently predict aerodynamic coefficients, flow fields, and unsteady forces. However, many studies mainly focused on 2-D airfoil images and relatively small ranges of Mach numbers, Reynolds numbers, and angles of attack. Zhang et al. [1] developed a convolutional neural network (CNN) model to learn the input and output relationship between a 2-D airfoil image and the corresponding lift coefficient. Yu et al. [2] proposed an improved CNN model and data augmentation technique using feature-enhanced images to predict the airfoil lift coefficients using a user-defined Open-source Field Operation and Manipulation (OpenFOAM) simulation. Zelong et al. [3] proposed a general approximation model based on the CNN and a signed distance function (SDF) to predict the lift coefficient, drag coefficient, and pitching moment coefficient of airfoils. They used the XFOIL solver to generate the aerodynamic coefficient data and compared the performance with the conventional Kriging model with a relatively small dataset (a few thousand for training and less than a hundred for testing). Lee et al. [4] sought to predict multiple aerodynamic coefficients for various 2-D missile cone shapes under transonic and supersonic flow using a CNN model. Jacob et al. [5] developed a U-Net architecture for multi-task learning to predict the drag coefficient and 3-D velocity fields of an arbitrary mesh. They expressed the input variables over a 3-D Cartesian grid to reduce feature dimension while capturing the intrinsic geometric properties. Recently, Sabater et al. [6] extended the prior works to study the prediction of surface pressure distributions for 2-D airfoil images and 3-D aircraft meshes. They trained a multilayer perceptron (MLP) and considered both subsonic and transonic flow conditions, especially noting that deep learning techniques can capture shock wave location and strength for transonic flow.

While several studies have shown the applicability of deep learning models for various aerodynamic analysis problems, much of their focus was on aerodynamic properties for 2-D airfoils. In addition, cases involving 3-D input representations used relatively small datasets with narrow flow condition ranges. Some studies lacked the validation procedures to ensure that the trained model is not over- or under-fitted to the training dataset. This gap analysis calls for a rigorous training methodology for a reliable implementation of deep learning techniques and improved accuracy for industrial applications.

Aerodynamic shapes under extreme flow conditions (e.g., hypersonic flow) are prone to surface removal caused by high temperatures, changing the aerodynamic coefficients. By training a deep NN model to understand this relationship, we can circumvent the numerical simulations (incurring high cost) and obtain an efficient means of predicting the degree of changes, which applies to the design of heat shields for spacecraft or online uncertainty handling by onboard controllers. He et al. [7] proposed deep NN models based on residual connection for image recognition. Since then, many deep learning models using residual connections (e.g., the Transformer model for natural language process [8]) have been developed. In particular, deep MLP with a residual connection can solve nonlinear regression problems such as climate prediction [9, 10].

We extend the residual deep NN models as an effective and flexible procedure to learn the abstract relationship between aerodynamic coefficient changes and the shape changes from ablation (illustrated Fig. 1). To the best of our knowledge, there is no explicit treatment of data-driven methods for predicting aerodynamic coefficient changes resulting from the shape changes due to ablation. Our work focuses on the prediction of aerodynamic coefficient changes with the following three core contributions:

1) We built a deep NN model with residual connection for nonlinear regression to predict aerodynamic coefficient changes due to shape transformations.
2) In addition to model building, we developed a relatively large dataset (order of 10^5) by incorporating an extended 3-D shape configuration and a wide flow condition range.

3) We applied the K-fold cross-validation and ensemble averaging methods to reduce the variance and improve the accuracy of the prediction.

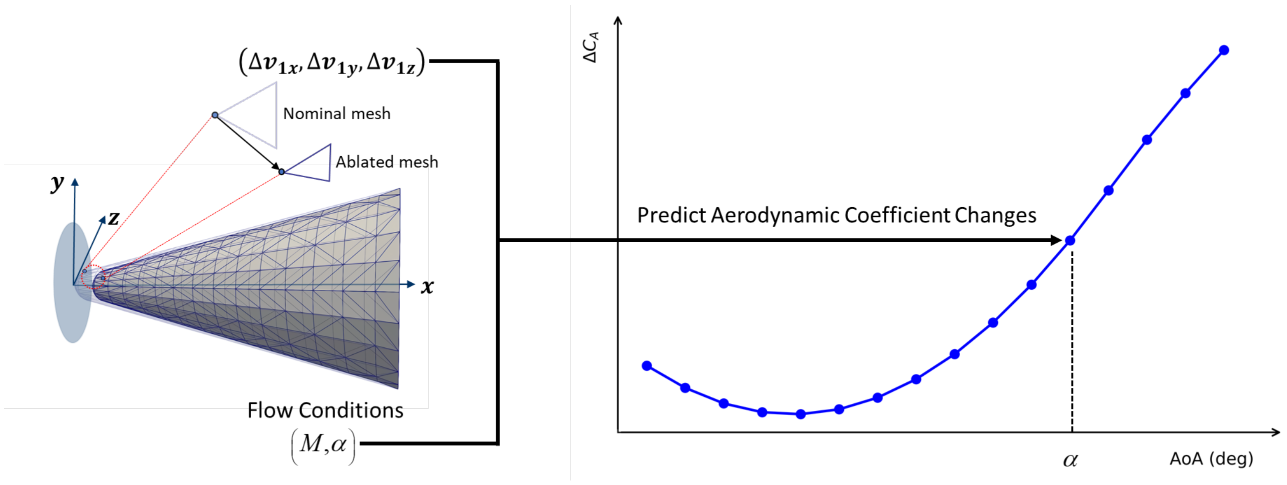


Figure 1. Schematic diagram: NN-based prediction of aerodynamic coefficient changes resulting from ablation

II. Preliminary: Nonlinear Regression with Neural Network

This section describes the neural network modeling approach to the nonlinear regression problem of predicting aerodynamic coefficient changes due to the cone shape changes caused by thermal ablation. A nonlinear regression model generally consists of a nonlinear function of various linear combinations of weights and biases (or the parameters) that outputs the prediction values given a set of input values. The precise physical relationship between the shape changes and the resulting changes in aerodynamic coefficients is not straightforward to describe in mathematical expressions. As noted earlier, it has been the convention to run a computer program to estimate the aerodynamic coefficients of three-dimensional shapes. Accordingly, we turn to machine learning models/approaches that can learn the underlying relationship between shape and aerodynamic coefficient changes. The proceeding subsections introduce the structure of a basic neural network model, which is the building block of more complex architectures, and the universal function approximation theorem for the neural network-based nonlinear regression.

A. Multilayer Feedforward Neural Network

The simplest neural network model is the multilayer feedforward neural network known as a multilayer perceptron (MLP). It contains one or more hidden layer(s) between the input and output layers. Each hidden layer involves weight and bias (neurons of the model), which are parameters to be optimized. Mathematically, we can describe the general input-output relationship for a multilayer neural network with a single hidden layer as follows. Let d_{in} and d_{out} represent the dimensions of the input feature vector ($\boldsymbol{x} \in \mathbb{R}^{d_{in}}$) and output (vector $\hat{\boldsymbol{y}} \in \mathbb{R}^{d_{out}}$), respectively. If the layer is *p*-dimensional, i.e., has *p* neurons, the data representation in the latent space is given by

$$\boldsymbol{p} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}\,, \tag{1}$$

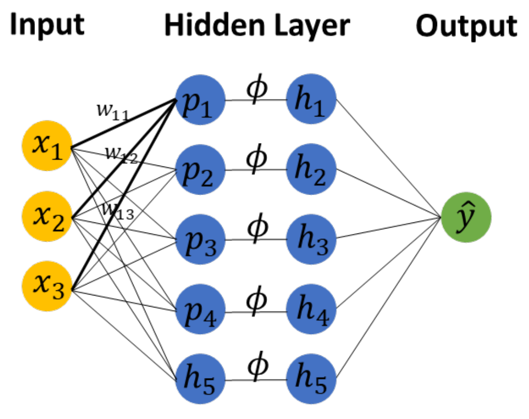
where $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times p}$ is a weight matrix and $\boldsymbol{b} \in \mathbb{R}^{p}$ is a p-dimensional bias vector. The intermediate output of the hidden layer is followed by a nonlinear activation function ($\phi : \mathbb{R}^p \to \mathbb{R}^p$), which is a sufficiently differentiable function applied to each component of $\boldsymbol{p}$. Accordingly, the output $\boldsymbol{h}$ of the hidden layer is given as

$$\boldsymbol{h} = \phi(\boldsymbol{p}) = \phi\left(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}\right). \tag{2}$$

The final output, or the prediction, is denoted by $\hat{\boldsymbol{y}}$ ($\in \mathbb{R}^{d_{out}}$) obtained as a function of $\boldsymbol{h}$ as follows

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_1^T \boldsymbol{h} + \boldsymbol{b}_1 = \boldsymbol{W}_1^T \left(\phi\left(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}\right)\right) + \boldsymbol{b}_1\,. \tag{3}$$

Eq. (1) shows that a multilayer feedforward neural network is associated with two sets of weight and bias parameters ($\boldsymbol{\theta} = (\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{W}_1, \boldsymbol{b}_1)$). Fig. 2 illustrates a feedforward neural net with five (= *p*) neurons on one hidden layer.



$$\hat{y} = \boldsymbol{W}_1^T \boldsymbol{h} + \boldsymbol{b}_1 \quad \boldsymbol{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix} = \phi(\boldsymbol{p})$$

$$\boldsymbol{p} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ \vdots & \ddots & \vdots \\ w_{15} & w_{25} & w_{35} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}$$

Figure 2. A simple multilayer feedforward neural net with five neurons

B. Deep Neural Network

A multilayer feedforward neural network can approximate any continuous function with an arbitrary precision provided the number of neurons is sufficiently (exponentially) large [11-13]. However, in practice, using many hidden layers is computationally intractable. Instead, empirical evidence in many real applications indicates that deep networks can learn complex functions. They are more expressive as they extend the modeling capacity of shallow networks to learn abstract and nonlinear relationships between input and output values. In terms of computational complexity during training, deep networks require far fewer neurons (and thus significantly fewer free weights) than shallow networks to achieve the same precision. The data representation at the lth hidden layer is as follows

$$\boldsymbol{h}_l = \phi(\boldsymbol{p}_{l-1}) = \phi\left(\boldsymbol{F}_l(\boldsymbol{h}_{l-1}, \boldsymbol{\theta}_l)\right) = \phi\left(\boldsymbol{W}_l^T \boldsymbol{h}_{l-1} + \boldsymbol{b}_l\right), \tag{4}$$

where $\boldsymbol{F}_l(\boldsymbol{h}_{l-1}, \boldsymbol{\theta}_l)$ is the lth linear hidden layer parametrized by $\boldsymbol{\theta}_l$ $(=(\boldsymbol{W}_l, \boldsymbol{b}_l))$ - an affine function followed by a nonlinear activation function. The lth layer takes $\boldsymbol{h}_{l-1}$ as input from the previous layer (l-1). The output ($\boldsymbol{h}_l$) is subsequently used as an input to the next layer (l+1). Thus, the data flow through the network from the input layer to the final output layer. Suppose $\boldsymbol{h}_{l-1}$ and $\boldsymbol{h}_l$ are p- and q-dimensional, respectively. Then, $\boldsymbol{W}_l \in \mathbb{R}^{p\times q}$ and $\boldsymbol{b}_l \in \mathbb{R}^q$. The deep neural network model $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{\theta})$ is constructed as a composition of individual $\boldsymbol{F}_l$'s for $l = 1, \ldots, L$ as

$$\hat{\boldsymbol{y}} = \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{F}_L\left(\boldsymbol{F}_{L-1}\left(\ldots \boldsymbol{F}_2\left(\boldsymbol{F}_1(\boldsymbol{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2\right)\ldots, \boldsymbol{\theta}_{L-1}\right), \boldsymbol{\theta}_L\right). \tag{5}$$

This composition produces a continuous piecewise linear function [14] that learns from training data to generate predictions for unseen input data. Without loss of generality, we set $\boldsymbol{x} = \boldsymbol{h}_0$ at the input layer and $\hat{\boldsymbol{y}} = \boldsymbol{h}_L$ at the final step of the model.

C. Nonlinear Regression Problem

This section discusses the general form of nonlinear regression using deep neural networks. Given an input-output pair (x, y) with N samples in the training set, we assume there exists an unknown, arbitrary complex function $f(x_i)$ that maps x_i to y_i with a relation $y_i = f(x_i) + \varepsilon_i$ (i = 1, …, N) where ε (~ $N(0, \sigma^2)$) is a zero-mean Gaussian noise. We construct a learning function of a deep neural net model ($\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{\theta})$) close to the true function ($f(x)$) using the (stochastic) gradient descent optimization.

III. Dataset Generation

This section outlines the construction of the training dataset (input and output data) used in this study. We first generate a series of synthetic aerodynamic shapes resulting from ablation. We then model the shape transformation by taking the difference between the ablated and initial cone shapes expressed in the mesh (or cell) vertex coordinates. We define the input feature vector $\boldsymbol{x}$ ($\in \mathbb{R}^{d_{in}}$) composed of aerodynamic shape transformation data ($\boldsymbol{z} \in \mathbb{R}^{m}$) and additional features ($\boldsymbol{u} \in \mathbb{R}^{3}$). The components of $\boldsymbol{u}$ are the nose radius (R_N) and flow conditions (Mach number (M) and angle of attack (α)). The shape is expressed using the mesh vertex coordinates ($i = 1, \ldots, N_m$) as follows

$$\boldsymbol{n}_i = \left(n_x, n_y, n_z\right)_i \tag{6}$$

$$\boldsymbol{v}_{1,i} = \left(v_{1x}, v_{1y}, v_{1z}\right)_i \tag{7}$$

$$\boldsymbol{v}_{2,i} = \left(v_{2x}, v_{2y}, v_{2z}\right)_i \tag{8}$$

$$\boldsymbol{v}_{3,i} = \left(v_{3x}, v_{3y}, v_{3z}\right)_i \tag{9}$$

where N_m is the number of triangular meshes. We fix N_m as 840 for uniformity of configuration. In Eqs. (6)-(9), $(n_x, n_y, n_z)_i$ is the normal vector of the ith mesh, and $\boldsymbol{v}_{j,i}$ ($= (v_{jx}, v_{jy}, v_{jz}), j = 1, 2, 3$) represents the coordinates of the jth vertex defining the ith mesh. A total of $9N_m$ (= 7360) values are used to define the initial (nominal) shape of the cone ($\boldsymbol{v}^{(n)}$) as follows

$$\boldsymbol{v}^{(n)} = \left\{\bigcup \boldsymbol{v}_{j,i}\right\}^{(n)}. \tag{10}$$

Given a nominal cone shape, we generate a series of (N_A) ablated shapes defined as $\boldsymbol{v}^{(n),(k)} = \left\{\bigcup \boldsymbol{v}_{j,i}\right\}^{(n),(k)}$ ($k = 1, \ldots, N_A$). As a result, there are a total of ($N_C \times N_A$) ablated shapes derived from N_C nominal shapes. Accordingly, the kth shape transformation from the nth nominal can be expressed as $\Delta\boldsymbol{v}^{(n),(k)} = \left\{\bigcup \Delta\boldsymbol{v}_{j,i}\right\}^{(n),(k)}$ where $\Delta\boldsymbol{v}_{j,i} = \left(\Delta v_{jx}, \Delta v_{jy}, \Delta v_{jz}\right)_i$ is the change in vertex location due to ablation defined as

$$\Delta v_{jx}^{(n),(k)} = v_{jx}^{(n)} - v_{jx}^{(n),(k)}, \tag{11}$$

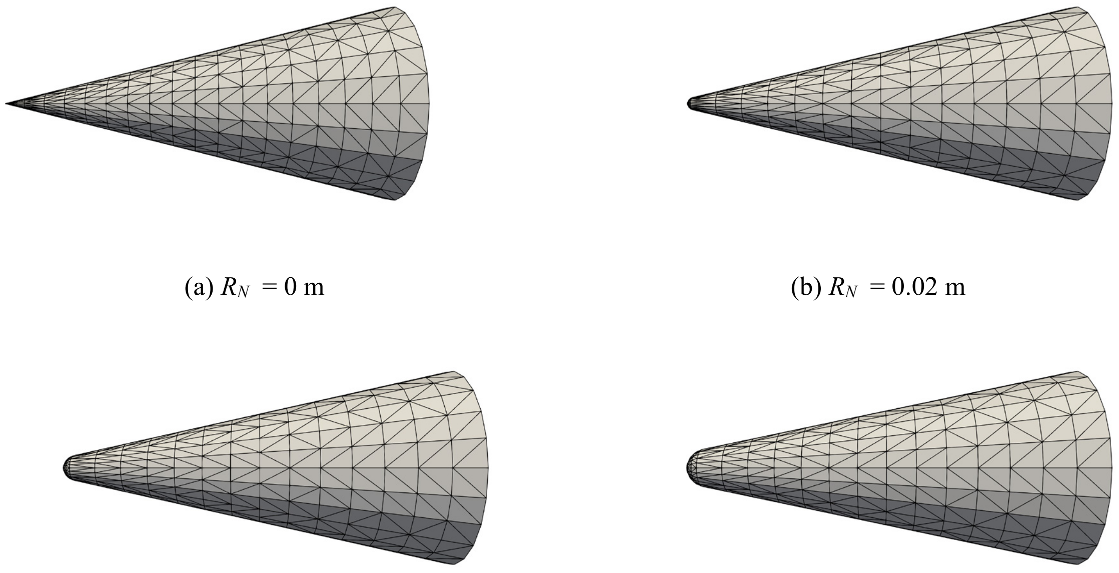
$$\Delta v_{jy}^{(n),(k)} = v_{jy}^{(n)} - v_{jy}^{(n),(k)}, \tag{12}$$

$$\Delta v_{jz}^{(n),(k)} = v_{jz}^{(n)} - v_{jz}^{(n),(k)}. \tag{13}$$

The shape transformation data (z) consists of the initial and deformed cone shapes and is expressed as

$$z = \left\{ v^{(n)} \bigcup \Delta v^{(n),(k)} \right\}_{k=1,\ldots,N_A}^{n=1,\ldots,N_R}. \tag{14}$$

Fig. 3 shows the four nominal 3D cone shapes based on the four nose radius (R_N) values – 0 *m*, 0.02 *m*, 0.04 *m*, and 0.06 *m*. Note that these nominal shapes are identical except for the difference in the nose radius (length: 1.2 *m*, fineness ratio 2.0).



(a) R_N = 0 m

(b) R_N = 0.02 m

(c) R_N = 0.04 m

(d) R_N = 0.06 m

Figure 3. Four nominal 3-D cone shapes with different nose radii (*R_N*)

We generated 400 ablated shapes by applying the ablation model to the nominal cone shapes. Fig. 4 shows a sample cone shape generated with a bluntness radius of 0.06 *m*. A stagnation point was selected randomly in the range between the cone tip and the closest grid, leading to an asymmetry in the ablated shape. The ablation depth (d_0) at the stagnation point was chosen randomly between 1/10 and 2/10 of the radius at the base. We assume the ablation occurs at the stagnation point, and the ablation depth for each vertex decreases towards the base because less ablation occurs farther away from the nose.

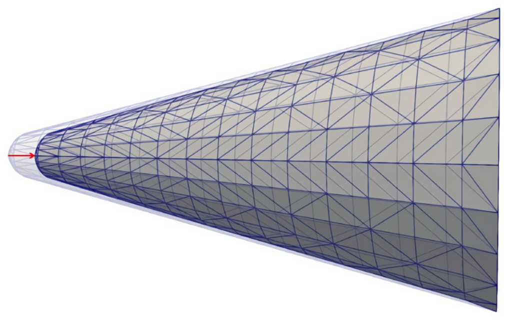


Figure 4. An example of an ablated cone shape

The aerodynamic coefficients of the cones in hypersonic flows were computed based on the modified Newtonian law [15]. When the freestream strikes a three-dimensional body, the pressure coefficient at an arbitrary mesh i on the surface is expressed as

$$C_{p,i} = C_{p_{\max}} \sin^2 \theta_i, \tag{15}$$

where θ_i is the angle between the freestream and mesh i [16]. The maximum pressure coefficient ($C_{p_{\max}}$) at the stagnation point is expressed as

$$C_{p_{\max}} = \frac{2}{\gamma M_\infty^2}\left\{\left[\frac{(\gamma+1)^2 M_\infty^2}{4\gamma M_\infty^2 - 2(\gamma-1)}\right]^{\gamma/(\gamma-1)}\left[\frac{1-\gamma+2\gamma M_\infty^2}{\gamma+1}\right]-1\right\}, \tag{16}$$

where γ is the specific heat ratio (= 1.4) and M_∞ is the freestream Mach number. The drag coefficient (C_D) and the lift coefficient (C_L) of the cone can be obtained by integrating the pressure coefficient ($C_{p,i}$) over the entire surface with proper projections. These coefficients are used to compute the drag (D) and lift (L) forces. The normal force (N) and axial force (A) can be obtained using the relationship expressed as follows

$$N = L\cos\alpha + D\sin\alpha, \tag{17}$$

$$A = -L\sin\alpha + D\cos\alpha, \tag{18}$$

where α is angle of attack. The pitching moment coefficient C_M is obtained by integrating the product of the pressure coefficient ($C_{p,i}$) projected onto the x-y plane of each mesh and the moment arm between the mesh center and the center of gravity over the entire surface (see Fig. 1).

The raw data consists of three parts – shape transformation data ($\boldsymbol{z}$), flow condition data ($\boldsymbol{u}$), and corresponding changes in aerodynamic coefficients ($\Delta\boldsymbol{C}_A$, $\Delta\boldsymbol{C}_N$, $\Delta\boldsymbol{C}_M$). We have 1604 cone shapes, consisting of 4 (= Nc) nominal cones and 1600 (= N_A) ablated shapes. The flow conditions include 11 Mach numbers ranging from 5.0 to 10.0 (increments of 0.5) and 16 angles of attack from -10 ° to 20 ° (increments of 2 °). Aerodynamic coefficients (C_N, C_A, and C_M) are calculated for each given shape at the specified flow conditions. Consequently, a total of 282,304 data points are generated for this study.

IV. Learning Methodology

We train our deep neural network model based on the supervised learning approach. The input features are the shape transformation data, and the ground truths ($\boldsymbol{y}$) are the corresponding changes in aerodynamic coefficients ($\Delta\boldsymbol{C}$). The error between the aerodynamic coefficient change ($\boldsymbol{y}$) and its prediction ($\hat{\boldsymbol{y}}$) is used as the loss function to be minimized by the stochastic training algorithm. While deep models offer flexibility in approximating any nonlinear function, their training algorithm is stochastic; each time a model is trained, it may learn a different mapping from input to output. This stochastic nature could lead to different prediction (test) performances depending on the training dataset, model configuration, weight initialization, and adopted training scheme. As a result, a single run of fitting a model can suffer from high variance in the test performance. In this case, repeated runs and combining the results can alleviate the high variance and improve the prediction performance using ensemble learning [17, 18]. In particular, this study adopts the *K*-fold cross-validation to generate different trained models and combines the results by model averaging.

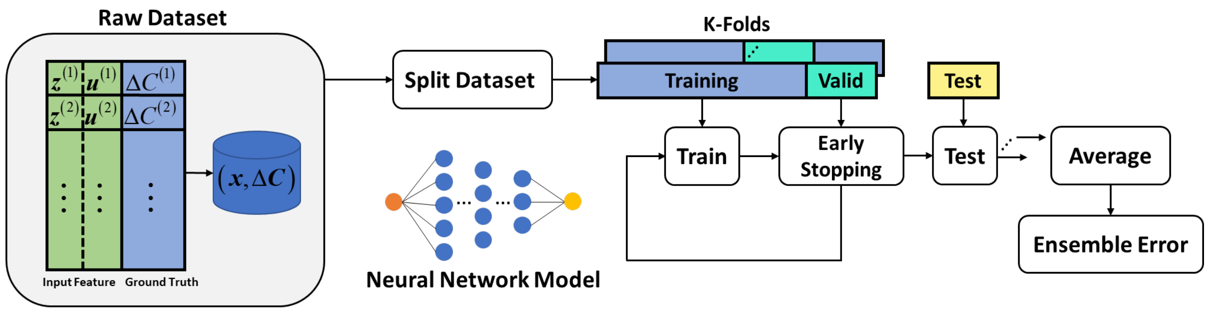


Figure 5. Outline of the learning methodology: *K*-fold cross-validation and model averaging

Fig. 5 illustrates our learning methodology that fits multiple models for each data split in *K*-fold cross-validation (*K* = 5). A fraction of the raw dataset is used as the test set for evaluating the performance of each model. The remaining dataset is then randomly split into K non-overlapping subsets, or "folds." The training procedure cycles through the K folds using (K-1) folds for training and the last single fold for validation. In this manner, each fold serves as validation exactly once, and we can fully exploit the available training dataset. By construction, we keep the initial shape data $\boldsymbol{v}^{(n)}$ and the corresponding label (aerodynamic coefficients) inside the training set to provide the baseline data during learning. Since the training and validation datasets are mutually exclusive, there is a low correlation between the predictive performances of models, and we can prevent data leakage during training. An ensemble of trained models performs better on average than a single best model since combining different predictions can average outliers and generate more accurate predictions and better generalization [19, 20]. Note that this approach is different from the commonly used cross-validations in selecting a model [21] or hyperparameter tuning for model configuration (e.g., choosing the number of neighbors in a k-nearest neighbor classifier).

A. Data Preprocessing

The step following the split of the raw dataset into K-folds before training the model is preprocessing each fold's dataset. This step transforms the raw data points within a similar scale so that the training algorithm converges faster. Specifically, we use the mean and standard deviation of the training fold to center the values at zero and standardize the variance to unity (Z normalization). By scaling data using the distribution of the training set within each cycle, we ensure that information from the validation or test set does not leak into training. This step maintains validation and test data separate from training for an unbiased model evaluation. Otherwise, data leakage into training can overestimate the expected performance [22]. In addition, the scaling factors for aerodynamic coefficient data (that belong to each cycle's training) are stored and later used to reconstruct predicted outputs for comparison with the ground truths in the test set.

B. Model Architecture

Fig. 6 illustrates the architecture of the proposed deep residual neural network (DRNN) model used to learn the nonlinear mapping from input feature data to output data (change in aerodynamic coefficients). Our proposed model consists of two inputs, which are processed separately by the model. The first input is the shape change data, which is fed into a fully-connected (FC) residual block. The second input consists of the flow condition and nose radius data and is processed by a Linear-BatchNorm-LeakyReLU (LBR) sub-block. Fig. 6 describes the building blocks used in our model. The model learns inherent features separately for each input data and projects the latent representation in the hidden dimension (d_h). Then, both latent representations are horizontally concatenated, and the augmented data is fed into another FC residual block for further learning. Finally, the final fully-connected layer projects the hidden data into a one-dimensional value, which is the predicted change in the aerodynamic coefficient.

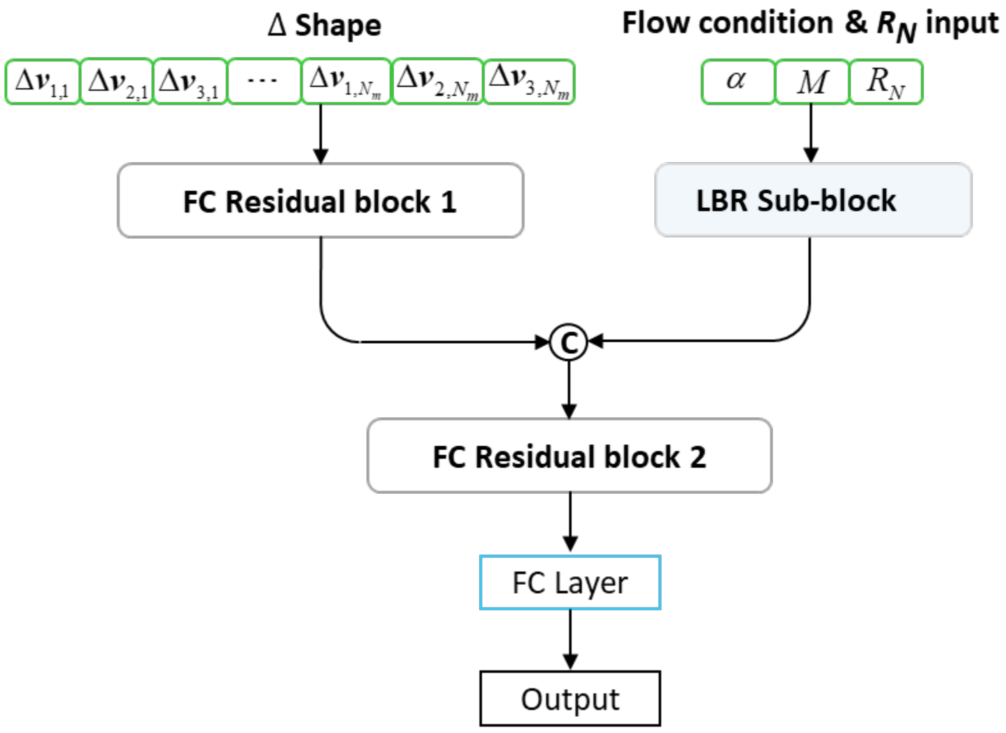


Figure 6. Proposed deep residual neural network (DRNN) model.

The simplest building block in our proposed model is the LBR sub-block, consisting of a linear (fully-connected) layer followed by batch normalization and a LeakyReLU activation. We use fully-connected layers in processing initial input values in a simple list of real numbers. Each neuron in the first fully-connected layer learns each fed input value. After the data are projected onto a hidden space, we standardize them for each mini-batch before they flow into the next layer. This intermediate step is known as *batch normalization*, a popular technique for training deep neural networks. Initially, the batch normalization step was thought to speed up the learning process by reducing the internal covariate shift of data flowing through the hidden layers [23]. However, it is now widely understood that the step smoothes out the optimization landscape (rather than reducing internal covariate shift) by re-parametrizing the underlying optimization problem [24]. The application of batch normalization in our model allows larger learning rates without sudden changes in the landscape (e.g., flat regions with vanishing gradients and sharp local minima with exploding gradients). This step is critical for stabilized training, leading to faster learning and making the model less sensitive to weight initialization. After batch normalization, the hidden data is activated by a nonlinear LeakyReLU function based on the generic ReLU activation. The LeakyReLU has a small slope for negative values, allowing it to capture negative representations. It is formally defined as follows.

$$LeakyReLU = \max\left(x, ax\right),\ a > 0 \tag{19}$$

We note that the nonlinear activation functions in each LBR sub-block enable the model to learn complex and nonlinear representations, which is the main goal of using a neural network model.

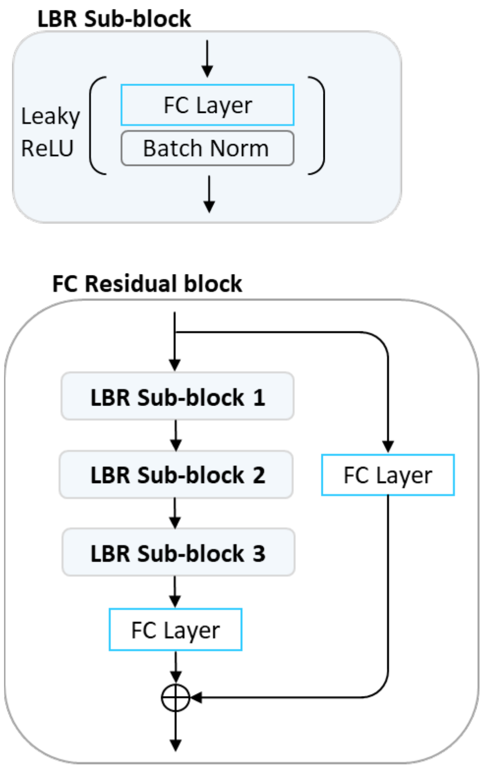


Figure 7. Building blocks (LBR Sub-block and FC Residual block) used for constructing the proposed model

The next building block is the FC residual block, which consists of three LBR sub-blocks and two fully-connected layers. One notable aspect is the residual connection [7], which provides another path for data to flow from the input to the latter, deeper parts of the model. In Fig. 7, this alternate path bypasses the three LBR sub-blocks and directly feeds the incoming data into a designated hidden space by a linear transformation. The resulting two data representations in the same hidden space are then added to form a more data-enriched representation. Mathematically, the element-wise addition is expressed as $F(x)+W^T x$, where x is the input to the block, $F(x)$ is the bypassed layer, and W is the weight for linear transformation.

We expect that the combination of LeakyReLU activation and residual connection can significantly improve the gradient flow through the hidden layers. During the backpropagation, the loss signal is multiplied by the derivative of the activation function. With a generic ReLU, approximately half of the cases would result in zero gradients. However, LeakyReLU preserves the negative signals and avoids zero gradients. Additionally, the residual connection helps the gradient flow be kept intact because the additional path "reminds" the previous representations when the primary path results in negligible gradients. This characteristic is especially helpful in avoiding the vanishing gradient problem. In an "unraveled view" [25], networks with residual connection act like an ensemble of shallow networks. Indeed, deeper networks contribute less to learning representations as the internal gradients might vanish, and residual connections are the shorter paths that carry gradients from early layers to the latter parts. In this work, the ablation creates small but meaningful changes to the nominal shape. It is necessary to keep the gradient flow intact to capture the small perturbations in aerodynamic properties – the model can learn the shape transformations and their effect on the aerodynamic property. Table 1 summarizes the specific model configuration used in this study. The total number of trainable parameters in this configuration is approximately 18.5 million.

Table 1. Model configuration for deep residual neural network

Input Layer	Δ Shape $\left(\Delta \boldsymbol{v}_{1,1}, \Delta \boldsymbol{v}_{2,1}, \Delta \boldsymbol{v}_{3,1}, \ldots\right)$			Flow condition & R_N $\left(\alpha, M, R_N\right)$	
Before concatenation	FC Residual Block 1	LBR 1	(7560, 2048)	LBR	(3, 64)
		LBR 2	(2048, 1024)		
		LBR 3	(1024, 256)		
		FC layer	(256, 64)		
		FC layer (residual)	(7560, 64)		
Concatenation	(64,) ∪ (64,) = (128,)				
After concatenation	FC Residual Block 2	LBR 1	(128, 256)	–	–
		LBR 2	(256, 128)		
		LBR 3	(128, 128)		
		FC layer	(128, 128)		
		FC layer (residual)	(128, 128)		
Output Layer	FC layer		(128, 1)		

C. Data split and Hyperparameters

The dataset is split into separate folds to avoid overfitting during the supervised learning process. An overfit model performs well on a training set, but its performance is significantly inferior on an unseen dataset. We can diagnose overfitting by monitoring the learning curves and comparing the training loss with the validation loss. Since the validation dataset is not used in training, the validation loss is a reasonable measure of the model's generalization performance. To avoid overfitting, we use an early stopping criterion during training. The training stops if the model's performance does not improve compared to the best-recorded validation loss for a prescribed number of epochs (stop patience). During the training process, we also use a learning rate scheduler that updates the learning rate by multiplying a factor (learning rate factor, usually smaller than 1.0). In particular, we use the ReduceLRonPlateau scheduler that reduces the learning rate when the validation loss does not improve for a prescribed number of epochs (scheduler patience). When learning stagnates, i.e., oscillates without improvement, the model often benefits from a reduced learning rate because it can take smaller steps to the optimal solution and escape the oscillatory behavior.

Splitting the dataset into subgroups with reasonable sizes is important. For example, a relatively small test set may not effectively represent all cases in the feature space (i.e., population), and the test results may not be reliable. Table 2 summarizes the train-validation-test split in this work. For each nominal shape, i.e., for a given nose radius, we generated 400 transformed shapes. Since our model aims to learn shape transformations due to ablation, we select a portion (66 shapes) of the transformed shapes as the test set. For each shape transformation, there are 4 (nose radius) × 11 (Mach number) × 16 (angle of attack) = 704 data points. So, the test set consists of 66 × 704 = 46,464 data points.

Table 2 Data split proportions (entire dataset)

Proportion	Training	Validation	Test
Number of data points	188,672	47,168	46,464
Fraction (%)	66.8	16.7	16.5

The training process involves several hyperparameters. We use the Root Mean Squared Error (RMSE) loss function and the Adam optimizer to fit the model. For model regularization, we use an L_2 penalty of 0.01, which is added to the loss function to penalize large weights. For each epoch, we train a mini-batch of 4096 samples. It is known that [26, 27] a large batch, while often leading to a reduced generalization, reaps the benefits of more data parallelization and faster training. On the contrary, small batches are considered noisy since fewer samples result in less accurate gradient estimates, helping the algorithm escape from “sharp minimizers” [26]. For the current problem, however, we empirically found that large-batch training does not significantly degrade the generalization performance when compared with small-batch training (such as 32 and 64). In addition, small batches were too noisy, suggesting a substantially lower learning rate for compensation. Table 3 summarizes the training hyperparameters used in this work. Algorithm 1 explains the learning methodology used to train our deep residual neural network.

Table 3. Training hyperparameters

Hyperparameter	Value
Batch size	4096
Number of epochs	500
Initial learning rate	4e-6
Learning rate factor	0.7
Scheduler patience	40
Stop patience	60
L_2 penalty	0.01
LeakyReLu slope	0.05

Algorithm 1 Learning method using early stopping on *K different* data folds

```
1   Input: training set ($\{(\boldsymbol{z}, \boldsymbol{u}), \Delta\boldsymbol{C}\}$), # of epochs ($E$), initial learning rate ($\eta$), learning rate factor
        ($\tau$), scheduler patience ($\kappa$), stop patience ($\varsigma$)
2   Hold out the test set and split the remaining dataset into $K$ folds
3   for fold $k = 0, \ldots, K-1$ do
4   |   Initialize trainable parameters $\theta$
5   |   Fit and transform $k^{th}$ fold's training set
6   |   for epoch = $1, \ldots, E$ do
7   |   |   Initialize epoch counter and train a model with Adam
8   |   |   If the validation score does not improve for $\kappa$ epochs
9   |   |   |   Update $\eta \leftarrow \tau\eta$ and reset epoch counter
10  |   |   If the validation score does not improve for $\varsigma$ epochs
11  |   |   |   Early-stop training
12  |   Evaluate model performance on the test set
13  Ensemble performance $\leftarrow$ final prediction averaged over $K$ different predictions
14  Return: ensemble error
```

V. Experimental Results and Discussion

This section discusses the experimental results of training the DRNN model and testing the prediction capability on unseen test data. The training used a computer system with Intel Core i7-10700K CPU (@3.80GHz, 96GB RAM) and NVIDIA GeForce RTX 3070Ti (8GB) graphic card running on Windows 10 Professional.

A. Evaluation metrics

The evaluation metric shows how well the model performs on an unseen test dataset, thereby assessing the prediction accuracy. The evaluation metric selected in this work is the RMSE expressed as

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2} \ . \tag{20}$$

The RMSE, given in the units of coefficients, represents how far the predicted values are away from the true values on average.

B. Training histories

In training our model, it is desirable to have a reasonable balance between model complexity and generalization capability. We can observe the model's learning process by observing the training and validation loss histories. Fig. 8 to Fig. 10 are the training history plots (with the early stopping indication) for ΔC_A, ΔC_N and ΔC_M, respectively, for the fold number selected between 0 and 4. They show the losses in the units of RMSE between the predicted output and the true labels during each epoch. All plots are accompanied by the corresponding learning rate history, which shows how the learning rate was adjusted during training. The training loss is directly used in updating the model's weights and biases, whereas the validation loss is only tracked to gauge performance on unseen data. Usually, as training proceeds, both training and validation losses decrease. After some time, they begin to plateau, and eventually, the validation loss begins to increase, which signals that the model is overfitting to the training data. To prevent overfitting, we keep a record of the model weights that show the highest validation performance until the early stopping criterion is met. The recorded model is then selected as the current fold's trained model for evaluation on the held-out test dataset.

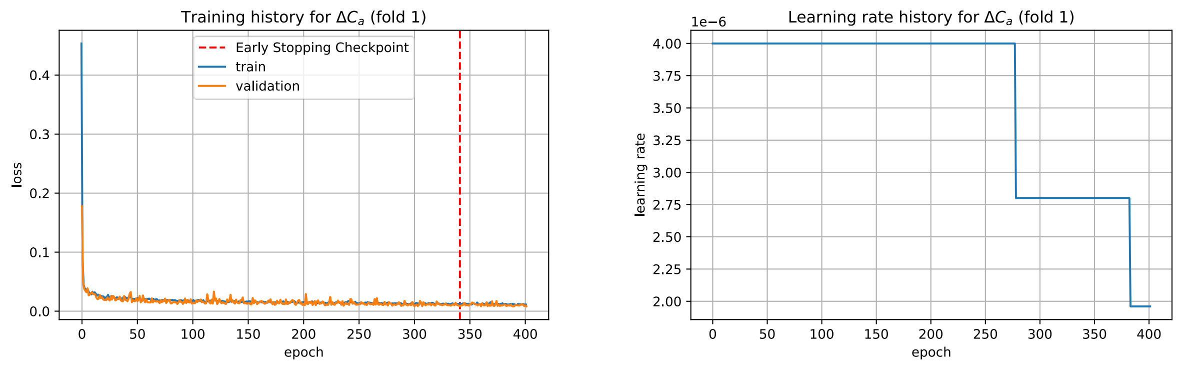


Figure 8. Training history for ΔC_A on fold 1 (left) and its learning rate history (right)

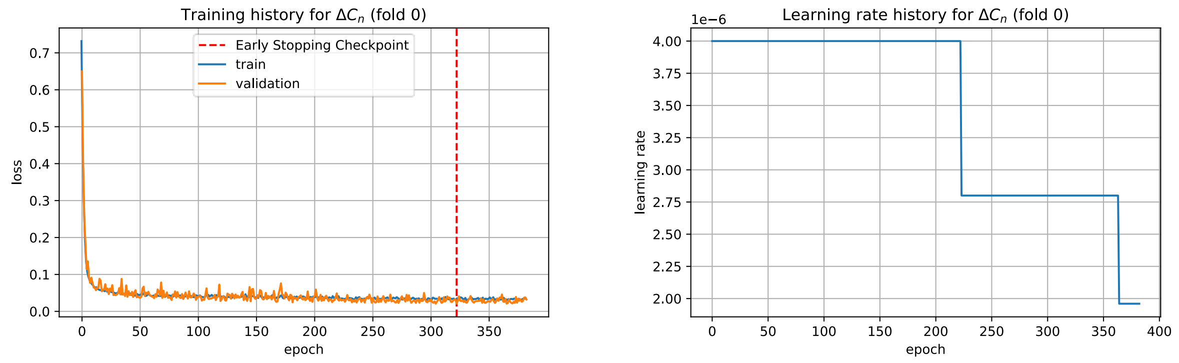


Figure 9. Training history for ΔC_N on fold 0 (left) and its learning rate history (right)

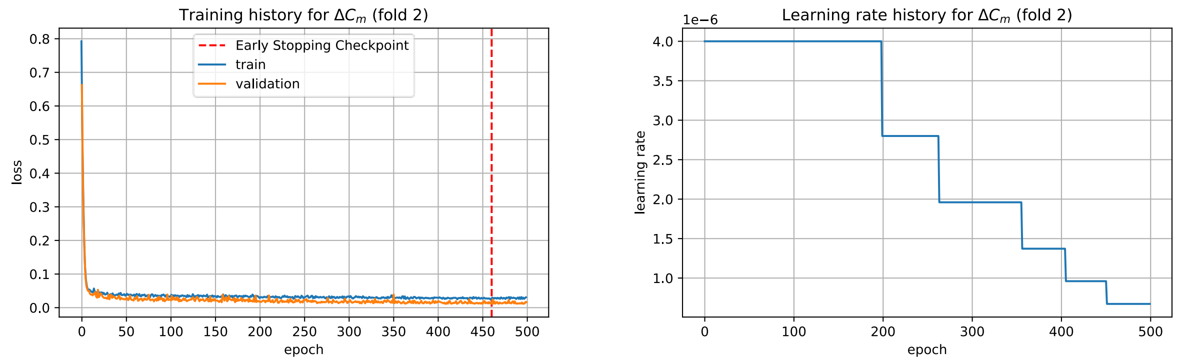


Figure 10. Training history for ΔC_M on fold 2 (left) and its learning rate history (right)

C. Experimental Results

During training, we select the model that shows minimum validation loss recorded by the early stopping checkpoint in Figs. 8- 10. Table 4 presents the minimum validation loss values recorded by each selected model on separate folds.

The values are generally consistent in folds, but there are discernable differences among trained models, which are attributable to differences in training-validation folds.

Table 4. Validation loss of selected models across each fold for all aerodynamic coefficients: training loss and test loss (standardized using the distribution of training fold)

Aerodynamic coefficient	Fold #	Training set	Test set
		Min. Validation loss, 10^{-2}	Standardized Test loss, 10^{-2}
ΔC_A	0	1.01	1.01
	1	0.80	1.00
	2	0.96	1.01
	3	1.19	0.87
	4	0.91	0.89
ΔC_N	0	2.07	2.07
	1	1.75	1.83
	2	1.78	1.98
	3	2.10	1.99
	4	2.61	2.60
ΔC_M	0	1.65	1.69
	1	1.28	1.33
	2	1.12	1.12
	3	1.60	1.65
	4	1.27	1.26

After training, we evaluated the prediction performance from each of the five models on 46,464 ground truths in the test set. Table 5 summarizes the relative performance of each fold's trained models for all aerodynamic coefficient changes. We observe test losses with similar order across folds for all aerodynamic coefficients, indicating the model's stability in the prediction. In addition, we note that model selection may have biases as tuning hyperparameters on the validation set is incrementally incorporated into model configuration. If the model is biased towards the validation set, there may be large differences between the validation and test losses. Each model's validation and test losses (evaluated after standardization) are very similar in this study. The results suggest that the trained models are not biased to the validation set and can generalize well to unseen data.

For all aerodynamic coefficients, the ensemble performance, measured as the average over five different predictions, is better than any single best model's prediction. Fig. 11 illustrates how individual predictions from different folds can be combined to generate more accurate predictions. Predicted ΔC_A values are notably close to the ground truth, capturing a nonlinear trend versus the angles of attack. Note that the individual predictions are slightly different; some overestimate while others underestimate ΔC_A. This inherent variability in models offers a sense of diversity that reduces variance in the final output when we average the predictions. Also, we observe that this improves the ability to generalize beyond the training dataset. We can observe a similar trend in prediction results of ΔC_N and ΔC_M .

Table 5. Prediction test performance (losses reported as on the original test set)

Aerodynamic coefficient change	Fold #	Test loss, 10^{-4}	Ensemble average loss, 10^{-4}
	0	2.02	
	1	2.06	
ΔC_A	2	2.06	1.24
	3	1.73	
	4	1.77	
	0	1.27	
	1	1.15	
ΔC_N	2	1.24	0.84
	3	1.23	
	4	1.59	
	0	9.25	
	1	7.41	
ΔC_M	2	6.22	4.41
	3	9.00	
	4	6.89	

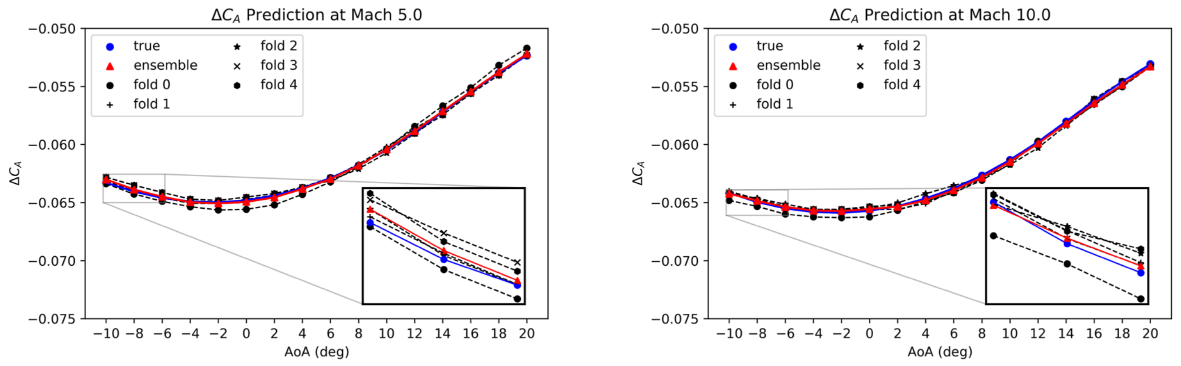


Figure 11. Ensemble average and individual fold predictions for ΔC_A (using DRNN, for selected *M*)

D. Comparison with multilayer perceptron (MLP) and Gaussian process (GP) regression model

To evaluate the performance of the proposed model, we compared its prediction performance with other approaches – the MLP model and the Gaussian process (GP) regression. We considered two types of MLP models with different numbers of neurons in a single hidden layer. The width of the MLP is set to be 1000 and 10,000, corresponding to approximately 7.5 million and 75 million trainable weights, respectively. Note that the number of parameters for the proposed DRRN stands in between the two MLPs with 1000 and 10,000 neurons. In training the MLP models, we used the same hyperparameters and training-validation folds as those used in the proposed model. The GP model was trained using the squared exponential kernel function (implemented in MATLAB with `fitrgp` function). Because of the enormous data size, the approximation method was used to estimate GP parameters. Finally, both the MLP and GP models adopted the *K*-fold cross-validation ($K = 5$) and ensemble averaging for a fair comparison.

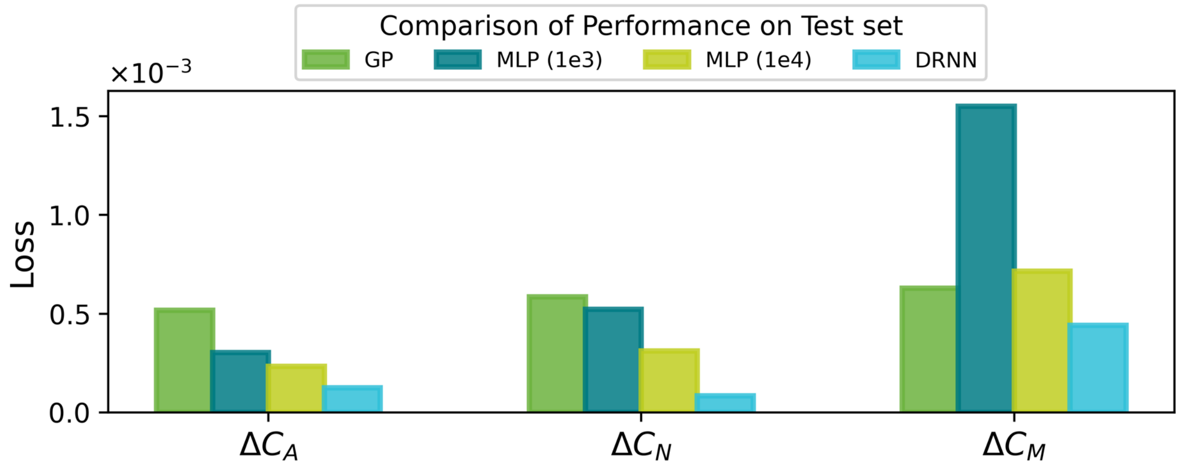


Figure 12. Performance comparison - ensemble-average RMSE losses for different models

Fig. 12 shows the test losses for all aerodynamic coefficients, highlighting the performance of different regression models. The neural network models (DRNN and MLP) have smaller test losses than the GP models, except for ΔC_M where the MLP models perform significantly worse than GP. We note that the GP models struggle with the high-dimension input, showing large test losses. As expected, the MLP predicts the aerodynamic coefficient changes better with more neurons. However, the exponentially growing number of trainable parameters with the increase in the width may cause computational/memory overload. The computer used in this study could not handle the MLP model with 100,000 neurons due to insufficient GPU memory. The DRNN model showed the lowest error for all cases, suggesting its superior prediction capability to other methods in unseen datasets.

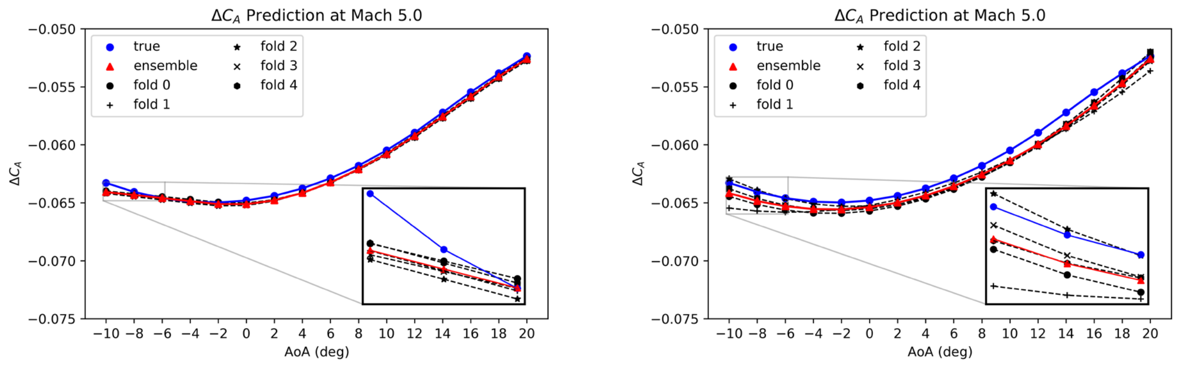


Figure 13. Prediction performance – MLP (left) and GP regression (right), M = 5.0

Fig. 13 shows the performance comparison between the MLP (10,000 neurons) and GP regression for predicting ΔC_A at $M = 5.0$. The two models did not perform as well as the DRNN model, particularly for ΔC_A, which is highly nonlinear. One notable aspect of MLP is that the ensemble averaging does not improve the accuracy since it does not introduce enough variability in the trained models. Note that only one hidden layer attempts to capture the inherent and abstract relationship between the input and output in the MLP. This architectural simplicity may have caused this result – individual predictions are similarly biased, and their ensemble does not average out the bias. The individual predictions using the GP regression show larger errors than the MLP model, leading to the ensemble average with a large bias.

VI. Conclusion

This paper introduces a data-driven approach using a deep residual neural network model to predict aerodynamic coefficient changes (axial/normal force and pitching moment coefficients) resulting from ablation. A low-cost, synthetic dataset consisting of 3-D nominal and ablated cones and the corresponding aerodynamic coefficient changes is used for learning. A prediction model based on deep residual neural networks that can learn abstract nonlinear relationships for accurate prediction is developed. We used K-fold cross-validation to generate five different prediction models to validate our methodology. Each model was tested on a separate holdout test set for performance evaluation. We found that these models show a reasonable level of generalization to unseen datasets for all aerodynamic coefficients. In addition, we combined the individual predictions and performed ensemble averaging that can effectively reduce the variance for improved accuracy. Performance comparison of the proposed model with the generic MLP and GP regression models demonstrated its prediction capability. For future work, we can extend the scope of the dataset to include more aerodynamic shapes, such as tangent ogive and Haack nose cones. In this case, we can use stratified *K*-fold cross-validation, where the folds are made with the class ratio the same as in the original dataset. This validation strategy would allow the model to capture intrinsic characteristics across general aerodynamic shapes without bias on a single shape.

Acknowledgments

This work was supported by Data-driven Flow Modeling Research Laboratory funded by Defense Acquisition Program Administration under Grant UD200046CD.

References

1. Zhang, Y., Sung, W. J., and Mavris, D. N. “Application of convolutional neural network to predict airfoil lift coefficient,” *2018 AIAA/ASCE/AHS/ASC structures, structural dynamics, and materials conference*. 2018, p. 1903.
doi: https://doi.org/10.2514/6.2018-1903
2. Yu, B., Xie, L., and Wang, F. “An Improved Deep Convolutional Neural Network to Predict Airfoil Lift Coefficient,” *Proceedings of the International Conference on Aerospace System Science and Engineering 2019*. Springer Singapore, Singapore, 2020, pp. 275-286.
doi: https://doi.org/10.1007/978-981-15-1773-0_21
3. Zelong, Y., Wang, Y., Qiu, Y., Bai, J., and Chen, G. “Aerodynamic Coefficient Prediction of Airfoils with Convolutional Neural Network.” 2019, pp. 34-46.
4. Lee, D. H., Lee, D., Lee, J., Lee, B. J., and Ahn, J. “Prediction of Multiple Aerodynamic Coefficients of Missiles using CNN,” *AIAA Scitech 2022 Forum*. 2022, p. 2439.
doi: https://doi.org/10.2514/6.2022-2439
5. Jacob, S. J., Mrosek, M., Othmer, C., and Köstler, H. “Deep Learning for Real-Time Aerodynamic Evaluations of Arbitrary Vehicle Shapes,” *arXiv preprint arXiv:2108.05798*, 2021.
doi: https://doi.org/10.48550/arXiv.2108.05798
6. Sabater, C., Stürmer, P., and Bekemeyer, P. “Fast Predictions of Aircraft Aerodynamics Using Deep-Learning Techniques,” *AIAA Journal* Vol. 60, No. 9, 2022, pp. 5249-5261.
doi: 10.2514/1.J061234
7. He, K., Zhang, X., Ren, S., and Sun, J. “Deep residual learning for image recognition,” *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770-778.
doi: 10.1109/CVPR.2016.90

8. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. "Attention is all you need," *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates Inc., Long Beach, California, USA, 2017, pp. 6000–6010.
doi: https://dl.acm.org/doi/10.5555/3295222.3295349

9. Chen, D., Hu, F., Nian, G., and Yang, T. "Deep Residual Learning for Nonlinear Regression," *Entropy* Vol. 22, No. 2, 2020, p. 193.
doi: https://doi.org/10.3390/e22020193

10. Jiang, C., Jiang, C., Chen, D., and Hu, F. "Densely connected neural networks for nonlinear regression," *Entropy* Vol. 24, No. 7, 2022, p. 876.
doi: https://doi.org/10.3390/e24070876

11. Hornik, K., Stinchcombe, M., and White, H. "Multilayer feedforward networks are universal approximators," *Neural Netw.* Vol. 2, No. 5, 1989, pp. 359–366.
doi: https://doi.org/10.1016/0893-6080(89)90020-8

12. Nielsen, M. A. *Neural Networks and Deep Learning*: Determination Press, 2015.

13. Cybenko, G. "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems* Vol. 2, No. 4, 1989, pp. 303-314.
doi: 10.1007/BF02551274

14. Strang, G. *Linear Algebra and Learning from Data*: Wellesley-Cambridge Press, 2019.

15. Lees, L. "Hypersonic Flow," *Journal of Spacecraft and Rockets* Vol. 40, No. 5, 2003, pp. 700-735.
doi: 10.2514/2.6897

16. Anderson, J. D. *Hypersonic and High-temperature Gas Dynamics*: American Institute of Aeronautics and Astronautics, 2006.

17. Naftaly, U., Intrator, N., and Horn, D. "Optimal Ensemble Averaging of Neural Networks," *Network: Computation in Neural Systems* Vol. 8, 1999.
doi: 10.1088/0954-898X/8/3/004

18. LeCun, Y., Bengio, Y., and Hinton, G. "Deep learning," *Nature*. Vol. 521, 2015, p. 256.

19. Bishop, C. M. *Neural Networks for Pattern Recognition*: Clarendon Press, 1996.

20. Krizhevsky, A., Sutskever, I., and Hinton, G. E. "ImageNet classification with deep convolutional neural networks," *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. Curran Associates Inc., Lake Tahoe, Nevada, 2012, pp. 1097–1105.
doi: https://doi.org/10.1145/3065386

21. Zhang, Y., and Yang, Y. "Cross-validation for selecting a model selection procedure," *Journal of Econometrics* Vol. 187, No. 1, 2015, pp. 95-112.
doi: https://doi.org/10.1016/j.jeconom.2015.02.006

22. Brownlee, J. *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*: Machine Learning Mastery, 2020.

23. Ioffe, S., and Szegedy, C. “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *International conference on machine learning*. PMLR, 2015, pp. 448-456.
doi: https://dl.acm.org/doi/10.5555/3045118.3045167

24. Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. “How does batch normalization help optimization?,” *Advances in neural information processing systems* Vol. 31, 2018.
doi: https://dl.acm.org/doi/10.5555/3327144.3327174

25. Veit, A., Wilber, M. J., and Belongie, S. “Residual networks behave like ensembles of relatively shallow networks,” *Advances in neural information processing systems* Vol. 29, 2016.
doi: https://doi.org/10.48550/arXiv.1605.06431

26. Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. “On large-batch training for deep learning: Generalization gap and sharp minima,” *arXiv preprint arXiv:1609.04836*, 2016.
doi: https://doi.org/10.48550/arXiv.1609.04836

27. Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. “Don’t decay the learning rate, increase the batch size,” 2017.
doi: https://doi.org/10.48550/arXiv.1711.00489