NASA/TM-20220006290

Predicting Two-Dimensional Airfoil Performance Using Graph Neural Networks

Paht Juangphanich
Glenn Research Center, Cleveland, Ohio

Justin Rush
Georgia Institute of Technology, Atlanta, Georgia

Natasha Scannell
University of Wisconsin-Milwaukee, Milwaukee, Wisconsin

April 2023

NASA/TM-20220006290

Predicting Two-Dimensional Airfoil Performance Using Graph Neural Networks

Paht Juangphanich
Glenn Research Center, Cleveland, Ohio

Justin Rush
Georgia Institute of Technology, Atlanta, Georgia

Natasha Scannell
University of Wisconsin-Milwaukee, Milwaukee, Wisconsin

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

April 2023

Acknowledgments

The authors would like to acknowledge the support of the Advanced Air Transport Technology (AATT) Project. This project allowed us to investigate the use of graph networks in aerospace applications.

This work was sponsored by the Advanced Air Vehicle Program at the NASA Glenn Research Center

Predicting Two-Dimensional Airfoil Performance Using Graph Neural Networks

Paht Juangphanich
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Justin Rush
Georgia Institute of Technology
Atlanta, Georgia 30332

Natasha Scannell
University of Wisconsin-Milwaukee
Milwaukee, Wisconsin 53211

Summary

Computer simulations require the use of meshes to simulate geometries. These meshes capture important geometric features of the design and can be used in machine learning modeling. This report explores the use of graph neural networks (GNNs) to learn features from two-dimensional (2D) airfoil designs represented as a set of nodes connected using edges. This type of network is common in aerospace applications: most geometries are represented as a mesh in order to perform analysis. The objective of this work is to use GNNs to predict the performance of 2D airfoils generated using the program XFOIL. The predicted performance parameters include bulk quantities such as coefficients of lift (C_L), drag (C_d, C_{dp}), moment (C_m), and node-specific quantities such as coefficient of pressure (C_p). In this report, a spline convolutional graph-based neural network is compared with deep learning neural networks to predict both bulk and node-specific quantities. The findings indicate the GNNs are able to predict bulk quantities quite well; however, when the number of outputs is increased, the deep neural network (DNN) proves to be better in its prediction capability. Two different normalization strategies were compared in the training of both GNNs and DNNs: minmax and standard deviation. In both types of networks, standard deviation scaling proved to be the best.

1.0 Introduction

Past researchers have used machine learning techniques such as deep neural networks (DNNs) and convolutional neural networks (CNNs) to estimate the performance of airfoils. Research that has utilized DNNs includes, but is not limited to, work by Papila et al. (Ref. 1), who used DNNs to optimize a supersonic turbine, given a set of design inputs; Madavan, Rai, and Huber (Ref. 2), who trained their DNN on airfoil design parameters such as y-values and metal angles to improve performance of supersonic turbine blades; and Rai and Madavan (Ref. 3), who revised their own design strategy and also proposed a new method with DNNs to optimize turbine airfoils.

Research that involved training a CNN to estimate airfoil performance includes the work of Yilmaz and German (Ref. 4), who trained a CNN on sample points in the x,y-space representing an airfoil shape. The coordinates were translated into an image and paired with a CNN. They noted that the next steps to improve feature prediction would be to train with an actual image of an airfoil. Zhang, Sung, and Mavris

(Ref. 5) used CNNs to predict the coefficient of lift of an airfoil, and Chen et al. (Ref. 6) used a similar method to predict the coefficients of lift, moment, and drag. Thuerey et al. (Ref. 7) used sample images of airfoils and simulation results as images with a CNN to predict an image depicting the performance of the airfoil. Their model used a multilayer perceptron (MLP) consisting of stacked CNN layers. This enabled the network to learn more detailed features of the design. Their MLP network was then paired with fully connected linear layers to predict a performance value.

The disadvantage of CNNs is that there are a lot of empty pixels in an image where the airfoil is not defined. The pixels represent an approximation of where the geometry is defined; therefore, the image resolution has an impact on the size of the neural network and the accuracy. In turbomachinery, meshes are more complicated, as there are blades, the hub, the shroud, tip clearances, cooling holes, and so forth. A two-dimensional (2D) representation of a geometry using an image provides a pixel approximation of the location of each point. For a 3D geometry, a 3D CNN would be needed to volumetrically capture the geometry; however, this requires a large amount of memory and storage. Why not use the same mesh for training that has already been created for analysis? This is where a graph neural network (GNN) would be useful.

Graph-based neural networks are relatively new. They were invented (Refs. 8 and 9) when researchers wanted to address acyclic graphs, which comprise nodes connected to nodes without forming a closed loop. Examples of GNNs include a tree, a social network, molecules, or pattern recognition such as purchasing history on Amazon.com. However, they can be used for another type of graph network: cyclic, which are either directed or undirected types of networks. An example of this would be a mesh or connected points defining a geometry. The way a GNN works is by performing a convolution on the nodes. Convolution on the nodes is achieved using "message passing." Message passing aggregates information from the neighboring nodes and updates the existing node with it. There are many different methodologies of how the messages are arranged, because with graph networks, each node contains features, the edges (ϵ) can contain weights, and the nodes or vertex (ν) can have other properties such as spatial location (e.g., x, y, z). Examples of graph networks range from molecular chemistry to classifying geometries using points from light detection and ranging (LiDAR); additional examples and code can be found on the PyTorch Geometric GitHub page (Ref. 10).

The design of aerospace airfoils often involves performing numerous numerical simulations using computational fluid dynamics (CFD) or finite element analysis (FEA) to design a single component. When paired with an optimization strategy, these methods produce large datasets that are rarely used after a design has been finalized. These computational datasets contain grids, unstructured or structured, where geometries are represented by nodes connected to other nodes through edges. This type of gridlike data structure is a graph and can be modeled using GNNs. There are several graph networks that are suited for geometry applications: PointCNN (Ref. 11), PointNET and PointNET++ (Refs. 12 and 13), and Dynamic Graph (Ref. 14). All of these methods are used primarily to classify the geometry of point clouds. An example would be detecting a chair and then locating what points belong to the legs of a chair. These methods are used to interpret the data from LiDAR in self-driving cars. Spline-based Convolutional Neural Network (SplineConv, Ref. 15), however, was designed to classify meshes. This report presents examples of how to use SplineConv to predict the performance of an airfoil. Performance characteristics include coefficient of lift (C_L), coefficient of drag (C_d), drag coefficient at zero lift (C_{dp}), and coefficient of moment (C_m). A lot of these applications rely on open datasets designed for classification. For aerospace applications, classifications may not be as important as regression in predicting performance. This report explores the use of SplineConv to learn geometrical features of airfoils, which are then paired to an MLP and used to predict the performance of "bulk" quantities such as C_L, C_d, C_{dp}, and C_m, and also node-specific quantities such as coefficient of pressure (C_p) values. The prediction accuracy of SplineConv is compared to the accuracy when using only MLP to predict all the quantities.

The nomenclature for this report is presented in the appendix to aid the reader.

2.0 Methodology

2.1 Overview

Two main steps were taken to train both the SplineConv and DNN, as shown in Figure 1. The first step, Data Generation, was to structure the data in a general way that can be used in both the MLP and the GNN. This was done by downloading a dataset of airfoils along with test conditions from an online resource (Ref. 16). Then the data was normalized: some airfoils have a different number of coordinate points (x,y) than others, and also a different number of C_p values along the chord, which is why each of the designs needed to be interpolated to have a consistent number of coordinate points and reevaluated using XFOIL (Ref. 17). The processed files were saved in a JSON format (Ref. 18). The next step was to modify the JSON data for both an MLP and GNN. PyTorch Geometric (Ref. 10) was used as the development framework for GNNs.

Airfoil Tools (Ref. 16) is a website containing over 900 different airfoil geometries along with performance data across a wide range of values for Reynolds number (Re) and angle of attack (α) and for two different free-stream turbulence levels (N_{crit}) of 5 and 9. However, Airfoil Tools does not contain data for C_p. This instead comes from XFOIL (Ref. 17) simulations. XFOIL was used to evaluate each airfoil at every condition—a combination of Re, α, and N_{crit}. The C_p value of each successful evaluation was saved to a JSON file. It is important to use JSON format, which uses ASCII text to save the data rather than binary; sometimes using different versions of Python or PyTorch can change the way binary files are saved, rendering the data useless. Convergence was checked by investigating an XFOIL output file containing the performance. If the simulation does not converge then XFOIL will not output any C_L or C_d values. Only the results that had converged were added to the dataset. The dataset and types are shown in Table I.

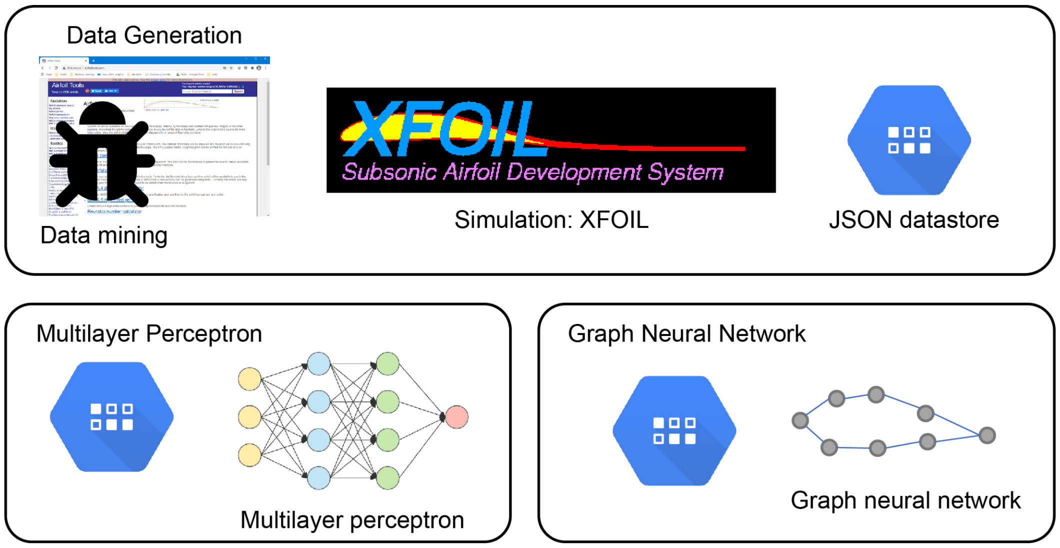

Figure 1.—Overview of evaluation methodology: data generation.

TABLE I.—NEURAL NETWORK INPUT AND OUTPUT DATA FOR AIRFOIL

Name	Inputs				Outputs[a]				
	Coordinates	Reynolds number, Re	Angle of attack, α	Turbulence level, N_{crit}	Lift, C_{L}	Drag, C_{d}	Moment, C_{m}	Drag at zero lift, C_{dp}	Pressure, C_{p}
String	Float array (198)	Float	Float	Float	Float	Float	Float	Float	Float array (198)

[a]Outputs are the coefficients of the listed properties.

The generalized dataset was then converted into an object structure to be used by PyTorch Geometric. For example, an airfoil would be represented as a single object with a set of properties: a vector that defines node features, an edge connectivity matrix detailing how each node is connected to the other, an edge feature array, and the (x,y) position of each node. The node features are the test conditions; if the same airfoil is tested at multiple conditions, then there are multiple copies of the data in the dataset. The data structure of GNNs is vastly different from that of the CNNs used for image recognition. CNNs use matrices that represent an image's red, green, and blue values. Setting up the dataset for MLP, though, requires the data for each airfoil to be in a vectorized format, which was much simpler to do. The output or labeled data in all the cases was a vector describing the intended values of C_{L}, C_{d}, C_{dp}, and C_{m} of each airfoil for each test condition. The objective was to compare the performance of GNNs with MLP in predicting the performance of airfoils.

2.2 The Multilayer Perceptron Overview

The MLP is a subset of DNNs. This neural network was created using PyTorch (Ref. 19). This network is made up of fully connected linear layers[1] shown in Figure 2. The inputs (n) were passed as a vector, comprising the height of each of the points along the outer airfoil surface (y_i) concatenated with Re, α, and N_{crit}. The number of inputs into the network was fixed. If a new airfoil were to be used, the number of points would have to be interpolated to match the network.

The hidden layer contains a number m of stacked linear layers to create fully connected hidden layers. A ReLU6 activation layer (Ref. 21) was used to initialize the layers to make the network differentiable and prevent vanishing gradients. Including a dropout layer can help reduce overfitting. Dropouts force these weights to deactivate at random, forcing other nodes to take on more responsibility (Ref. 22). This can result in more iterations to converge. The original paper on dropouts (Ref. 22) suggests that the max-norm regularization of the linear layers should be at most three to help prevent weights from approaching infinity and to improve training results. In this report, dropouts were not used between the layers because it would require hyperparameter tuning to achieve the best prediction. The MLP can be tuned relatively quickly, but the GNN would take months. The authors' code in GitHub allows the potential for dropouts to be used for future research.

[1]When using another extremely popular machine learning framework, Keras (Ref. 20), the "linear" layers are called "Dense."

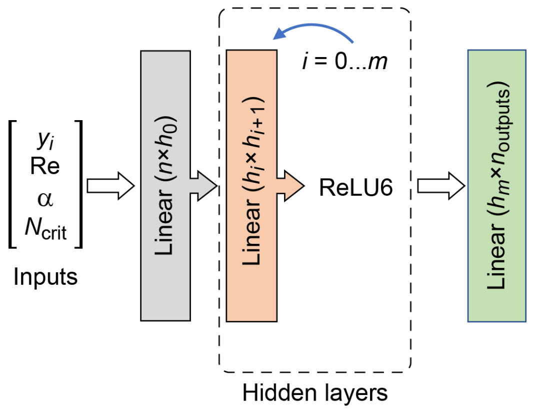

Figure 2.—MLP network setup. Inputs for the m layers are point height y, Reynold number Re, angle of attack α, and turbulence level N_{crit}; n is number of inputs, h are dimensions of hidden layers, and ReLU6 is activation layer.

2.3 Spline-Based Convolutional Neural Network

A GNN is a type of neural network that uses the connectivity of nodes or vertices to pass information. An image can be viewed as a type of GNN with each pixel as a node connected to their neighbors. The pixels' red, green, and blue values range from 0 to 255 and would be considered node features. Social networks and point clouds are also graph networks. What makes a graph different from an image is that the number of connections or pixels does not limit the learning; this is true for classification but not regression. Additionally, nodes can have multiple connections. The connections can also have weights (edge attributes) that, depending on the network, influence the message that is passed between nodes through their connected edges—a process known as message passing (Ref. 22). GNNs are more suitable for 2D and 3D shapes than for images or volumetric space of pixels—voxels—because a graph network only captures what is needed, using nodes and edges; depending on the message-passing algorithm, it can also factor in the x,y-coordinates of the nodes.

There are various methodologies for solving GNNs: kernel, spatial, and spectral. The latter involves the calculation of eigenvalues. However, for this application the spatial method of SplineConv, (Ref. 15) is used because of its ability to use the coordinates to extract information of the node-to-node relationships.

2.3.1 Message Passing

The fundamental idea behind graph networks is message passing (Ref. 23). Message passing relates to how information travels from one node to another through connected edges. A function can be applied to messages from neighboring nodes that are aggregated through a "mean," "min," or "max," which is used to update the current node state. A separate function can be used to update the previous node's information. Figure 3 outlines the message-passing process for SplineConv.

Figure 3 displays the process of a GNN. Airfoil geometries can be represented by mathematical equations, which makes them ideal for SplineConv. SplineConv is a kernel-based GNN that is used for feature recognition on structured and unstructured graphs or meshes. The flowchart on the right in Figure 3 shows the message passing used to update the node state; everything to the left is the creation of the message. On the left are the nodes P, Q, R, and S, represented by the coordinates x and y, and the flow conditions α, Re, and N_{crit}. The way neighboring nodes Q, P, and S influence node R is through applying a spline-basis (N) kernel operator across a vector of positions (u) of each node (i) and dimension (d). B_p is the product of the basis of each node. The index p represents the number of spline bases P. A spline basis can consist of d number of points defining it.

$$B_p(u) = \prod_{i=1}^{d} N_{i,p_i}(u_i) \tag{1}$$

After computing the basis, the data is then passed into a kernel function ($g_l(u)$) where the weights ($w_{p,l}$) of the networks are trained. The index (l) represents each feature; for example, α, Re, and N_{crit}. The sum of the weights multiplied by B_p represent each spline basis function computed at coordinates u. The results are summed for up to P B-spline bases. On a high level, the kernel function ($g_l(u)$) creates the message "g" (Figure 3) that is later aggregated by either "sum," "mean" (default) or "max." To update the future node state, the SplineConv uses a spline weighting function "γ" in conjunction with the aggregated message information.

$$g_l(u) = \sum_{p \in P} w_{p,l} \cdot B_p(u) \tag{2}$$

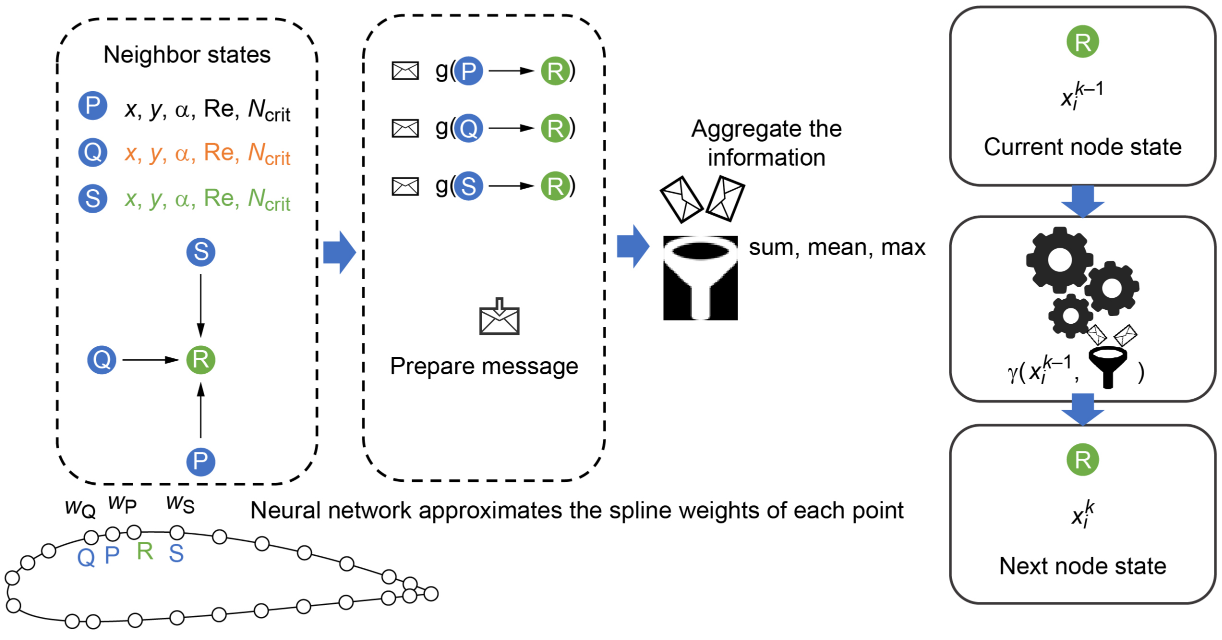

Figure 3.—Message passing with SplineConv model.

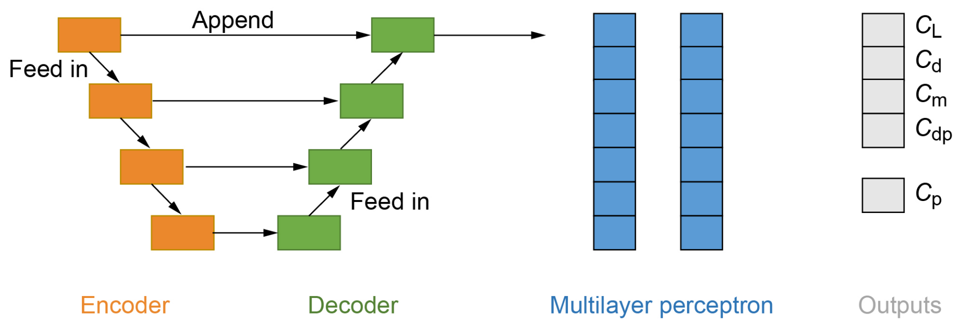

Figure 4.—Graphic neural network architecture: SplineConv is the encoder and decoder. Information is then passed into MLP, and a linear layer used to estimate the outputs.

2.3.2 Architecture and Parameterization

The GNN model is an encoder-decoder type paired with an MLP network, as illustrated in Figure 4. Each of the four encoders (orange) and four decoders (green) are the SplineConv model (Ref. 15). The inputs to this model are the node features (3), edge attributes (198), and position (198). The output of each of the encoders expands the number of features recognized from 3 to 16, for example. The decoder decreases the number of features. The line connecting the encoder to decoder represents an appending of the data to the data that are fed in. The number of encoder-decoder layers was varied in the SplineCNN along with neurons and hidden layers in the MLP. In many encoder-decoder design patterns, a BatchNorm (Ref. 24) is generally used. BatchNorm has been known to improve accuracy in image classification tasks; however, for this application, it was not necessary since the data were already normalized.

2.4 Data Normalization

Normalization is required in machine learning; it keeps the data at the same scale and reduces the possibility of large gradients in the model. Two different normalization strategies were compared for both MLP and GNNs: minmax and standard normalization. Minmax refers to the dataset being scaled between 0 and 1. This means the inputs α, Re, and N_{crit} and the outputs C_{L}, C_{d}, C_{dp}, C_{m}, and C_{p} at each x-coordinate value were scaled between 0 (*min*) and 1 (*max*). In Equations (3) to (5) x describes an arbitrary quantity; for example, C_{L}, C_{d}, C_{m}, C_{dp}, y, C_{p}, and so forth.

$$x_{\text{scaled}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{3}$$

The second normalization strategy is the standard scaler. The standard scaler (Eq. (4)) computes the mean and standard deviation (*stdev*) of each of the inputs and outputs and normalizes them by converting the data to a standard score x_{std}.

$$x_{\text{std}} = \frac{x - mean}{stdev} \tag{4}$$

$$x_{\text{scaled}} = x_{\text{std}}\left(max - min\right) + min \tag{5}$$

Both normalization methods were used in the comparison of the neural network models.

2.5 Loss Function

The loss function used by both MLP and the GNN was the mean-squared error (MSE) function, Equation (6), where n is the length of the input array and Y and $\hat{Y}$ are the predicted and actual outputs, respectively. This function was minimized during training of the neural network to reduce the average overall error.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{6}$$

2.6 Model Comparison Matrix

Table II and Table III contain the model configurations that were trained and analyzed. Each of the four models for both network types were compared when predicting the four outputs (C_L, C_d, C_{dp}, and C_m) and when predicting these four outputs in addition to C_p at every x-coordinate, for a total of 202 outputs.

3.0 Results

The results compare the MLP with the GNN for a variety of network configurations. The data are organized to show the training error following the network error for randomly selected airfoils from the test dataset. Models were trained to predict four quantities of lift (C_L), drag (C_d and C_{dp}), and moment (C_m). The next set of results show the results of training the models to predict the same four quantities along with the coefficient of pressure (C_p) at each point.

TABLE II.—GNN (SplineConv[a]) ENCODER-DECODER CONFIGURATIONS

Model	Normalization	Input features	Encoder features	Decoder features	MLP layers
minmax-GNN	minmax	3	16, 32, 64	64, 32, 16	None
minmax-GNN-MLP-64×4	minmax	3	16, 32, 64	64, 32, 16	64, 64, 64, 64
standard-GNN	standard	3	16, 32, 64	64, 32, 16	None
standard-GNN-MLP-64×4	standard	3	16, 32, 64	64, 32, 16	64, 64, 64, 64

[a](Ref. 15).

TABLE III.—MLP NETWORK CONFIGURATIONS

Model	Normalization	MLP layers
minmax-MLP-64×4	minmax	64, 64, 64, 64
minmax-MLP-128×4	minmax	128, 128, 128, 128
standard-MLP-64×4	standard	64, 64, 64, 64
standard-MLP-128×4	standard	128, 128, 128, 128

3.1 Train and Test Results: Multilayer Perceptron

Figure 5 shows the training results of the four models in Table III. These models were tasked with predicting only C_L, C_d, C_{dp}, and C_m. The learning rate used was a step decay starting at 10^{-2} and decaying every 10 epochs to 10^{-5}. An epoch is measured as a single cycle through the data. Because the dataset is large and to reduce training time, 20 percent of the dataset was selected at random for training. The same 20 percent was also used to train the GNN. From the training results, the best models were the "minmax-MLP-64×4" followed by "standard-MLP-64×4" and "standard-MLP-128×4."

Figure 6 contains the training results for the MLP models when predicting the C_p along with the existing four parameters. The results show a much worse prediction since the model was trying to reduce the average loss for all 198 outputs of C_p along with C_L, C_d, C_{dp}, and C_m. The best model was the minmax-MLP-64×4.

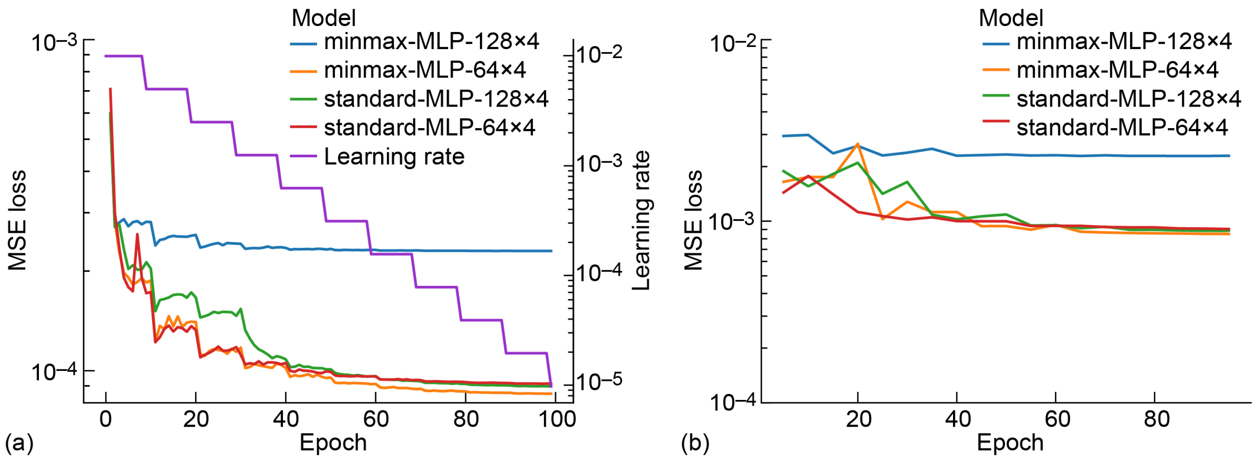

Figure 5.—MLP airfoil modeling results for predicting coefficients of lift (C_L), drag (C_d), drag at zero lift (C_{dp}), and moment (C_m). (a) Training. (b) Testing.

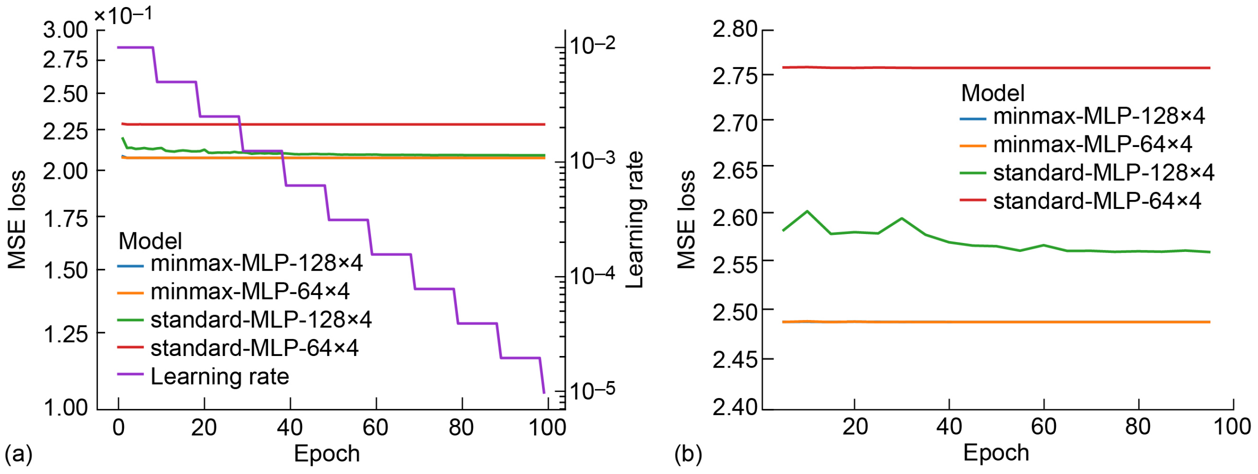

Figure 6.—MLP airfoil modeling results for predicting coefficients of lift (C_L), drag (C_d), drag at zero lift (C_{dp}), moment (C_m), and pressure (C_p). (a) Training. (b) Testing.

3.2 Train and Test Results: Graph Neural Network

The GNN took 1 day to train a model compared with 1 to 2 hours per model for MLP. This was due to the structure of the data being a list of objects. According to the training and test results in Figure 7, the best models were the minmax-GNN-MLP with (64×4) and without pairing with an MLP (no hidden layers). The learning rate was stepped down from 10^{-2} to 10^{-5} within 20 epochs; 20 was chosen based on the training time and the convergence of loss.

Figure 8 demonstrates that when training a model to predict the C_p along with the other four output parameters, none of the GNN models were able to match the accuracy of the MLP alone.

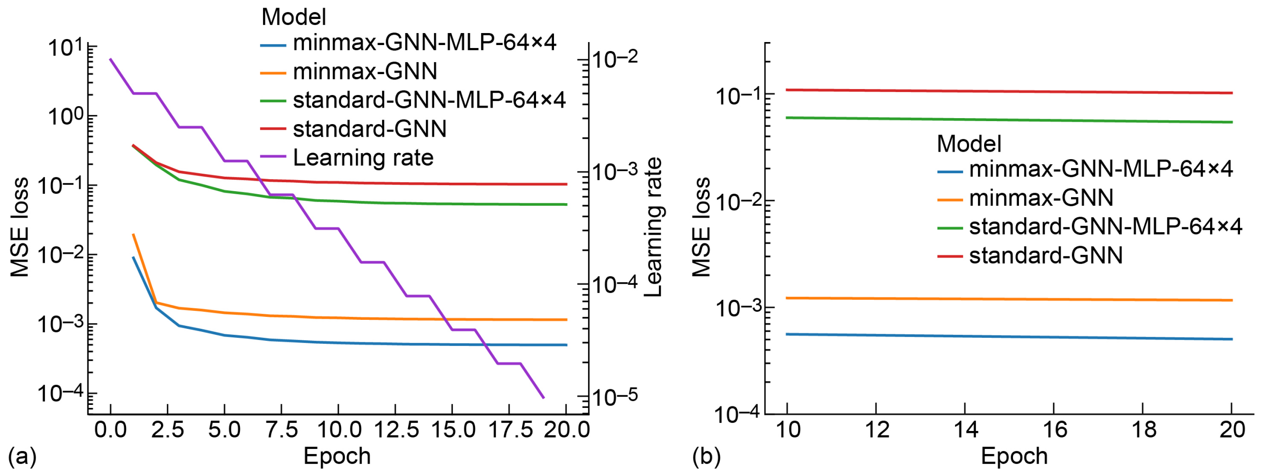

Figure 7.—GNN airfoil training results for predicting coefficients of lift (C_L), drag (C_d), drag at zero lift (C_{dp}), and moment (C_m). (a) Training. (b) Testing.

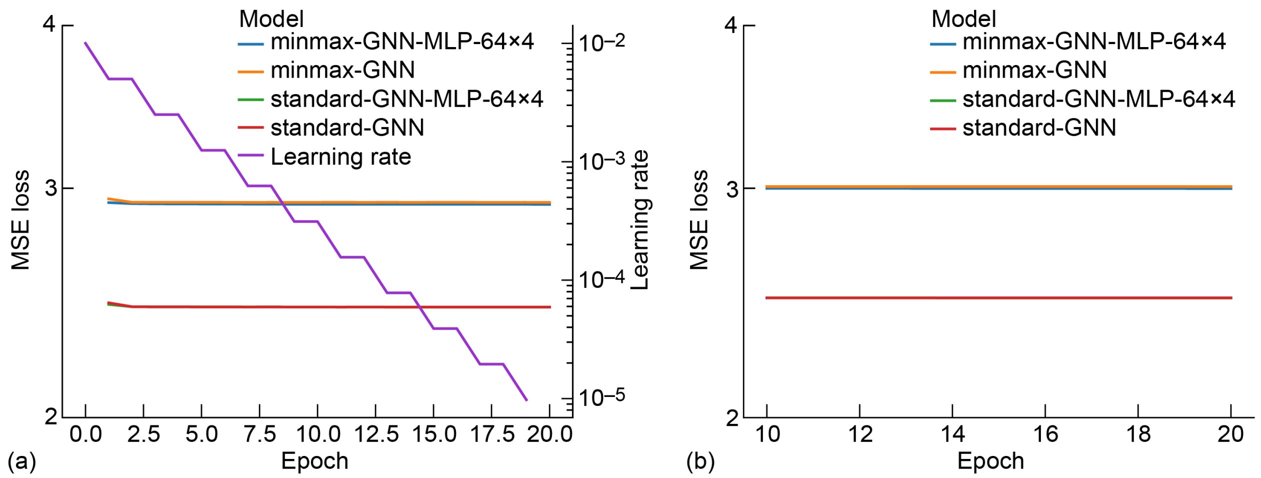

Figure 8.—GNN training results for predicting airfoil coefficients of lift (C_L), drag (C_d), drag at zero lift (C_{dp}), moment (C_m), and pressure (C_p).

3.3 Comparing Models Using an Arbitrary Airfoil

Training results are only part of the analysis. It is more valuable to investigate how well the models predict the performance of data outside of their training dataset. Figure 9 compares each neural network on an R149 airfoil. The GNN model standard-GNN-64x4 was able to predict performance close to the values from XFOIL, and GNNs proved to be better than the MLP at capturing the trend in C_d and C_{dp}. This is contrary to the errors observed in the training data.

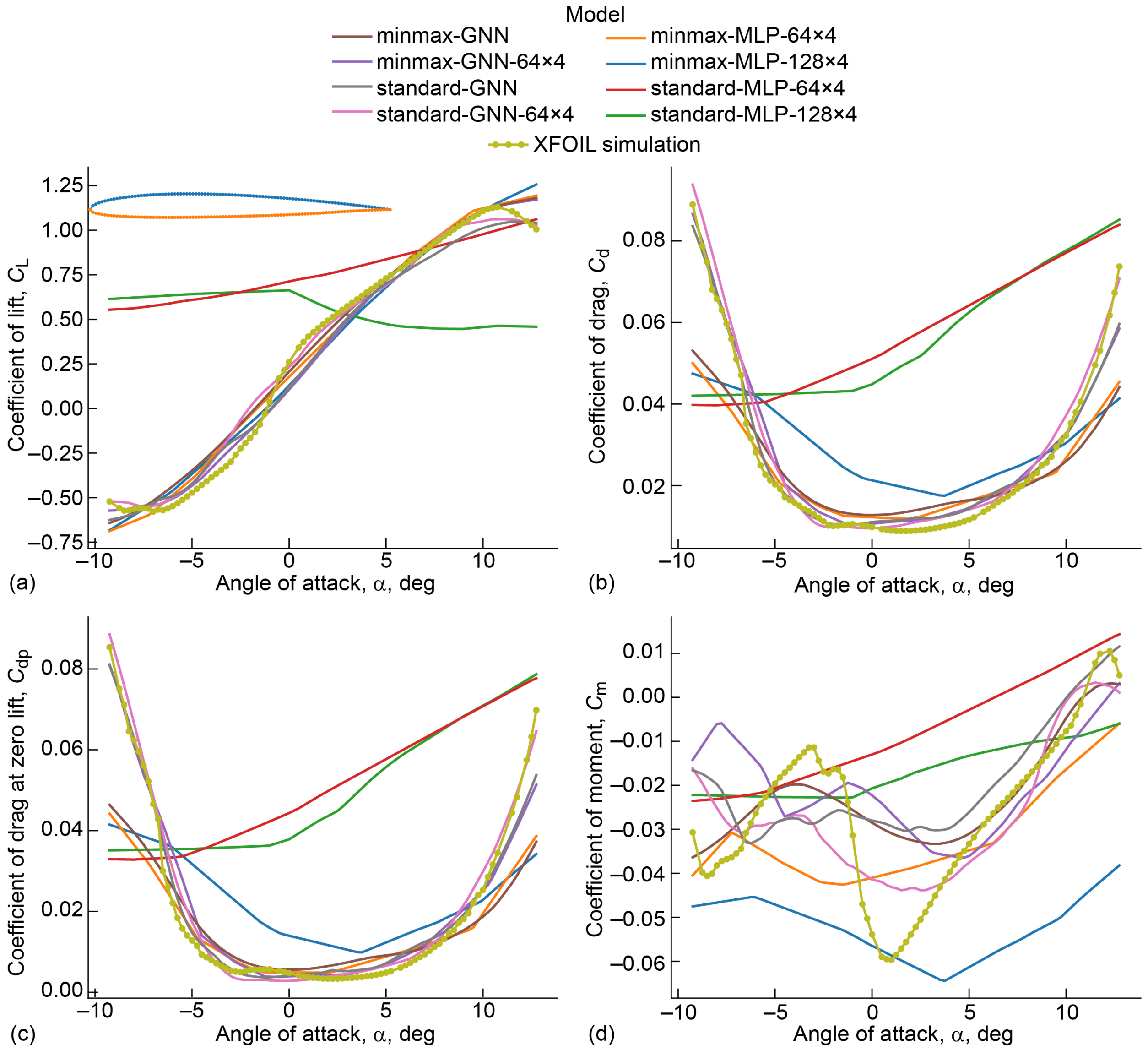

Figure 9.—Predictions of R149 airfoil performance from MLP models with and without including GNN. XFOIL simulation results included for comparison. Reynolds number Re = 2×10^5 and turbulence level N_{crit} = 9. (a) Coefficient of lift, C_L. (b) Coefficient of drag, C_d. (c) Coefficient of drag at zero lift, C_{dp}. (d) Coefficient of moment, C_m.

When investigating a different airfoil (the R149), pairing the GNNs with an MLP was better at predicting performance than using MLP alone. The best model was found to be standard-GNN-64×4. In the example of the E852 airfoil shown in Figure 10, the coefficient of moment (C_m) was difficult to predict; however, the same GNN model with standard deviation proved to be the best architecture and normalization strategy.

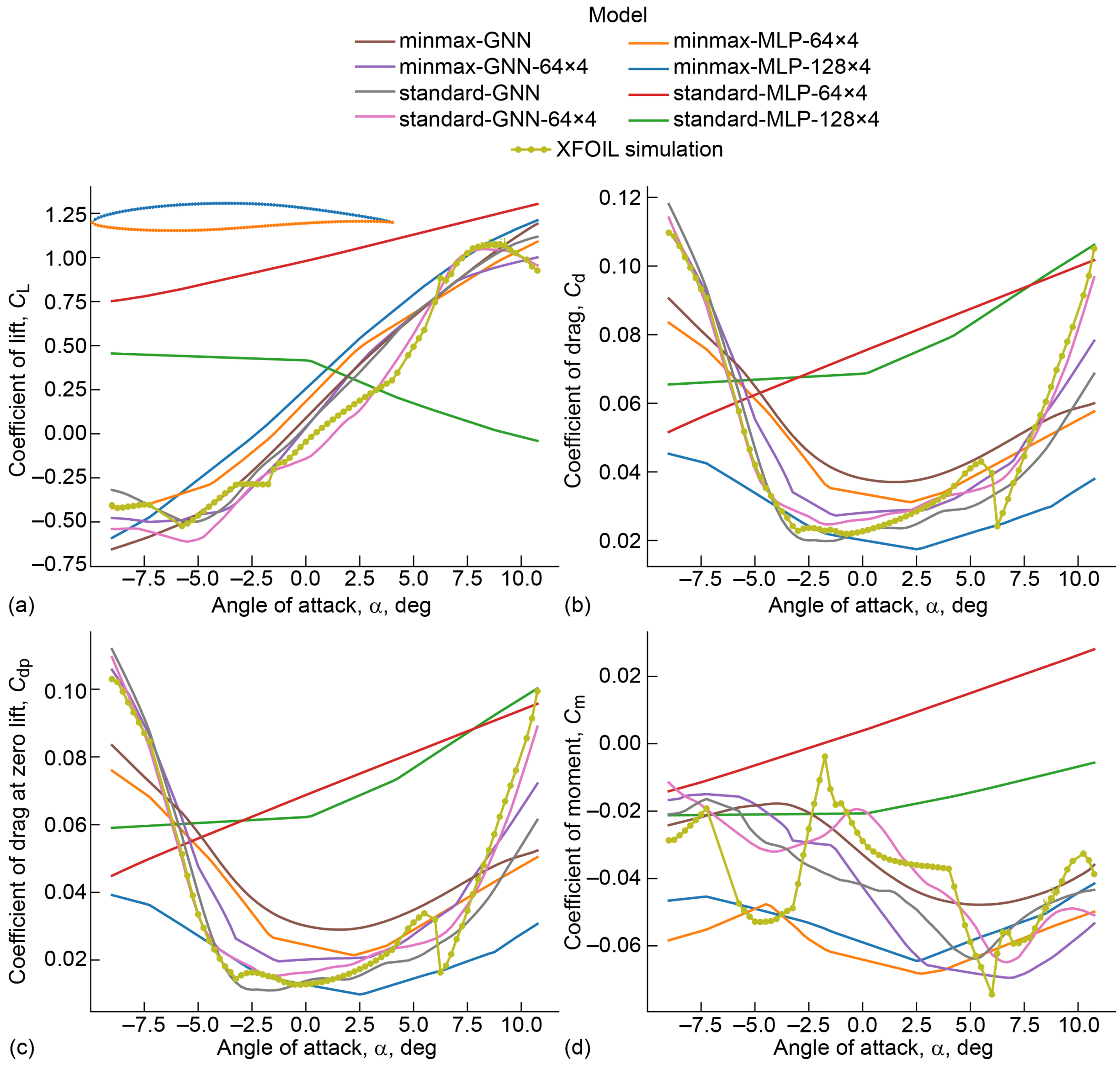

Figure 10.—Predictions of E852 airfoil performance from MLP models with and without including GNN. XFOIL simulation results included for comparison. Reynolds number Re = 2×10^5 and turbulence level N_{crit} = 9. (a) Coefficient of lift, C_L. (b) Coefficient of drag, C_d. (c) Coefficient of drag at zero lift, C_{dp}. (d) Coefficient of moment, C_m.

Predicting C_p required increasing the number of predicted outputs from 4 to 202, and because of this change, the accuracy was severely affected. Figure 11 shows the C_p prediction plot for the R149 airfoil. MLP struggled to predict C_p. Lines with sharp fluctuations are the predictions for suction and pressure side colliding. Unfortunately, when predicting C_p using the same network architecture, the GNNs were not able to provide an accurate prediction. The authors recommend limiting the number of predicted values to around four for the most accurate predictions. Increasing the number significantly reduces the accuracy of all predictions.

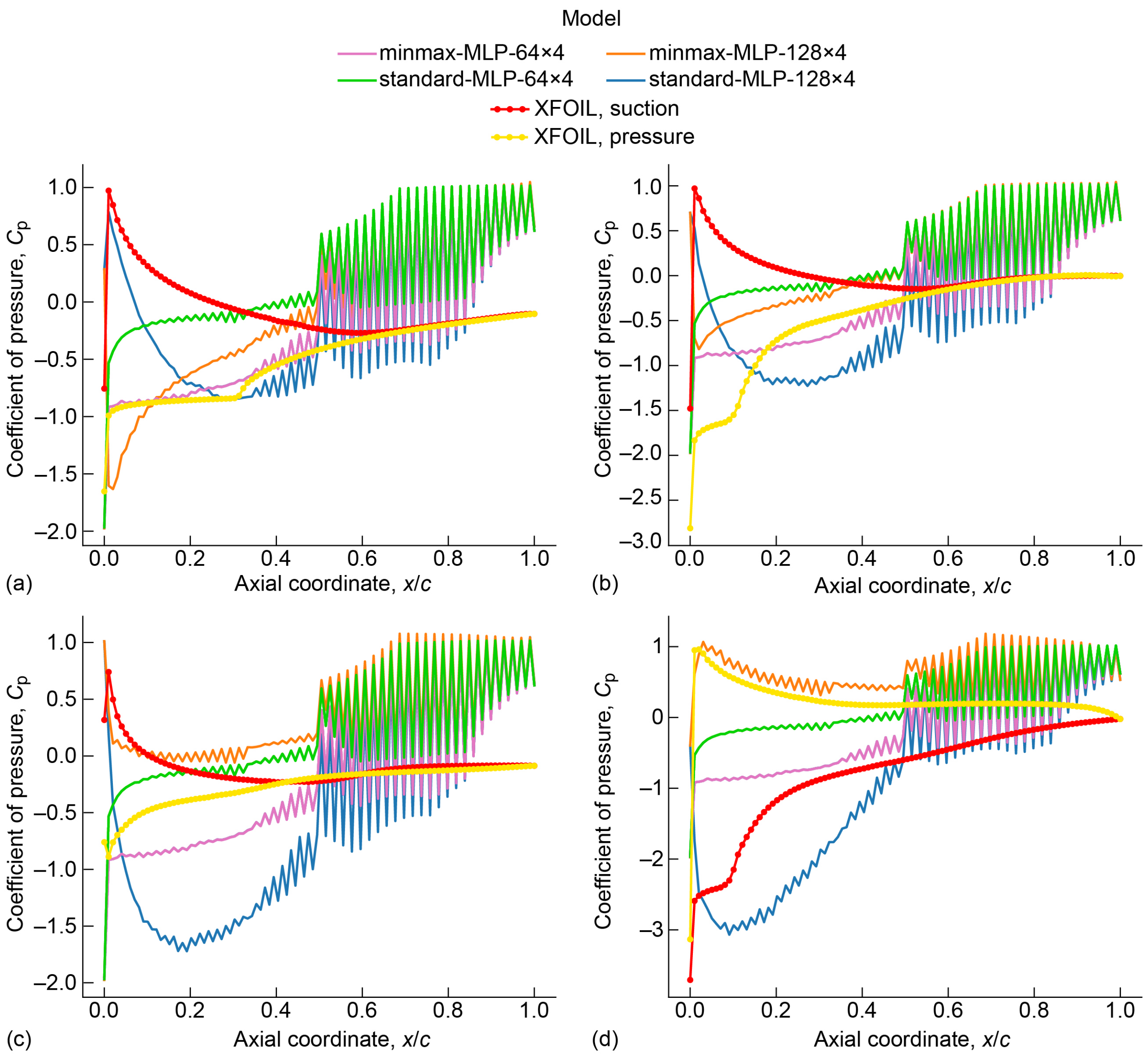

Figure 11.—Predictions of coefficient of pressure (C_p) for R149 airfoil MLP models, over axial coordinate (x) normalized by chord length (c). XFOIL simulation results included for comparison. XFOIL red line indicates suction side, and yellow line indicates pressure side. Reynolds number Re = 2×10^5 and turbulence level N_{crit} = 9. (a) Angle of attack α = –6.25°. (b) α = –2.5°. (c) α = –1.75°. (d) α = 0.75°.

4.0 Concluding Remarks

The result of the training suggests that the multilayer perceptron (MLP) network is able to reach a lower mean-squared error than the graph neural network (GNN); however, in practice, GNNs proved to be more beneficial in minimizing prediction error. One downside of using a GNN is the long training time due to the data structure being stored in object form. Despite this complexity, GNNs offer a new way of capturing the shape of the geometry of interest and could possibly be used with unstructured meshes.

There are many different variations of GNNs that may offer better predictions. The authors have made the code and dataset available on GitHub and have also written tutorials that can be used to reproduce all the plots and results presented in this report. Moreover, the authors would like to invite other researchers to use the dataset on GitHub to investigate different network architectures. We hope that the community can leverage this work to develop aerospace- and aeronautic-specific message-passing algorithms.

References

1. Papila, Nilay, et al.: Shape Optimization of Supersonic Turbines Using Response Surface and Neural Network Methods. AIAA 2001–1065, 2001.
2. Madavan, Nateri K.; Rai, Man Mohan; and Huber, Frank W.: Neural Net-Based Redesign of a Gas Generator Turbine for Improved Unsteady Aerodynamic Performance. AIAA 99–2522, 1999.
3. Rai, Man Mohan; and Madavan, Nateri K.: Application of Artificial Neural Networks to the Design of Turbomachinery Airfoils. J. Propuls. Power, vol. 17, no. 1, 2001, pp. 176–183.
4. Yilmaz, Emre; and German, Brian J.: A Convolutional Neural Network Approach to Training Predictors for Airfoil Performance. AIAA 2017–3660, 2017.
5. Zhang, Yao; Sung, Woong Je; and Mavris, Dimitri N.: Application of Convolutional Neural Network to Predict Airfoil Lift Coefficient. AIAA 2018–1903, 2018.
6. Chen, Hai, et al.: Multiple Aerodynamic Coefficient Prediction of Airfoils Using a Convolutional Neural Network. Symmetry, vol. 12, no. 4, 2020, p. 544.
7. Thuerey, Nils, et al.: Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows. AIAA J., vol. 58, no. 1, 2020, pp. 25–36.
8. Gori, Marco; Monfardini, Gabriele; and Scarselli, Franco: A New Model for Learning in Graph Domains. Proceedings of International Joint Conference on Neural Networks, Montreal, 2005, pp. 729–734.
9. Scarselli, Franco, et al.: The Graph Neural Network Model. IEEE Trans. Neural Netw., vol. 20, no. 1, 2009, pp. 61–80.
10. Fey, Matthias; and Lenssen, Jan Eric: PyTorch Geometric. 2019. https://github.com/pyg-team/pytorch_geometric Accessed July 26, 2022.
11. Li, Y., et al.: PointCNN: Convolution On *X*-Transformed Points. 2018. https://arxiv.org/abs/1801.07791 Accessed July 26, 2022.
12. Qi, Charles R., et al.: PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. 2016. https://arxiv.org/abs/1612.00593 Accessed July 26, 2022.
13. Qi, Charles R., et al.: PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. 2017. https://arxiv.org/abs/1706.02413 Accessed July 26, 2022.
14. Wang, Yue, et al.: Dynamic Graph CNN for Learning on Point Clouds. ACM Trans. Graph., vol. 38, no. 5, 2019, pp. 1–12.
15. Fey, Matthias, et al.: SplineCNN: Fast Geometric Deep Learning With Continuous B-Spline Kernels. 2017. https://arxiv.org/abs/1711.08920 Accessed July 26, 2022.

16. Airfoil Tools. 2022. www.airfoiltools.com Accessed Nov. 20, 2020.
17. Drela, Mark: XFOIL 6.99. Subsonic Airfoil Development System, 2020. https://web.mit.edu/drela/Public/web/xfoil/ Accessed July 26, 2022.
18. JavaScript Object Notation: Introducing JSON. https://www.json.org/json-en.html Accessed July 26, 2022.
19. Paszke, Adam, et al.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32, NeurIPS Proceedings, 2019. https://papers.nips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html Accessed July 26, 2022.
20. Chollet, F., et al.: Keras. 2015. https://github.com/fchollet/keras Accessed Aug. 23, 2022.
21. Brownlee, Jason: Machine Learning Mastery: A Gentle Introduction to the Rectified Linear Unit (ReLU). 2019. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ Accessed July 26, 2022.
22. Srivastava, Nitish, et al.: Dropout: A Simple Way to Prevent Neural Networks From Overfitting. J. Mach. Learn. Res., vol. 15, 2014, pp. 1929–1958.
23. Gilmer, Justin, et al.: Neural Message Passing for Quantum Chemistry. Proceedings of the 34th International Conference on Machine Learning, vol. 70, 2017, pp. 1263–1272.
24. Ioffe, Sergey; and Szegedy, Christian: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning, vol. 37, 2015, pp. 448–456.

Additional Resource

Source code and dataset for this report:

Airfoil Learning. 2022. https://github.com/nasa/airfoil-learning Accessed July 27, 2022.

Appendix—Nomenclature

2D	two dimensional
B_p	product of basis of node; Equation (1)
C_d	coefficient of drag
C_dp	coefficient of drag at zero lift
C_L	coefficient of lift
C_m	coefficient of moment
C_p	coefficient of pressure
CFD	computational fluid dynamics
CNN	convolutional neural network
d	*s*pline basis dimensions; Equation (1)
DNN	deep neural network
FEA	finite element analysis
$\mathrm{g}_l(u)$	continuous convolution kernel functions; Equation (2)
GNN	graph neural network
h	size (number of neurons) of hidden layer; Figure 2
i	index representing each node
k	index representing history of node state; Figure 3
l	index representing α, Re, or N_crit
LiDAR	light detection and ranging
m	number of layers in MLP; Figure 2
MLP	multilayer perceptron (neural network)
MSE	mean-squared error (function); Equation (6)
N_crit	turbulence level
N_{i,p_i}	array of spline kernel operators applied to each point; Equation (1)
n	number of inputs being passed into machine learning network in Figure 2 length of loss function input array; Equation (6)
P	number of B-spline bases; Equation (2)
p	index representing spline bases P
Re	Reynolds number
ReLU6	activation layer in MLP
stdev	standard deviation
u	coordinates (x,y) that define geometry; Equations (1) and (2)
$w_{p,l}$	weights tuned during training process by an optimization function; Equation (2)
x	axial (horizontal) coordinate arbitrary quantity; Equations (3) to (5)
Y	predicted output of loss function; Equation (6)
$\hat{Y}$	actual output of loss function; Equation (6)
y	vertical coordinate
α	angle of attack
γ	SplineConv spline weighting function (Figure 3)
ϵ	graph edge that connects nodes to nodes
ν	vertex: coordinates that define position of node; similar to u

Subscripts of *x*

min	minimum value of an array
max	maximum value of an array
scaled	scaled value from an array
std	standard deviation