AIAA Journal
Vol. 57, No. 3, March 2019

# Inverse Design of Airfoil Using a Deep Convolutional Neural Network

Vinothkumar Sekar,[*] Mengqi Zhang,[†] Chang Shu,[‡] and Boo Cheong Khoo[§]
*National University of Singapore, Singapore 117575, Republic of Singapore*

DOI: 10.2514/1.J057894

**This paper proposes an approach to perform the inverse design of airfoils using deep convolutional neural networks (CNNs). The conventional approaches are based on the solution of differential equations, which are either difficult to solve or take a tedious procedure to obtain the solution. As shown in this paper, the deep learning technique can be used effectively to obtain the airfoil shape from the coefficient of pressure distribution. More specifically, the CNN is applied due to its ability to handle any airfoil geometry without the need for complex parametrization. In the training phase, the pressure coefficient distribution is fed as input to the CNN to obtain a prediction model for the airfoil shape. In the testing phase, a new pressure coefficient distribution is given to the CNN model, generating an airfoil shape that is very close to the associated airfoil with an $L_2$ error of less than 2% for most of the cases. An extensive investigation of the effects of various hyperparameters in the CNN architectures is performed. The results show that the CNN method is very efficient in terms of computational time and shows a competitive prediction accuracy. It thus constitutes an attractive new method in the inverse design of airfoils, especially when existing data can be readily exploited.**

## Nomenclature

| | | |
|---|---|---|
| $a$ | = | neuron output |
| $b$ | = | bias term |
| $C_p$ | = | coefficient of pressure |
| $E$ | = | mean square error |
| $E_W$ | = | mean square error gradients |
| $I$ | = | input |
| $w$ | = | neuron weight |
| $y_p$ | = | predicted value |
| $y_t$ | = | true value |
| $\sigma$ | = | activation function |
| $*$ | = | convolution operation |

*Subscripts*

| | | |
|---|---|---|
| $i, j, k, m, n, c$ | = | indices |

*Superscript*

| | | |
|---|---|---|
| $l$ | = | layer |

## I. Introduction

INVERSE problems pose a great challenge in terms of numerical algorithms, computational time, and convergence. In the field of aerodynamics, the inverse design of airfoil shapes is of paramount importance for aerodynamic components such as aircraft wings, helicopter rotor blades, etc. Given the flow conditions, the shape determines the pressure distribution over the airfoil, which in turn determines the performance of the aerodynamic components. In a forward approach, the airfoil shape and flow conditions are specified; and Navier–Stokes (NS) equations are solved in order to obtain the pressure distribution. In an inverse approach, the pressure distribution is specified and the airfoil shape is obtained as an output of a complex optimization procedure, which is generally known as aerodynamic shape optimization [1]. The shape optimization approach is, in general, very time consuming because of its complex algorithm and the requirement of numerous forward calculations. The latter is due to the fact that, at every iteration of the optimization procedure, the airfoil shape is morphed, and hence the NS solutions need to be updated for the new morphed shape.

The shape optimization algorithms are broadly classified into gradient-free and gradient-based algorithms. An extensive review on both gradient-free and gradient-based aerodynamic shape optimization methods was given in Ref. [2]. Gradient-free methods can find the global solution of the optimization problem [3], but the algorithms are relatively complex and the convergence is very slow as compared to gradient-based algorithms. This is due to the fact that the gradient-free algorithms need to evaluate a large number of different shapes while performing optimization, and are hence prohibitively expensive while using a NS solver. Compared to gradient-free methods, gradient-based methods are simpler and require fewer iterations to find an optimal solution, and hence have a faster convergence; but, they might not be able to find the global solution [4]. Gradient-based methods require the calculation of gradients of the objective function either directly or using adjoint methods. Although gradient-based methods have been successfully applied for aerodynamic optimization, they require continuity of the design space. In the presence of discontinuity, the algorithm becomes less robust and results in poor convergence [5]. Hence, there is a need for a gradient-free, fast, robust, global, and accurate methodology to do aerodynamic inverse design.

In the recent years, there has been significant development in the field of machine learning, and the data-driven modeling approaches for physical systems are becoming increasingly popular. In the community of computational fluid dynamics, there is a dramatic increase in the collection of numerical simulation data. With the aid of machine learning, these existing scientific databases can be used for data-driven modeling, design, and optimization. As turbulence is a complex phenomenon [6,7] with rich features that evade Reynolds-averaged Navier–Stokes (RANS) models, recently, researchers in the field of turbulence modeling community extensively applied machine learning techniques to improve RANS models [8]. Among them, Ling et al. [9] were probably the first to apply a deep learning technique in RANS turbulence modeling. In addition, there have been some attempts to use machine learning for aerodynamic prediction, design, and optimization. Rai and Madavan [10] used multilayered neural networks [multilayer perceptron (MLP)] for the aerodynamic design of turbomachinery, and they found it efficient in achieving design targets. Kharal and Saleem [11] and Sun et al. [12]

Received 9 September 2018; accepted for publication 11 November 2018; published online 11 January 2019. Copyright © 2018 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved. All requests for copying and permission to reprint should be submitted to CCC at www.copyright.com; employ the ISSN 0001-1452 (print) or 1533-385X (online) to initiate your request. See also AIAA Rights and Permissions www.aiaa.org/randp.

[*]Ph.D. Student, Department of Mechanical Engineering, 9 Engineering Drive 1.

[†]Assistant Professor, Department of Mechanical Engineering, 9 Engineering Drive 1.

[‡]Professor, Department of Mechanical Engineering, 9 Engineering Drive 1; mpeshuc@nus.edu.sg (Corresponding Author).

[§]Professor, Department of Mechanical Engineering, 9 Engineering Drive 1.

used a MLP along with airfoil parametrization techniques in order to obtain the shapes of the airfoils from the target conditions with a relatively smaller airfoil database. However, these airfoil parametrization techniques to some extent lacked accuracy and required manual interference in parametrization [13].

All the aforementioned works were based on a MLP neural network. Compared to a MLP, the convolutional neural network (CNN) requires fewer trainable parameters and provides the flexibility to learn any complex geometrical shape directly without the requirement of parametrization. In addition, a deep CNN is able to extract high-dimensional features efficiently, and its performance scales with the database size. In many challenging machine learning tasks, such as image recognition [14], CNNs are rapidly replacing the MLP. In fluid mechanics, Guo et al. [15] applied the CNN to approximate the steady-state laminar flow, in which the CNN's ability to learn geometrical details was well demonstrated. Zhang et al. [16] used the CNN to predict the aerodynamic coefficients of airfoils at various Mach and Reynolds numbers, and they achieved good prediction accuracy. Yilmaz and German [17] applied the CNN to map airfoil shapes to pressure distribution and achieved a testing accuracy of about 80%. These recent works reflect the increased attention of applying deep CNN techniques in the field of fluid mechanics.

In this work, we propose to apply the deep CNN algorithm in the inverse design of the airfoil shape. The existing pressure coefficient distribution will be input to the deep CNN to train a prediction model for the airfoil shape. The model will then be tested using new pressure coefficient distribution data to see if it can yield an airfoil shape that is close to the correct one (associated with the pressure coefficient distribution). The paper is organized as follows. Section II provides the details on the multilayer perceptron and the convolutional neural network along with the data preparation methods for training. In Sec. III, the prediction results of the CNN are discussed along with the influence of CNN hyperparameters; the effect of the input data size on convergence and accuracy is analyzed. Concluding remarks are provided in Sec. IV.

## II. Methodology

Machine learning techniques are broadly classified into supervised and unsupervised techniques. The scope of the present work is limited to supervised machine learning techniques applied to regression problems, in which the algorithm is trained using labeled inputs to distill the underlying features in the data. Deep CNNs are the choice of algorithms for the present work. The details of the MLP and CNN are briefly explained in this section.

### A. Multilayer Perceptron

The multilayer perceptron [18], in general, is known as the vanilla neural network (or neural network); it is made up of several neurons, which are connected together in a complex manner to form a network. Neurons are the basic elements of the MLP, which perform nonlinear input–output mapping defined by the regression problem. As shown in Fig. 1, the neuron has $n$ inputs $x_1, x_2, \ldots, x_n$ and its corresponding weights $w_1, w_2, \ldots, w_n$. The weighted input

$$\sum_{i=1}^{n} w_i x_i + b$$

is called signal to the neuron $z$, where $b$ is a bias term. The output is obtained by passing the input signal through a nonlinear activation function $\sigma(\cdot)$. Therefore, the neuron output is written as follows:

$$a(x) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) = \sigma(z) \quad (1)$$

where $a(x)$ is called the activation output (i.e., the neuron output), and $z$ is the signal to the neuron. Consider a typical MLP neural network formed by a dense connection of several neurons as shown in Fig. 2, which consists of input, hidden, and output layers. Each layer

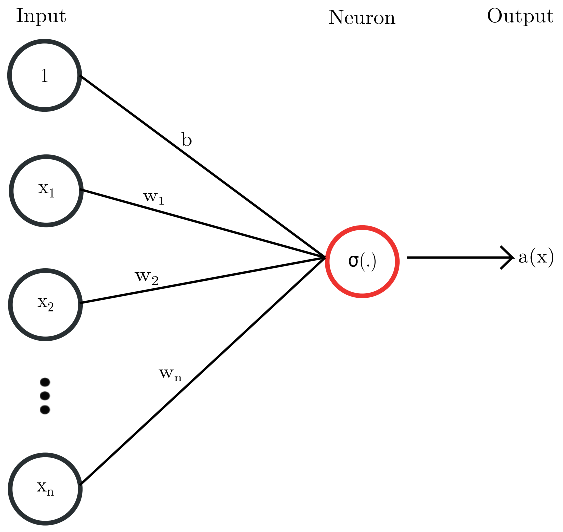

**Fig. 1** **Typical neuron [$x_1, x_2, \ldots, x_n$ indicate input, $w_1, w_2, \ldots, w_n$ indicate weights, $b$ indicates bias term, $\sigma$ indicates activation function, and $a(x)$ represents output].**

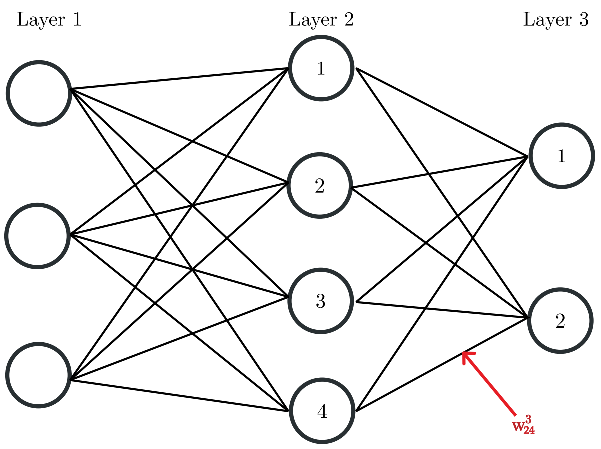

**Fig. 2** **Typical MLP neural network (weight $w_{24}^3$ indicates weight connection from fourth neuron in (3 – 1)th layer to second neuron in third layer).**

receives input from its previous layers, except the input layer, which receives the user-fed input. The output of a $j$th neuron in the $l$th layer $a_j^l$ is obtained as follows:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{(l-1)} + b_j^l\right) \quad (2)$$

where $w_{jk}^l$ denotes the weight of the connection from the $k$th neuron in the $(l-1)$th layer to the $j$th neuron in the $l$th layer, and $b_j^l$ indicates the bias term of the $j$th neuron in the $l$th layer. The process of obtaining the neuron activations is called feedforward. The weights and biases of the MLP are trained using a backpropagation algorithm [19]. The details of the backpropagation algorithm applied to the MLP were given by Nielsen [20].

### B. Convolutional Neural Network

The MLP has a wide application in several fields. However, there are some limitations of MLPs in the field, such as computer vision, in which the images are directly used as input. This is due to the fact that, for the imagelike data input, MLPs involve too many trainable parameters (weights) than CNN during the training phase, which makes MLPs inefficient in processing image data. Hence, in order to alleviate this problem, LeCun et al. [21] introduced the convolutional neural network with much fewer trainable parameters for application

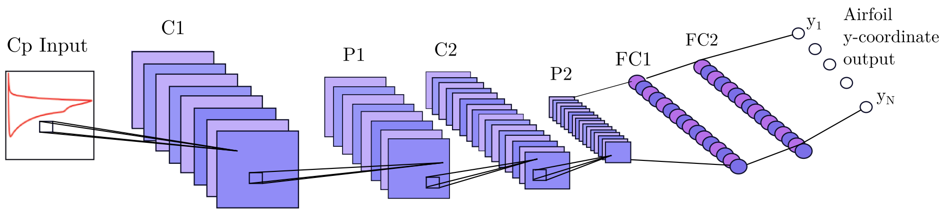

**Fig. 3 Typical convolutional neural network.**

in computer vision. CNNs comprise several types of layers, such as the convolutional layer, the pooling layer, and the fully connected layer. A typical schematic of the CNN architecture used in this study is shown in Fig. 3. Some of the important features of the CNN architecture are explained here. C1 and C2 indicate the convolved feature maps of the first and second convolution layers, P1 and P2 indicate the pooled feature maps of the first and second pooling layers, and FC1 and FC2 indicate the first and second fully connected layers.

### 1. *Convolutional Layer*

The convolutional layer plays an important role in CNN operation. This layer consists of learnable kernels (also known as filters), which are small in spatial dimensions. The weights of the kernels make an elementwise scalar product with the region connected to the input volume, known as the receptive field. As the kernel glides through the entire input, the scalar product is calculated as shown in Fig. 4. This operation is performed for one kernel. There will be a number of such kernels in each convolutional layer. As mentioned earlier, each kernel produces its own scalar product. The output of all the kernels is stacked in depth dimension, called activation maps. Then, the calculated activation maps are passed through a nonlinear activation unit, to be introduced in the following. The scalar dot product operation along with the nonlinear activation unit forms one convolutional layer. Depending on the architecture, many such convolutional layers can be added to the CNN. The convolution operation for one filter (also known as cross correlation in machine learning terminology) can be shown as follows:

$$(I * w)_{ij} = \sum_{m=0}^{l_1-1} \sum_{n=0}^{l_2-1} \sum_{c=1}^{C} w_{m,n,c} \cdot I_{i+m,j+n,c} \tag{3}$$

where $I$ is input image with length $L$, height $H$, and depth $C$; and $w$ indicates the filter with a size of $l_1 \times l_2$. The output of the convolution for one filter $a_{ij}^l$ can be expressed as follows:

$$a_{ij}^l = \sigma(w_{m,n,c}^l * a_{i+m,j+n,c}^{l-1}) + b \tag{4}$$

There will be a bank of filters used in every convolution layer. The spatial dimensionality of the output volume of the convolution layers can be altered using different sizes of the filters: the stride and the zero padding. For the nonlinear activation unit, the rectified linear unit (RELU) function is usually favored because of its ability to train the network much faster. The RELU function is defined as the positive part of its argument as $\sigma(x) = \max(0, x)$.

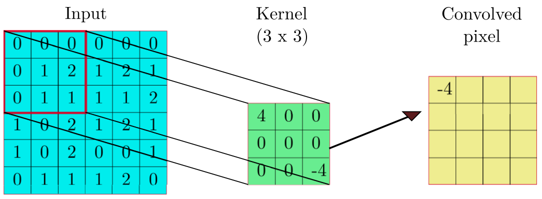

**Fig. 4 Typical convolution operation in the CNN.**

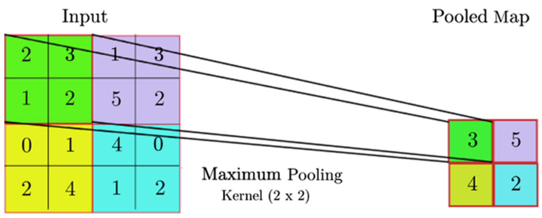

**Fig. 5 Typical pooling operation in the CNN.**

### 2. *Pooling and Fully Connected Layer*

The pooling layer generally follows a convolutional layer. The pooling layer performs spatial reduction of the dimension of the given input, which is known as downsampling. The pooling layer operates on the output of the convolutional layer, and it scales its dimensionality using a specified pooling operation. The pooling operations used are either maximum pooling or average pooling. For example, with a pooling kernel size of $2 \times 2$, the maximum pooling operation gives the maximum of four numbers, as shown in Fig. 5. Fully connected layers contain neurons that have full connections to the previous layers. This layer is exactly same as the MLP neural network. Depending on the requirement, there can be more than one fully connected layer used in the CNN architecture.

## C. Data Preparation

Data preparation is an important task in machine learning. In the present work, the $C_p$ distribution over the upper and lower surfaces of the airfoil is taken as input to the CNN and the airfoil shape is directly obtained as output. Ideally, the $C_p$ distribution of the airfoils needs to be generated by solving the Navier–Stokes equation, which is a prohibitively time-consuming task, and thus fewer training samples can be collected. Hence, we opt to use XFOIL as a simulation tool to obtain more $C_p$ distribution, which will help to render the CNN model to its maximal effectiveness. A dataset that consists of 1343 airfoils is used for this study. Currently, we perform this study for a fixed angle of attack and Reynolds number. XFOIL [22] is used to obtain the $C_p$ distribution of dataset airfoils for a Reynolds number of 100,000 and an angle of attack of 3 deg. A sample airfoil and its corresponding input image ($C_p$ distribution) are shown in Fig. 6.

Because the CNN requires gridlike data input, the $C_p$ distribution of the upper and lower surfaces is converted into inverted gray images of size $144 \times 144 \times 1$ separately. The inverted gray image consists of values from zero to one, as mentioned in Ref. [15]; the grid along which the $C_p$ line fully passes through will have a value of one, the grid along which the line partially passes through will have a value of less than one, and all other grids will have a value of zero. Finally, the input to the CNN architecture is a matrix of $144 \times 144 \times 2$ (upper and lower $C_p$ distribution images). In addition, the input data are prepared with sizes of $216 \times 216 \times 2$ and $144 \times 144 \times 1$ (combining the upper

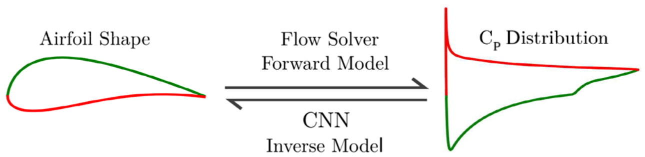

**Fig. 6 Illustration of forward (flow solver) and inverse (CNN) model.**

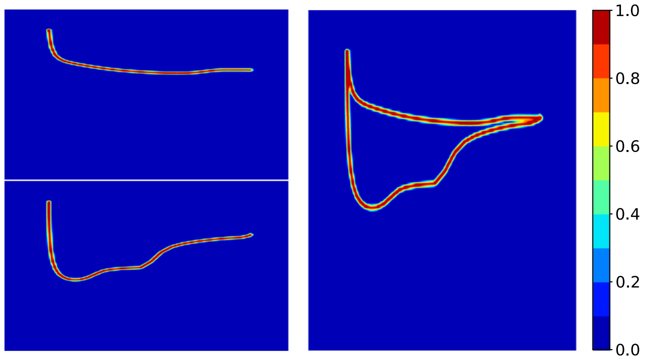

**Fig. 7 Visualization of $C_p$ input image: separated image ($144 \times 144 \times 2$) on left, and single image ($144 \times 144 \times 1$) on right.**

and lower $C_p$ into a single image). A sample of the $C_p$ input data with one and two images is shown in Fig. 7. The output of the CNN architecture is the airfoil $y$ coordinates at a fixed chord length. The output has 70 $y$-coordinate points in total: 35 for the upper airfoil curve, and another 35 for the lower airfoil curve.

## III. Results and Discussions

### A. CNN Training

The implementation, training, and predictions of the CNN architecture are performed with the open-source software Tensorflow [23] using Keras Application Programming Interface (API) [24]. CNN training is an optimization process in which the weights and the biases of the CNN are trained such that the defined mean square error (MSE) is minimized. The weights and biases of the CNN are trained using the backpropagation algorithm [19]. Let us define the MSE for a batch of $m$ training samples as follows:

$$E = \frac{1}{m}\sum_{m} \| y_t^m - y_p^m \|^2 \tag{5}$$

where $y_t$ are the true data, and $y_p$ is the CNN prediction. The trainable parameters are $W = [w_c \quad b_c \quad w_f \quad b_f]$, where $w_c$ and $b_c$ denote the kernel weights and biases of the convolutional layers and $w_f$ and $b_f$ denote the weights and biases of the fully connected layers. To optimize the weights and biases, one needs to obtain the gradients of the error function $E$ with respect to the weights and biases

$$E_W = \left[ \partial E/\partial w_c \quad \partial E/\partial b_c \quad \partial E/\partial w_f \quad \partial E/\partial b_f \right]$$

The backpropagation algorithm is applied to compute the gradient information $E_W$ [25]. Once the gradients are obtained, an optimizer is used to update the weights and biases. Minibatch mode training is adapted, in which the gradients are accumulated and the weights and biases are updated once for a minibatch of samples. The summarized learning steps for a minibatch of $m$ samples are shown as follows:

1) Input a set of $m$ training samples.
2) Obtain the CNN output $y_p$ using feed forward process
3) Compute the MSE error of output layer $E$.
4) Obtain the gradient information $E_W$ using backpropagation.
5) Update the trainable parameters using an optimizer.

Steps 1–5 indicate a single training iteration. Let us consider that we have $n$ training samples; the learning step is performed for all $n$ samples in a minibatch of $m$ samples. Exposing all the samples to the CNN once completes one epoch, which consists of several minibatch iterations. Several epochs are required to achieve the desired convergence.

A total number of 1200 airfoil data are fed to the CNN as input. Trainings are performed with a training/validation split of 80/20% of the fed input data. First, the trainable parameters $W$ are initialized either randomly or following a normal distribution. The optimization is performed using a stochastic gradient decent algorithm called ADAM [26]. ADAM is a first-order gradient-based algorithm that

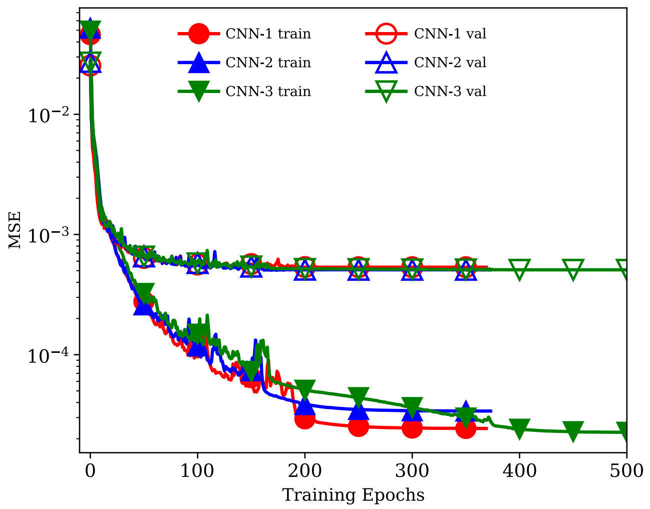

**Fig. 8 MSE convergence history of CNN-1, CNN-2, and CNN-3 with input size of $144 \times 144 \times 2$ (val indicates validation).**

uses an adaptive learning rate based on past gradient information for each parameter update. So far, ADAM may be the best choice of algorithm for deep learning [27]. The optimization iterations are carried out until the MSE of validation reaches a steady state. An initial learning rate of $5.0 \times 10^{-4}$ is chosen with a learning rate scheduler, which reduces the learning rate in steps when the MSE of validation shows no improvement over several epochs. Minibatch mode training is adopted with a batch size of 64, in which the weights and biases are updated after exposing the CNN to one batch of samples. Minibatch training is the preferred choice of learning mode in deep learning because of its quick convergence [28]. To avoid overfitting, training is terminated with a predefined early stopping criteria, when the MSE of validation shows no further improvement over several epochs even after reduction of the learning rate. The training and validation convergence of the MSE for some selected CNN architectures are shown in Fig. 8. Note that the training of CNN-1 and CNN-2 (to be defined next) is terminated by early stopping criteria at around 380 epochs, in which the MSE of validation has already reached a steady-state value. Furthermore, restarting the training shows no further improvement in convergence.

### B. Influence of CNN Hyperparameters

#### *1. Number of Convolutional Layers*

We investigated several CNN architectures by varying the hyperparameters, namely, the number of convolutional layers, the filter size and filter depth, and the number of fully connected layers. The used CNN architectures with their hyperparameters are listed in Table 1. Each convolution layer is followed by a RELU activation and pooling layer. Based on trial and error, a CNN architecture with three convolution layers and two fully connected layers is chosen as a base model called CNN-1. Furthermore, additional layers are added to CNN-2 and CNN-3, as given in Table 1. Training is performed for all the architectures, and the error convergence of the CNNs is shown in Fig. 8. CNN-1 and CNN-2 converge quickly, but to a slightly higher MSE value than CNN-3. CNN-3 achieves a better training and validation MSE convergence. However, all three architectures achieve a close convergence and it is clear that the listed architectures achieve one of the hyperparameter (number of convolutional layers) convergence. The CNN-3 architecture is used in the following study and predictions because of its least validation error. Furthermore, increasing the number of layers may have no effect on the convergence because there is no significant difference in the validation MSEs between CNN-2 and CNN-3.

#### *2. Convolutional Filter Size*

CNN-3 is considered as the base model to further study the effect of convolutional filter size. The filter sizes of first three layers are

Table 1 CNN architecture with its layers and parameters

| Layer type | CNN-1 | CNN-2 | CNN-3 |
|---|---|---|---|
| Input[a] | $I_1/I_2/I_3$ | $I_1/I_2/I_3$ | $I_1/I_2/I_3$ |
| First convolution[b] | $4 \times 4$, 32, $3 \times 3$ | $4 \times 4$, 32, $3 \times 3$ | $4 \times 4$, 32, $3 \times 3$ |
| Second convolution | $4 \times 4$, 64, $3 \times 3$ | $4 \times 4$, 64, $3 \times 3$ | $4 \times 4$, 64, $3 \times 3$ |
| Third convolution | $4 \times 4$, 128, $3 \times 3$ | $4 \times 4$, 128, $3 \times 3$ | $4 \times 4$, 128, $3 \times 3$ |
| Fourth convolution | — — | $2 \times 2$, 128, $2 \times 2$ | $2 \times 2$, 128, $2 \times 2$ |
| Fifth convolution | — — | — — | $2 \times 2$, 2562 $\times 2$ |
| Fully connected[c] | $2 \times 100$ | $3 \times 100$ | $3 \times 100$ |
| Output | 70 | 70 | 70 |

[a]$I_1$, $I_2$, and $I_3$ indicate input sizes of $144 \times 144 \times 2$, $216 \times 216 \times 2$, and $144 \times 144 \times 1$, respectively.
[b]In the first convolutional layer of CNN-1, $4 \times 4$ indicates the convolutional filter size, 32 indicates the number of filters, and $3 \times 3$ indicates the pooling filter size.
[c]In the fully connected layer, $2 \times 100$ indicates two layers with 100 neurons per layer.

Table 2 Variation of CNN-3 with filter size and number of filters

| Layer type | CNN-3 | CNN-3-FS6[b] | CNN-3-FS8 | CNN-3-N0.5x[c] | CNN-3-N2x |
|---|---|---|---|---|---|
| First convolution | $4 \times 4$, 32, $3 \times 3$ | 6[a]×6[a], 32, $3 \times 3$ | 8[a]×8[a], 32, $3 \times 3$ | $4 \times 4$, 16[a], $3 \times 3$ | $4 \times 4$, 64[a], $3 \times 3$ |
| Second convolution | $4 \times 4$, 64, $3 \times 3$ | 6[a]×6[a], 64, $3 \times 3$ | 8[a]×8[a], 64, $3 \times 3$ | $4 \times 4$, 32[a], $3 \times 3$ | $4 \times 4$, 128[a], $3 \times 3$ |
| Third convolution | $4 \times 4$, 128, $3 \times 3$ | 6[a]×6[a], 128, $3 \times 3$ | 8[a]×8[a], 128, $3 \times 3$ | $4 \times 4$, 64[a], $3 \times 3$ | $4 \times 4$, 256[a], $3 \times 3$ |
| Fourth convolution | $2 \times 2$, 128, $2 \times 2$ | $2 \times 2$, 128, $2 \times 2$ | $2 \times 2$, 128, $2 \times 2$ | $2 \times 2$, 64[a], $2 \times 2$ | $2 \times 2$, 256[a], $2 \times 2$ |
| Fifth convolution | $2 \times 2$, 256, $2 \times 2$ | $2 \times 2$, 256, $2 \times 2$ | $2 \times 2$, 256, $2 \times 2$ | $2 \times 2$, 128[a], $2 \times 2$ | $2 \times 2$, 512[a], $2 \times 2$ |

[a]Variations of convolutional layers; other layers are the same as CNN-3.
[b]FS6 indicates filter size of $6 \times 6$.
[c]N0.5x indicates number of filters are reduced by 0.5 times those of CNN-3.

altered to obtain CNN-3-FS6 (with $6 \times 6$) and CNN-3-FS8 (with $6 \times 6$) architectures, as shown in Table 2; and no other parameter is changed. Training is performed, and the MSE convergence history is shown in Fig. 9. Increasing the filter size from $4 \times 4$ to $6 \times 6$ improves the overall convergence slightly. Further increase of the filter size to $8 \times 8$ improves the validation convergence at the expense of training errors. For example, for a typical imageNet architecture with an input size of $227 \times 227$, filter sizes varying from $11 \times 11$ to $3 \times 3$ are generally used [29].

### 3. *Number of Convolutional Filters*

To investigate the effect of the number of filters, the number of filters in each convolutional layer of CNN-3 are reduced by a factor of two (CNN-3-N0.5x) and increased by a factor of two (CNN-3-N2x), as detailed in Table 2. The effect of the number of filters on the convergence of the MSE is shown in Fig. 10. It is evident from the results that a reduction of the number of filters (CNN-3-N0.5x) shows poor convergence. Increasing the number of filters from CNN-3 does not show significant improvement in the MSE convergence, but it has a higher computational cost. Hence, the number of convolutional filters listed in the CNN-3 architecture may be a better choice for similar problems.

### 4. *Number of Fully Connected Layer*

To study the effect of fully connected layers, CNN-3 is modified to have two and three fully connected layers (CNN-3-FC2 and CNN-3-FC4), as shown in Table 3. The influence of fully connected layers on convergence is given in Fig. 11. CNN-3-FC2 converges to a slightly higher validation MSE, and CNN-3-FC4 has slightly better validation MSE convergence. Because there is no significant difference in convergence, a fully connected layer of two to four may be a good choice.

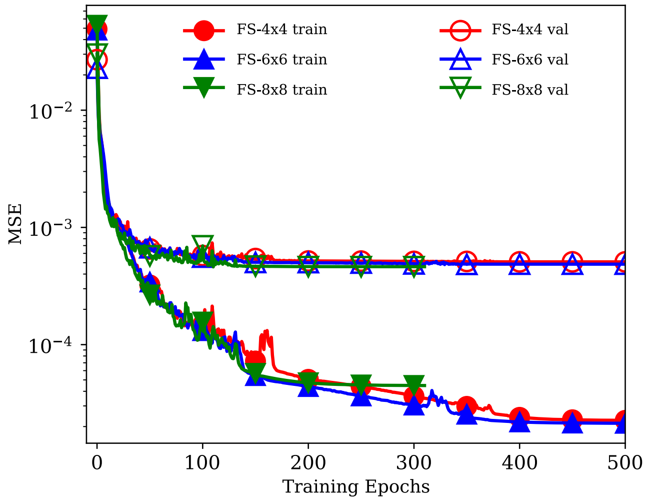

**Fig. 9 MSE convergence history of CNN-3 with different convolutional filter sizes (FSs). (FS4 indicates filter size of $4 \times 4$.)**

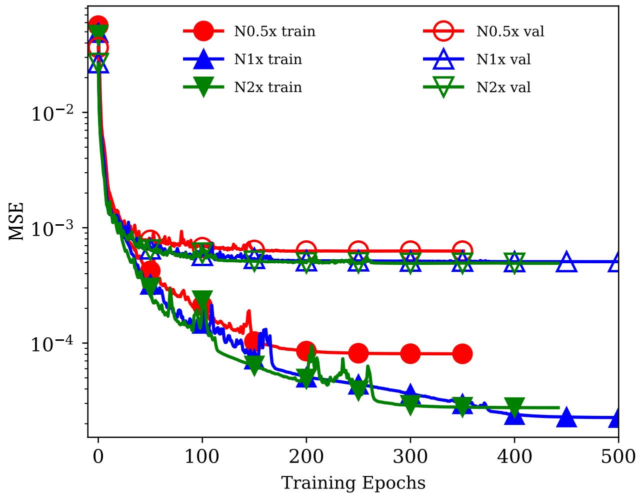

**Fig. 10 MSE convergence history of CNN-3 with different number of convolutional filters. (N0.5x indicates the number of filters are reduced by half from CNN-3.)**

**Table 3 Variation of CNN-3 with number of fully connected layers**

| Parameter | Value |
|---|---|
| Layer type | Fully connected |
| CNN-3 | $3 \times 100$ |
| CNN-3-FC2[b] | 2[a]×100 |
| CNN-3-FC4 | 4[a]×100 |

[a]Variations of fully connected layers; other layers are the same as CNN-3.
[b]FC2 indicates two fully connected layers.

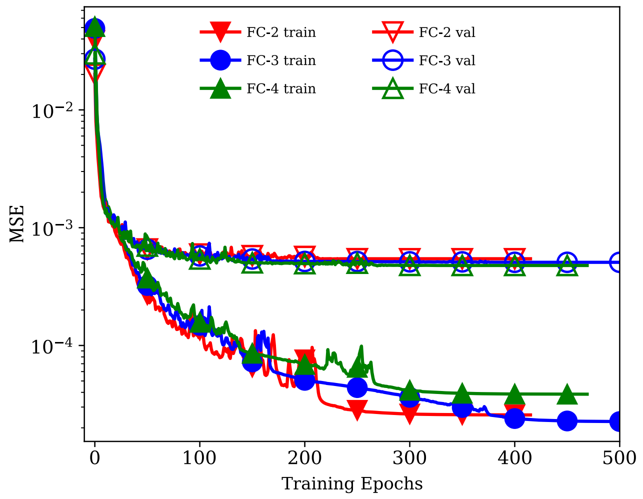

**Fig. 11 MSE convergence history of CNN-3 different number of fully connected (FC) layers. (FC2 indicates two fully connected layers.)**

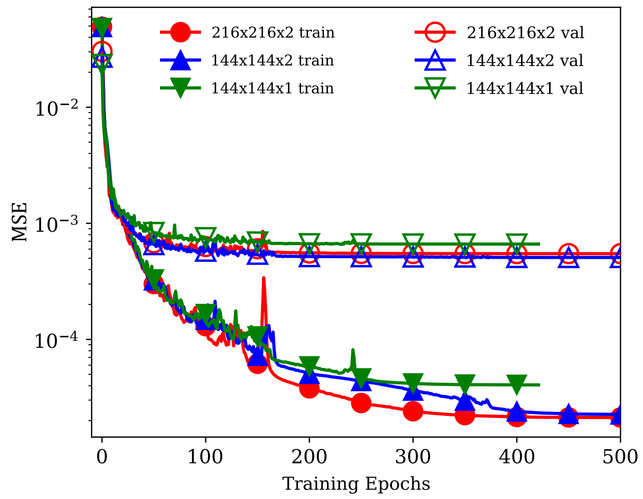

**Fig. 12 MSE convergence history of CNN-3 with different input sizes.**

*5. Other Parameters*

Because the pooling layer performs subsampling operation (reducing the dimensionality of input), it is recommended to use filter sizes not more than $3 \times 3$. The important features in the feature maps may be lost by choosing a pooling value larger than $3 \times 3$. In this study, pooling filters of size $3 \times 3$ and $2 \times 2$ are used with a stride of three and two, respectively. All the convolutional operations are performed with a stride of one.

### C. Influence of Input Image Size

Investigation is performed on the size of the input $C_p$ distribution image. In this case, the $C_p$ distributions over upper and lower airfoil surfaces are used as separate images. Two different sizes of input image with $144 \times 144 \times 2$ and $216 \times 216 \times 2$ are considered. CNN-3 architecture is trained using both input sizes, and the error convergences are shown in Fig. 12. It is evident from Fig. 12 that the convergence is not improved further by increasing the given input image size. In addition, both upper and lower $C_p$ distributions are combined and given as a single image size of $144 \times 144 \times 1$. The error convergence of CNN-3 with input $144 \times 144 \times 1$ is shown in Fig. 12, which indicates that the separate image for upper and lower $C_p$ distributions has slightly better convergence.

### D. Influence of CNN Training Parameters

*1. Learning Rate*

We investigate the MSE convergence of CNN-3 with a series of initial learning rates (LRs) varying from $1 \times 10^{-2}$ to $1 \times 10^{-5}$. The effect of the learning rate on convergence of the MSE is shown in Fig. 13. It is observed from the results that a larger initial LR has a faster convergence. However, for a large initial LR of $1 \times 10^{-2}$, the MSE does not converge below a value of $10^{-2}$ over several hundred epochs, even after reducing the LR to a minimum value. This may be due to the fact that the optimization algorithm skips the optimal path because of a large initial LR. A smaller initial LR always results in a better convergence to a minimum value, but it requires a large number of epochs, which is computationally inefficient. As shown in Fig. 13, a LR of $1 \times 10^{-5}$ converges smoothly, but it requires more than 2000 epochs to achieve the same convergence of LR $= 1 \times 10^{-3}$. Hence, it is recommended to use an optimal initial LR with a LR scheduler that reduces the LR when the MSE reaches a plateau. In this study, a safe initial LR of $5 \times 10^{-4}$ is used as the optimum for the training of CNNs.

*2. Minibatch Size*

Furthermore, the investigation is carried out to study the effect of minibatch size (MBS) on convergence. Three different minibatch sizes (32, 64, and 128) are chosen, and the effect on convergence is shown in Fig. 14. In minibatch mode training, the trainable parameters are updated for a minibatch of samples. Hence, choosing a smaller batch size may result in faster convergence, but it involves more iterations per epoch. Also, choosing a very small batch size may result in oscillatory convergence because the calculated error gradients for a small batch of samples may not fit well with the other samples. In this study, MBSs of 32 and 64 have oscillatory yet better

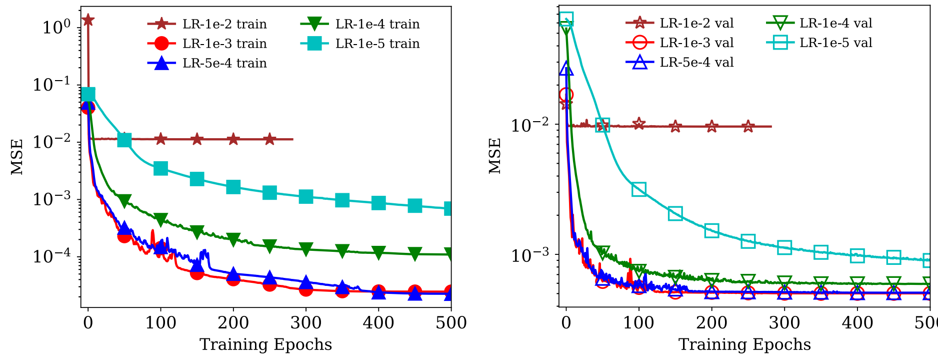

**Fig. 13 MSE convergence history of CNN-3 with different learning rates: training (left) and validation (right).**

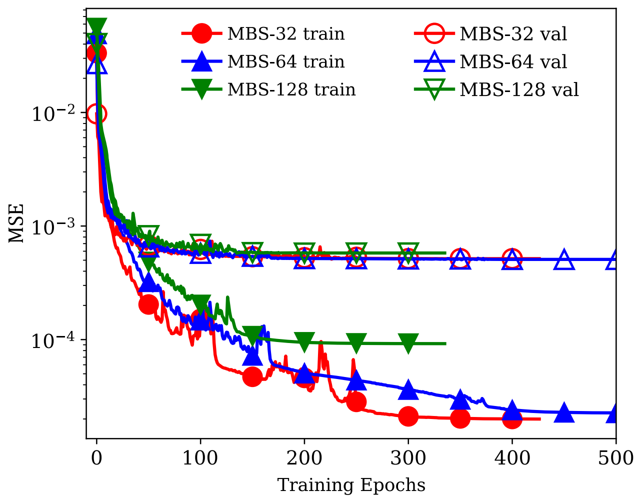

**Fig. 14 MSE convergence history of CNN-3 with different minibatch sample sizes. (MBS32 indicates minibatch size of 32 samples.)**

**Table 4 Comparison of CPU times for input size of $144 \times 144 \times 2$**

| Simulation type | CPU time (Intel Xeon at 3.3 GHz), CPU hours |
|---|---|
| CNN-3 training (1200 samples) | 113.3 |
| CNN-3 prediction per airfoil | $1.3 \times 10^{-6}$ |
| Conventional methods [30] | 7.0 |

convergence than a MBS of 128. Hence, we use a MBS of 64 as the optimum for the training. However, it is important to note that the recommended minibatch size depends on the size of the training data. Generally, it is recommended to use a minibatch size between 32 and 256 [28].

### E. CNN Predictions

The trained CNN-3 architecture is used to perform the predictions of the training and testing set airfoils. The required CPU time of CNN-3 for training and predictions are shown in Table 4 [30], which indicates that, when the database is ready and the training is done, the CNN requires less than 1 s to predict an airfoil shape. A dataset consists of 143 airfoils, which are unseen by the CNN during training, that are used for testing the trained CNN. The qualitative accuracy spreads of the training and testing samples are shown in Fig. 15. The points are clustered along the 45 deg line, which shows that the predicted values are close to true value. The computed $L_2$ relative error for the training prediction and testing prediction comparison chart is shown in Fig. 16 quantitatively. From Fig. 16, it is clear that, for the training set prediction, the errors are less than 0.5% for the majority of the samples; for the testing set prediction, the errors are less than 2.0% for the majority of the samples. For some selected airfoils, the output of the CNNs are shown in Fig. 17 for training and in Fig. 18 for testing cases. The comparison shows good match of the predicted airfoil profile with the true one, and these plots demonstrate the generalization performance of the trained CNN architecture for the given task. Therefore, we can conclude that the trained CNN shows reasonable performance for the prediction of airfoil shape from the given $C_p$ distribution.

In addition, the averaged $L_2$ relative error of all the CNN architectures is listed in Table 5, which indicates that, overall, most of the defined architectures are able to perform well in both training and testing set predictions. All the architectures have an averaged $L_2$ relative error of less than 0.11% for the training set prediction and less than 0.65% for the testing set prediction. The $L_2$ relative error distributions of all the architectures are shown in Figs. 19 and 20 for training and testing cases respectively, which indicates the variation of the $L_2$ relative error distribution with respect to hyperparameters. From the error distribution, it is observed that there are still a few outliers in the prediction of all the CNN architectures. This is due to the fact that the $C_p$ distribution generated using XFOIL has slight oscillations. It is possible that the CNN learns and maps the oscillations into a perturbed airfoil profile. To examine the true

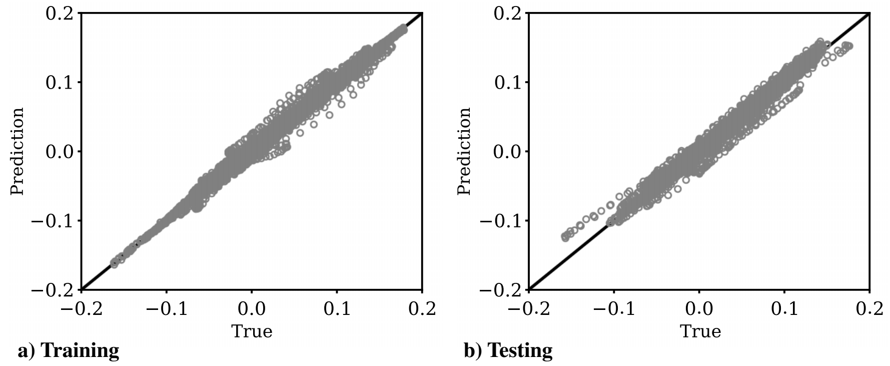

**Fig. 15 CNN-3 prediction with accuracy spread (true vs predicted): training set (left) and testing set (right).**

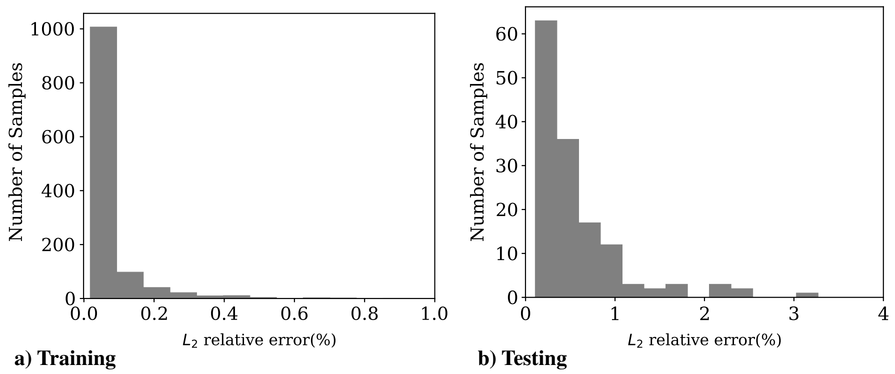

**Fig. 16 CNN-3 ($144 \times 144 \times 2$) prediction with $L_2$ error distribution: training set (left) and testing set (right).**

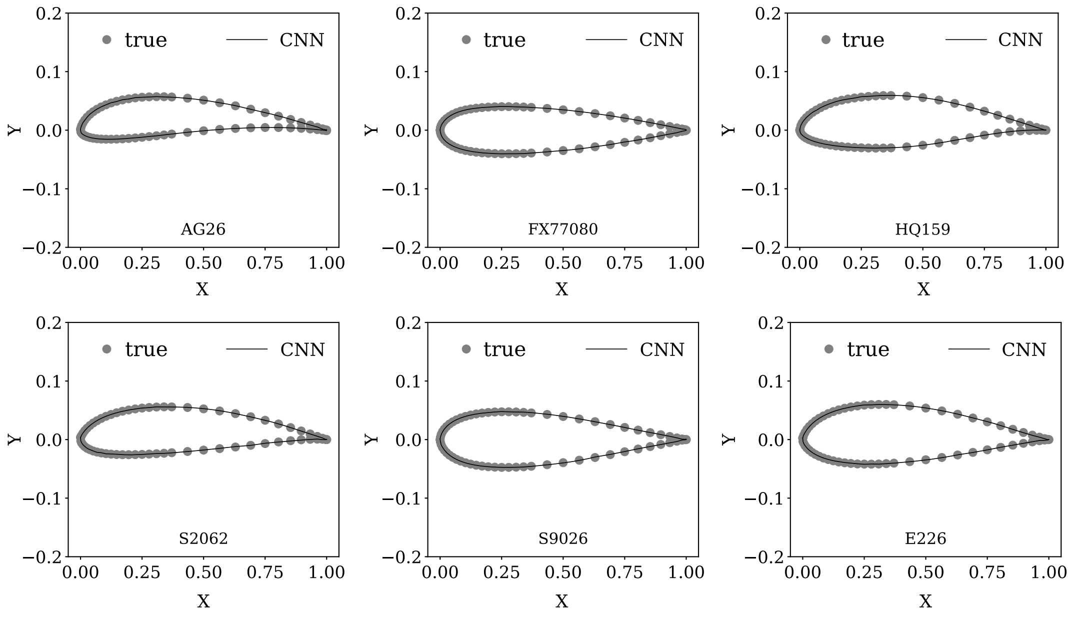

Fig. 17 CNN-3 (144 × 144 × 2) prediction of typical training airfoils. (Airfoil name is indicated below the airfoil.)

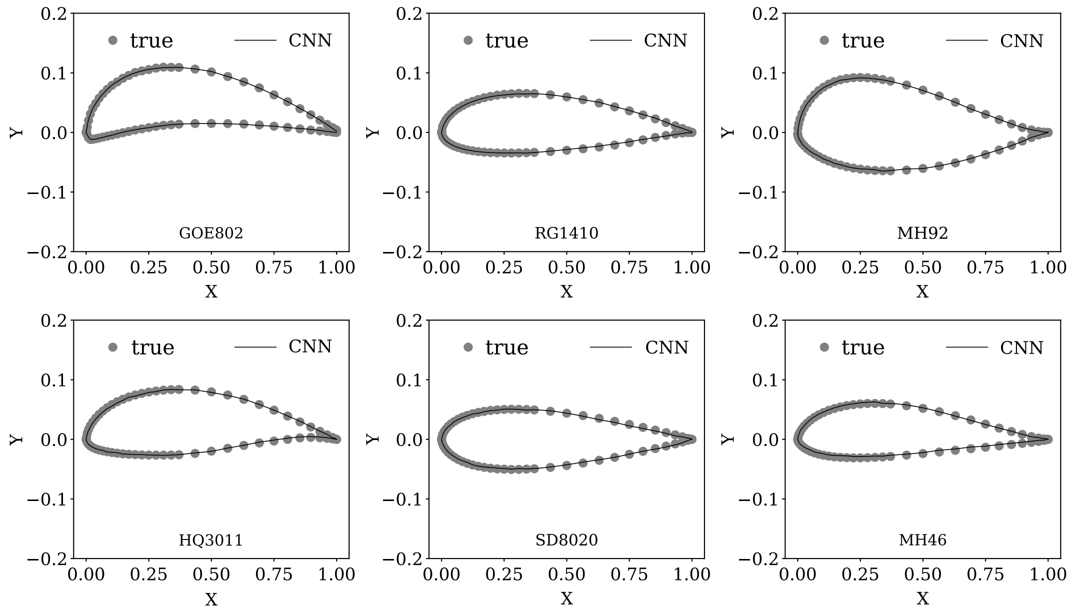

Fig. 18 CNN-3 (144 × 144 × 2) prediction of typical testing airfoils. (Airfoil name is indicated below the airfoil.)

Table 5 Averaged $L_2$ relative error comparison of different CNN architecture predictions[a]

| CNN | Training error, % | Testing error, % |
|---|---|---|
| CNN-1 | 0.078822 | 0.614078 |
| CNN-2 | 0.084819 | 0.603569 |
| CNN-3 | 0.075504 | 0.602819 |
| CNN-3-$I_2$ | 0.075551 | 0.632623 |
| CNN-3-$I_3$ | 0.092232 | 0.615975 |
| CNN-3-FS6 | 0.071971 | 0.563247 |
| CNN-3-FS8 | 0.086556 | 0.556368 |
| CNN-3-N0.5x | 0.114487 | 0.649495 |
| CNN-3-N2x | 0.075977 | 0.589517 |
| CNN-3-FC2 | 0.078581 | 0.606387 |
| CNN-3-FC4 | 0.085607 | 0.601700 |
| CNN-3-MBS32 | 0.071947 | 0.588935 |
| CNN-3-MBS128 | 0.115051 | 0.659968 |

[a]Input type $I_1$ is used as default unless stated explicitly. Also, a LR of $5 \times 10^{-4}$ and a MBS of 64 are used unless stated explicitly.

accuracy of the CNN, it will thus be necessary to apply a NS solver in the aforementioned procedure, which can be a future work.

## IV. Conclusions

In this work, a methodology is proposed to perform the inverse design of an airfoil using a deep convolutional neural network. The coefficient of pressure distribution $C_p$ is input to the CNN, and the airfoil shape is obtained as output. Several CNN architectures (CNN-1, CNN-2, and CNN-3) with varying layer depths and hyperparameters are trained. From the error convergence, it is clear that the chosen architectures have similar error convergence, and the present training achieves convergence of layer depth. In addition, with the chosen CNN-3 architecture, the effect of the input image size on performance is analyzed. From the error convergence of CNN-3 with two different input image sizes (216 × 216 × 2 and 144 × 144 × 2), it is clear that the results are not improved further by increasing the input image size. Furthermore, with CNN-3, the training is performed for one (upper and lower $C_p$ combined) and two (upper and lower $C_p$

separated) image inputs. The error convergence shows that the double-image input is slightly more accurate than the single-image input. In addition, the CNN-3 architecture is modified to have different filter sizes, numbers of filters, and numbers of fully connected layers. Among the parameters mentioned, reducing the number of filters (CNN-3-N0.5x) significantly affects the MSE convergence, and the

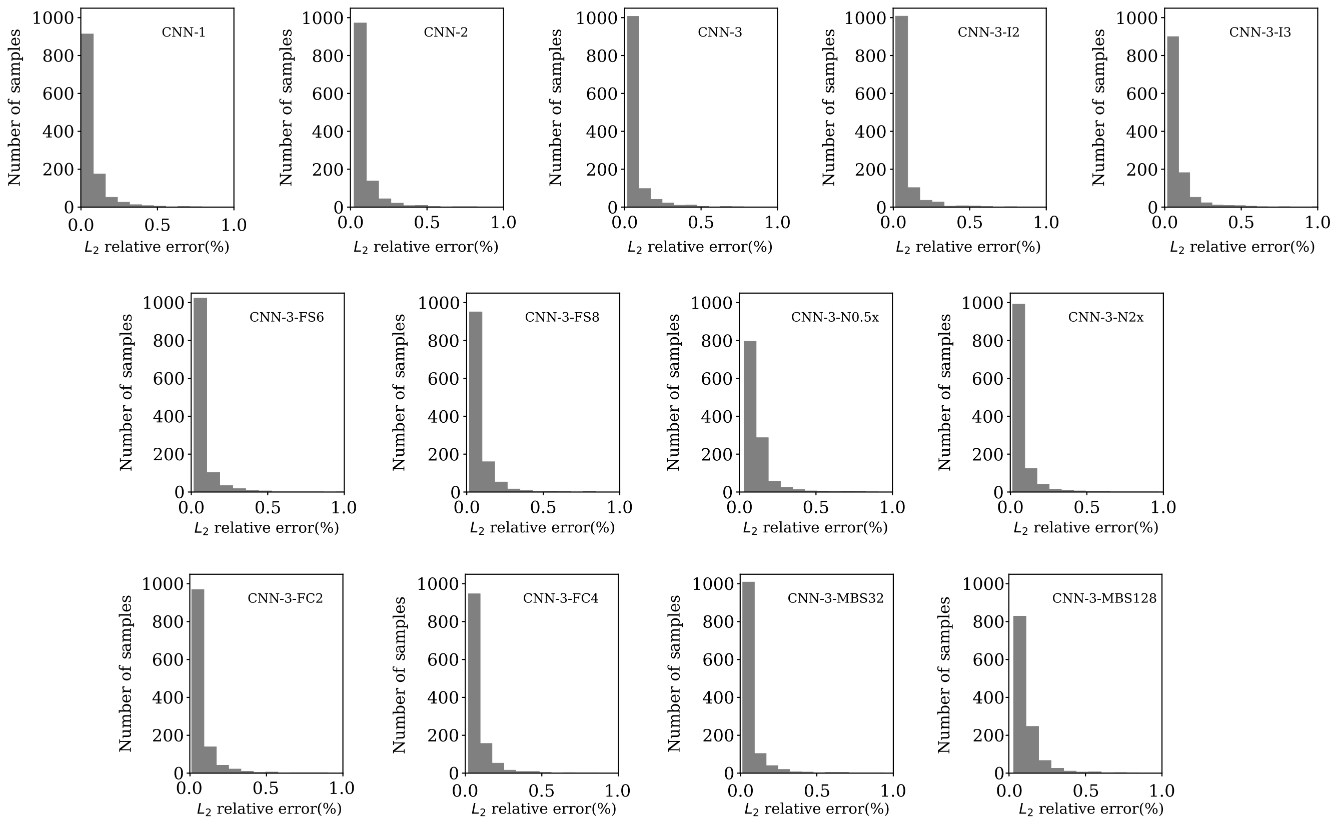

**Fig. 19** ***$L_2$* relative error (in percentages) distribution of training set prediction of different CNN architectures.**

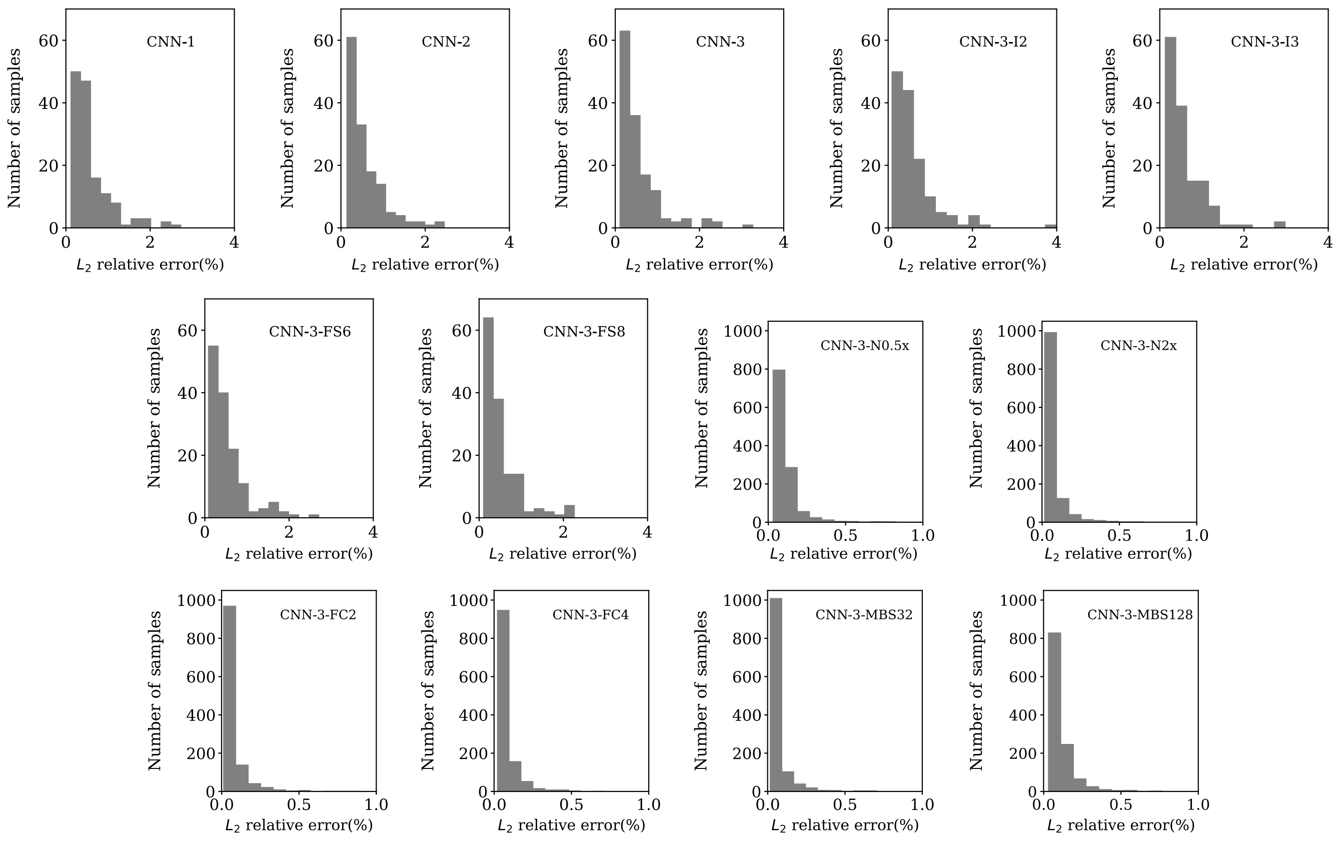

**Fig. 20** ***$L_2$* relative error (in percentages) distribution of test set prediction of different CNN architectures.**

variation of other parameters shows no significant difference in the MSE convergence. Furthermore, CNN-3 is trained with varying learning rates and minibatch sizes. It is observed from the results that choosing an optimum initial learning rate and minibatch size is important to achieve a faster and better convergence of the MSE. It is important to note that the mentioned effect of the variation of the hyperparameters and training parameters on the MSE convergence is specific to similar problems with a similar size of database.

The predictions are performed for training and testing samples with the CNN-3 architecture along with an image input of $144 \times 144 \times 2$. The predicted results show that the present approach has reasonable accuracy for both training and testing samples. Hence, it can be concluded that the CNN is able to distill the underlying function to map the $C_p$ distribution into the airfoil shape and it shows good generalization performance in the prediction of testing samples that are unseen by the CNN during training. In addition, it is worth mentioning that, when training is done, the time taken for CNN prediction per airfoil takes less than 1 s; whereas conventional methods require time on the order of hours, although conventional methods are useful in the case in which a new task is assigned. As the results show, the method emerges as a gradient-free, fast, robust, global, and accurate tool, complementing the other conventional methods in the arsenal of fluid dynamists. Therefore, the present CNN-based method can be directly deployed to engineering tasks that involve real-time airfoil design or, at least, can be used to provide an initial solution to optimization. Future work may include the inverse design of airfoils by considering the effect of the Reynolds number and the angle of attack. In addition, it will be interesting to explore the inverse design capability of the CNN for three-dimensional wing geometries. Overall, the CNN-based inverse design method is very general and can be widely applied across the spectrum of engineering.

## Appendix: Sample Convolution Feature Maps

To visualize the convolution process, a few convolved feature maps of the convolutional layers of CNN-3 with an input size of $144 \times 144 \times 1$ are shown in Fig. A1.

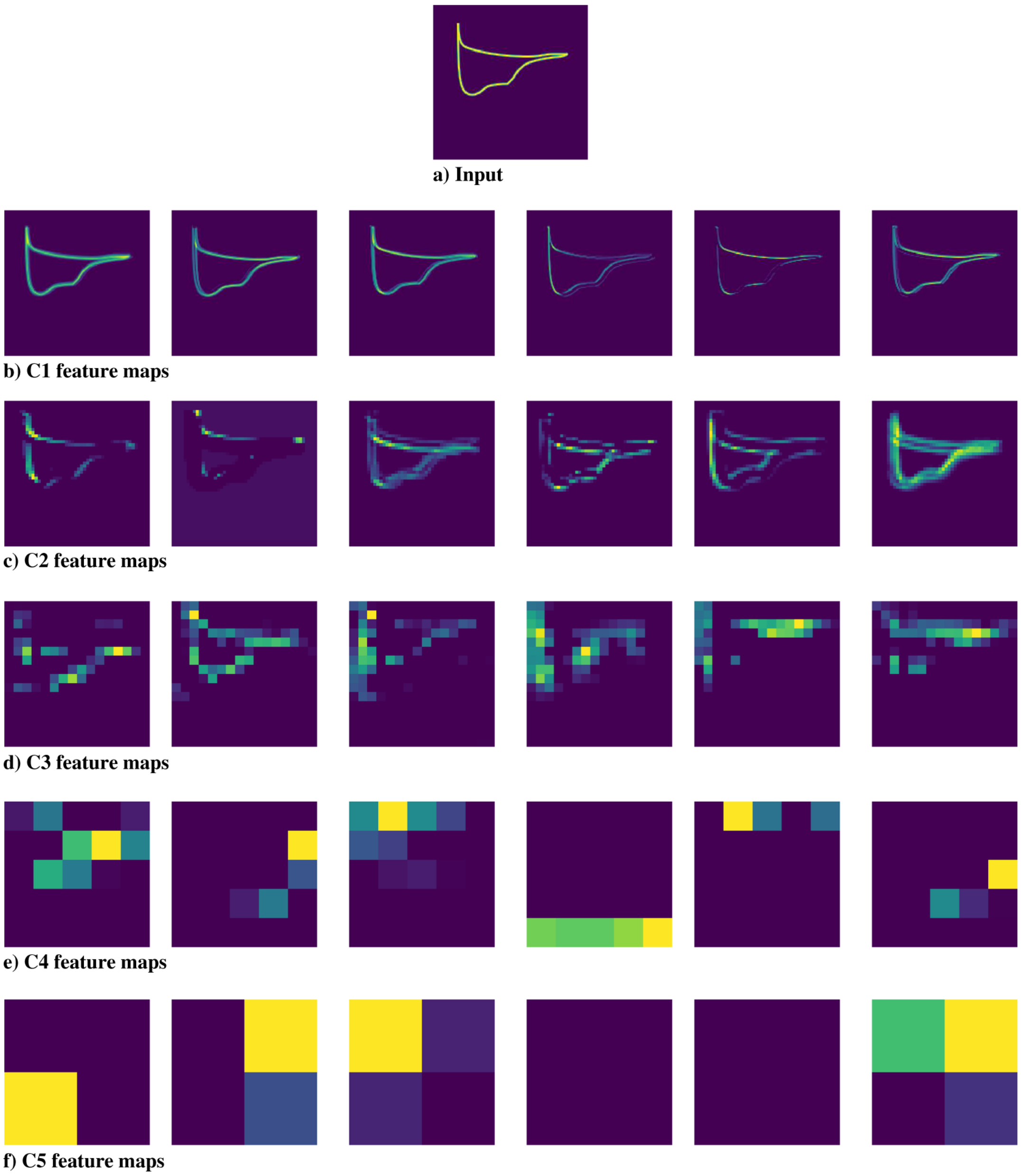

**Fig. A1 CNN-3 (144 × 144 × 1) with first six convolved feature maps of convolutional layers. C1 indicates first convolutional layer.**

## Acknowledgments

The authors are thankful to the National Supercomputing Centre, Singapore, for providing access to their supercomputing facility. In addition, the authors are thankful to the National University of Singapore and the Ministry of Education, Singapore, for the research scholarship.

## References

[1] Jameson, A., "Aerodynamic Design Via Control Theory," *Journal of Scientific Computing*, Vol. 3, No. 3, 1988, pp. 233–260. doi:10.1007/BF01061285

[2] Skinner, S. N., and Zare-Behtash, H., "State-of-the-Art in Aerodynamic Shape Optimisation Methods," *Applied Soft Computing Journal*, Vol. 62, Jan. 2018, pp. 933–962. doi:10.1016/j.asoc.2017.09.030

[3] Venter, G., and Sobieszczanski-Sobieski, J., "Particle Swarm Optimization," *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, Materials Conference, and Co-Located Conferences*, AIAA Paper 2002-1235, 2002. doi:10.2514/6.2002-1235

[4] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed., Addison-Wesley Longman, Boston, MA, 1989, pp. 1–25.

[5] Catalano, L. A., "Aerodynamic Shape Design Using Hybrid Evolutionary Computing and Multigrid-Aided Finite-Difference Evaluation of Flow Sensitivities," *Engineering Computations*, Vol. 32, No. 2, 2015, pp. 178–210. doi:10.1108/EC-02-2013-0058

[6] Lee, C., and Wu, J.-Z., "Transition in Wall-Bounded Flows," *Applied Mechanics Reviews*, Vol. 61, No. 3, 2008, Paper 030802. doi:10.1115/1.2909605

[7] Tang, Q., Zhu, Y., Chen, X., and Lee, C., "Development of Second-Mode Instability in a Mach 6 Flat Plate Boundary Layer with Two-Dimensional Roughness," *Physics of Fluids*, Vol. 27, No. 6, 2015, Paper 064105. doi:10.1063/1.4922389

[8] Duraisamy, K., Iaccarino, G., and Xiao, H., "Turbulence Modeling in the Age of Data," *Annual Review of Fluid Mechanics*, Vol. 51, No. 1, 2019. doi:10.1146/annurev-fluid-010518-040547

[9] Ling, J., Kurzawski, A., and Templeton, J., "Reynolds Averaged Turbulence Modelling Using Deep Neural Networks with Embedded Invariance," *Journal of Fluid Mechanics*, Vol. 807, Nov. 2016, pp. 155–166. doi:10.1017/jfm.2016.615

[10] Rai, M. M., and Madavan, N. K., "Aerodynamic Design Using Neural Networks," *AIAA Journal*, Vol. 38, No. 1, 2000, pp. 173–182. doi:10.2514/2.938

[11] Kharal, A., and Saleem, A., "Neural Networks Based Airfoil Generation for a Given $C_p$ Using Bezier-PARSEC Parameterization," *Aerospace Science and Technology*, Vol. 23, No. 1, 2012, pp. 330–344. doi:10.1016/j.ast.2011.08.010

[12] Sun, G., Sun, Y., and Wang, S., "Artificial Neural Network Based Inverse Design: Airfoils and Wings," *Aerospace Science and Technology*, Vol. 42, April–May 2015, pp. 415–428. doi:10.1016/j.ast.2015.01.030

[13] Samareh, J. A., "A Survey of Shape Parameterization Techniques," *NASA CP-1999-209136*, June 1999, pp. 333–344, https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19990050940.pdf.

[14] LeCun, Y., Bengio, Y., and Hinton, G., "Deep learning," *Nature*, Vol. 521, No. 7553, 2015, pp. 436–444. doi:10.1038/nature14539

[15] Guo, X., Li, W., and Iorio, F., "Convolutional Neural Networks for Steady Flow Approximation," *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press, New York, 2016, pp. 481–490. doi:10.1145/2939672.2939738

[16] Zhang, Y., Sung, W.-J., and Mavris, D., "Application of Convolutional Neural Network to Predict Airfoil Lift Coefficient," *AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech Forum*, AIAA Paper 2018-1903, 2018.

[17] Yilmaz, E., and German, B., "A Convolutional Neural Network Approach to Training Predictors for Airfoil Performance," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, AIAA AVIATION Forum*, AIAA Paper 2017-3660, June 2017, pp. 1–19. doi:10.2514/6.2017-3660

[18] Rosenblatt, F., "The Perceptron, a Perceiving and Recognizing Automaton Project Para," Vol. 85, Nos. 460–461, Rept. Cornell Aeronautical Lab., Buffalo, NY, 1957.

[19] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Representations by Back-Propagating Errors," *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, MA, 1988, pp. 696–699, http://dl.acm.org/citation.cfm?id=65669.104451 [retrieved 2018].

[20] Nielsen, M. A., *Neural Network and Deep Learning*, Determination Press, 2015, pp. 39–53.

[21] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D., "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, Vol. 1, No. 4, 1989, pp. 541–551. doi:10.1162/neco.1989.1.4.541

[22] Drela, M., *XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils BT-Low Reynolds Number Aerodynamics*, Springer, Berlin, 1989, pp. 1–12.

[23] Abadi, M., et al., "TensorFlow: A System for Large-Scale Machine Learning," *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*, USENIX Association, Berkeley, CA, 2016, pp. 265–283, https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf [retrieved 2018].

[24] Chollet, F., "Keras: The Python Deep Learning Library (online database)," 2015, https://keras.io [retrieved 09 Sept. 2018].

[25] Goodfellow, I. J., "Multidimensional, Downsampled Convolution for Autoencoders," Univ. of Montreal TR, Montreal, 2010.

[26] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations (ICLR)*, Vol. 5, 2015.

[27] Ruder, S., "An Overview of Gradient Descent Optimization Algorithms," [cs.LG], 2017.

[28] Bengio, Y., "Practical Recommendations for Gradient-Based Training of Deep Architectures," *Neural Networks: Tricks of the Trade*, Springer, New York, 2012, pp. 437–478.

[29] 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 (NIPS'12)*, edited by Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., Vol. 1, Curran Associates Inc., 2012, pp. 1097–1105.

[30] Tashnizi, E. S., Taheri, A. A., and Hekmat, M. H., "Investigation of the Adjoint Method in Aerodynamic Optimization Using Various Shape Parameterization Techniques," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, Vol. 32, No. 2, April/June 2010, pp. 176–186, http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1678-58782010000200012&nrm=iso [retrieved 2018].

R. K. Kapania
*Associate Editor*