AIAA Journal
Vol. 58, No. 1, January 2020

Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows

Nils Thuerey,[*] Konstantin Weißenow,[†] Lukas Prantl,[‡] and Xiangyu Hu[§]
Technical University of Munich, 85748 Garching, Germany

https://doi.org/10.2514/1.J058291

This study investigates the accuracy of deep learning models for the inference of Reynolds-averaged Navier–Stokes (RANS) solutions. This study focuses on a modernized U-net architecture and evaluates a large number of trained neural networks with respect to their accuracy for the calculation of pressure and velocity distributions. In particular, it is illustrated how training data size and the number of weights influence the accuracy of the solutions. With the best models, this study arrives at a mean relative pressure and velocity error of less than 3% across a range of previously unseen airfoil shapes. In addition all source code is publicly available in order to ensure reproducibility and to provide a starting point for researchers interested in deep learning methods for physics problems. Although this work focuses on RANS solutions, the neural network architecture and learning setup are very generic, and applicable to a wide range of partial differential equation boundary value problems on Cartesian grids.

I. Introduction

DESPITE the enormous success of deep learning methods in the field of computer vision [1–3], and first success stories of applications in the area of physics simulations [4–6], the corresponding research communities retain a skeptical stance toward deep learning algorithms [7]. This skepticism is often driven by concerns about the accuracy achievable with deep learning approaches. The advances of practical deep learning algorithms have significantly outpaced the underlying theory [8], and hence many researchers see these methods as black-box methods that cannot be understood or analyzed.

With the following study our goal is to investigate the inference of Reynolds-averaged Navier–Stokes (RANS) simulations of airfoils in two dimensions with deep learning methods, that is, neural networks (NNs). Once trained, these networks can be evaluated very efficiently and thus make it possible to obtain RANS solutions with significantly reduced runtimes compared with a regular solver. Additionally, we investigate the accuracy of trained models. We also illustrate that despite the lack of theoretical guarantees, deep learning methods can be analyzed and employed thanks to the large number of existing practical examples. We show how the accuracy of the flow predictions changes with respect to central training parameters such as network size and the number samples available for training.

A second closely connected goal of our work is to provide a public test bed and evaluation platform for deep learning methods in the context of computational fluid dynamics (CFD). Both code and training data are publicly available at https://github.com/thunil/Deep-Flow-Prediction [9], and are kept as simple as possible to allow for quick adoption for experiments and further studies. As learning task we focus on the direct inference of RANS solutions from a given choice of boundary conditions, that is, airfoil shape and freestream velocity. The specification of the boundary conditions as well as the solution of the flow problems will be represented by Eulerian field functions, that is, Cartesian grids. For the solution we typically consider velocity and pressure distributions. Deep learning as a tool makes sense in this setting, as the functions we are interested in, that is, velocity and pressure, are smooth and well represented on Cartesian grids. Also, convolutional layers, as a particularly powerful component of current deep learning methods, are especially well suited for such grids.

The learning task for our goal is very simple when seen on a high level: we have a unique relationship between the inputs in the form of boundary conditions and the solution, we have full control of the data generation process, we can generate arbitrary amounts of samples with very little noise in the solutions, and we can train our models in a fully supervised manner. The difficulties rather stem from the complex relationship between inputs and the solutions, and the high requirements for accuracy. To illustrate the inherent capabilities of deep learning in the context of flow simulations, we will also intentionally refrain from including any specialized physical priors such as conservation laws. Instead, we will employ straightforward, state-of-the-art convolutional neural network (CNN) architectures and evaluate in detail, based on more than 500 trained CNN models, how well they can capture the behavior of the RANS equations. As a consequence, the setup we describe in the following is a very generic approach for boundary value problems of partial differntial equations (PDEs) problems and as such is applicable to a variety of other equations beyond RANS.

II. Related Work

Although machine learning in general has a long history in the field of CFD [10], more recent progress mainly originates from the advent of deep learning, which can be attributed to the seminal work of Krizhevsky et al. [1], who were the first to employ deep CNNs in conjunction with graphics processing unit (GPU)-based back-propagation training. Beyond the original goals of computer vision [11–13], targeting physics problems with deep learning algorithms has become a field of research that receives strongly growing interest. Especially problems that involve dynamical systems pose highly interesting challenges. Among them, several papers have targeted predictions of discrete Lagrangian systems. For example, Battaglia et al. [14] introduced a network architecture to predict two-dimensional rigid-body dynamics, which also can be employed for predicting object motions in videos [15]. The prediction of rigid-body dynamics with a different architecture was proposed by Chang et al. [16], whereas improved predictions for Lagrangian systems were targeted by Yu et al. [17]. Other researchers have used recurrent forms of NNs to predict Lagrangian trajectories for objects in height fields [18].

Deep learning algorithms have also been used to target a variety of flow problems, which are characterized by continuous dynamics in high-dimensional Eulerian fields. Several of the methods were proposed in numerical simulation, to speed up the solving process. For example, CNN-based pressure projections were proposed [4,19], whereas others have considered learning time integration [20]. The

Received 27 February 2019; revision received 16 July 2019; accepted for publication 25 September 2019; published online 9 November 2019.

[*]Professor, Department of Informatics, 15 Boltzmannstr. 3.

[†]Master Student, Department of Informatics, 15 Boltzmannstr. 3.

[‡]Ph.D. Student, Department of Informatics, 15 Boltzmannstr. 3.

[§]Adjunct Teaching Professor, Department of Mechanical Engineering, 15 Boltzmannstr. 3.

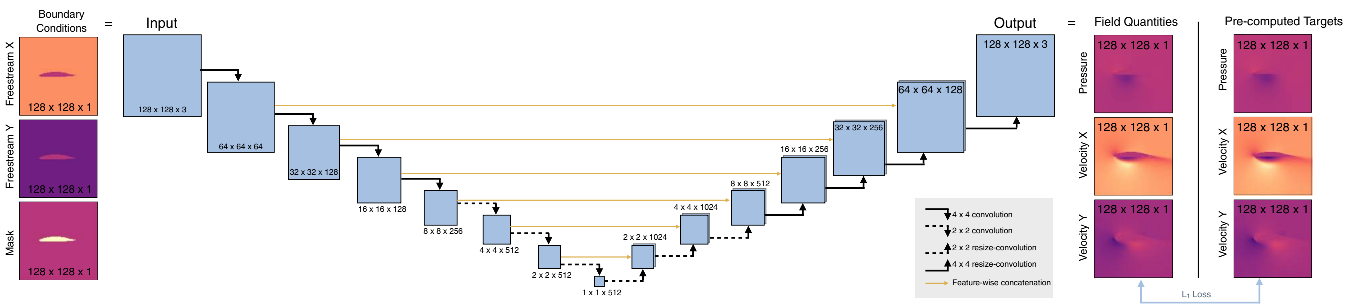


Fig. 1 An overview of the neural network architecture with input and output fields. References are shown on the right side. The black arrows denote convolutional layers, and orange arrows indicate skip connections.

work of Chu and Thuerey [21] targets an approach for increasing the resolution of a fluid simulation with the help of learned descriptors. Xie et al. [5], on the other hand, developed a physically based generative adversarial networks (GANs) model for super-resolution flow simulations. Deep learning algorithms also have potential for coarse-grained closure models [22,23]. Additional works have targeted the probabilistic learning of Scramjet predictions [24], and a Bayesian calibration of turbulence models for compressible jet flows [25]. Learned aerodynamic design models have also received attention for inverse problems such as shape optimization [26–28].

In the context of RANS simulations, machine learning methods were successfully applied for modeling Reynolds stresses from DNS data [29,30]. Additionally, deep learning techniques were employed to address turbulence uncertainty [31–33], as well as parameters for more accurate models [34]. Ling et al., on the other hand, proposed an NN-based method to learn anisotropy tensors for RANS-based modeling [35]. NNs were also successfully used to improve turbulence models for simulations of flows around airfoils [36]. An important difference of our approach to the aforementioned works, which typically embed a trained model in a numerical solver, is that our learning approach bypasses the PDE model and infers solutions purely via the chosen CNN architecture.

The data-driven paradigm of deep learning methods has also led to algorithms that learn reduced representations of space–time fluid data sets [37,38]. NNs in conjunction with proper orthogonal decompositions to obtain reduced representations of the flow were also explored [39]. Similar to our work, these approaches have the potential to yield new solutions very efficiently by focusing on a known, constrained region of flow behavior. However, these works target the learning of reduced representations of the solutions, whereas we target the direct inference of solutions given a set of boundary conditions.

Closer to the goals of our work, Farimani et al. proposed a conditional GAN to infer numerical solutions of heat diffusion and lid-driven cavity problems [40]. Methods to learn numerical discretization [41] and to infer flow fields with semisupervised learning [42] were also proposed recently. Zhang et al. developed a CNN that infers the lift coefficient of airfoils [43]. We target the same setting, but our networks aim for the calculation of high-dimensional velocity and pressure fields.

While our work focuses on the U-Net architecture [44], as shown in Fig. 1, a variety of alternatives have been proposed over the last years [45–47]. Adversarial training in the form of GANs [48,49] is likewise very popular. These GANs encompass a large class of modern deep learning methods for image super-resolution methods [50] and for complex image translation problems [2,51]. Adversarial training approaches have also led to methods for the realistic synthesis of porous media [52], or point-based geometries [53].

In the following, we will focus on RANS simulations. Here, a variety of application areas [54–57], hybrid methods [58], and modern variants [59–61] exist. We target the classic Spalart–Allmaras (SA) model [62], as this type of solver represents a well-established and studied test case with practical industry relevance.

III. Nonlinear Regression with Neural Networks

NNs can be seen as a general methodology to regress arbitrary nonlinear functions f. In the following, we give a very brief overview. More in-depth explanations can be found in corresponding books [63,64].

We consider problems of the form $\boldsymbol{y} = \hat{f}(\boldsymbol{x})$; that is, for a given input $\boldsymbol{x}$ we want to approximate the output $\boldsymbol{y}$ of the true function $\hat{f}$ as closely as possible with a representation f based on the degrees of freedom $\boldsymbol{w}$ such that $\boldsymbol{y} \approx f(\boldsymbol{x}, \boldsymbol{w})$. In the following, we choose NNs to represent f. NNs model the target functions with networks of nodes that are connected to one another. Nodes typically accumulate values from previous nodes, and apply the so-called activation functions g. These activation functions are crucial to introduce nonlinearity and effectively allow NNs to approximate arbitrary functions. Typical choices for g are hyperbolic tangent, sigmoid, and rectified linear unit (ReLU) functions.

The previous description can be formalized as follows: for the layer l in the network, the output of the ith node $a_{i,l}$ is computed with

$$a_{i,l} = g\left(\sum_{j=0}^{n_{l-1}} w_{ij,l-1} a_{j,l-1}\right) \tag{1}$$

Here, n_l denotes number of nodes per layer. To model the bias, that is, a per node offset, we assume $a_{0,l} = 1$ for all l. This bias is crucial to allow nodes to shift the input to the activation function. We employ this commonly used formulation to denote all degrees of freedom with the weight vector $\boldsymbol{w}$. Thus, we will not explicitly distinguish regular weights and biases below. We can rewrite Eq. (1) using a weight matrix W as $\boldsymbol{a}_l = \boldsymbol{g}(W_{l-1}\boldsymbol{a}_{l-1})$. In this equation, $\boldsymbol{g}$ is applied componentwise to the input vector. Note that without the nonlinear activation functions $\boldsymbol{g}$ we could represent a whole network with a single matrix W_0, that is, $\boldsymbol{a} = W_0\boldsymbol{x}$.

To compute the weights, we have to provide the learning process with a loss function $L(\boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{w}))$. This loss function is problem specific and typically has the dual goal to evaluate the quality of the generated output with respect to $\boldsymbol{y}$, as well as reduce the potentially large space of solutions via regularization. The loss function L needs to be at least once differentiable, so that its gradient $\nabla_y L$ can be back-propagated into the network in order to compute the weight gradient $\nabla_w L$. To obtain a robust estimate of the gradient without requiring the full set of training data, it is typically computed for a small, randomly selected subset, a *mini-batch* or in short *batch*.

Moving beyond fully connected layers, where all nodes of two adjacent layers are densely connected, the so-called *convolutional layers* are central components that drive many successful deep learning algorithms. These layers make use of fixed spatial arrangements of the input data, in order to learn filter kernels of convolutions. Thus, they represent a subset of fully connected layers typically with much fewer weights. It has been shown that these convolutional layers are able to extract important *features* of the input data, and each convolutional layer typically learns a whole set of convolutional kernels.

The convolutional kernels typically have only a small set of weights, for example, $n \times n$ with $n = 5$ for a two-dimensional data set. As the inputs typically consist of vector quantities, for example, i different channels of data, the number of weights for a convolutional layer with j output features is $n^2 \times i \times j$, with n being the size of the kernel. These convolutions extend naturally to higher dimensions.

To learn and extract features with larger spatial extent, it is typically preferable to reduce or enlarge the size of the inputs rather than enlarging the kernel size. For these resizing operations, NNs commonly employ either *pooling* layers or strided convolutions. While strided convolutions have the benefit of improved gradient propagation, pooling can have advantages for smoothness when increasing the spatial resolution (i.e., "de-pooling" operations) [65]. Stacks of convolutions, in conjunction with changes of the spatial size of the input data by factors of two, are common building blocks of many modern NN architectures. Such convolutions acting on different spatial scales have significant benefits over fully connected layers, as they can lead to vastly reduced weight numbers and correspondingly well-regularized convolutional kernels. Thus, the resulting smaller networks are easier to train, and usually also have reduced requirements for the amounts of training data that are needed to reach convergence.

IV. Method

In the following, we describe our methodology for the deep-leaning-based inference of RANS solutions.

A. Data Generation

To generate ground truth data for training, we compute the velocity and pressure distributions of flows around airfoils. We consider a space of solutions with a range of Reynolds numbers $Re = [0.5, 5]$ million, incompressible flow, and angles of attack in the range of ± 22.5 deg. We obtained 1505 different airfoil shapes from the UIUC database [66], which were used to generate input data in conjunction with randomly sampled freestream conditions from the range described above. The RANS simulations make use of the widely used SA [62] one equation turbulence model, and solutions are calculated with the open source code *OpenFOAM*. Here we employ a body-fitted triangle mesh, with refinement near the airfoil. At the airfoil surface, the triangle mesh has an average edge length of $1/200$ to accurately resolve the boundary layer of the flow. The discretization is coarsened to an edge length of $1/16$ near the domain boundary, which has an overall size of 8×8 units. The airfoil has a length of 1 unit. Typical examples from the training data set are shown in Fig. 2. Note that in this figure each component and data set are normalized independently for visualization. Thus the images in Fig. 2 show the structure of the solutions rather than the range of values across different solutions.

Although typical RANS solvers, such as the one from OpenFOAM, require large distances for the domain boundaries in order to reduce their negative impact on the solutions around the airfoil, we can focus on a smaller region of 2×2 units around the airfoil for the deep learning task. As our model directly learns from reference data sets that were generated with a large domain boundary distance, we do not need to reproduce solution in the whole space where the solution was computed, but rather can focus on the region in the vicinity of the airfoil without impairing the solution.

To facilitate NN architectures with convolutional layers, we resample this region with a Cartesian 128^2 grid to obtain the ground truth pressure and velocity data sets, as shown in Fig. 3. The resampling is performed with a linear weighted interpolation of cell-centered values with a spacing of $1/64$ units in OpenFOAM. As for the domain boundaries, we only need to ensure that the original solution was produced with a sufficient resolution to resolve the boundary layer. As the solution is smooth, we can later on sample it

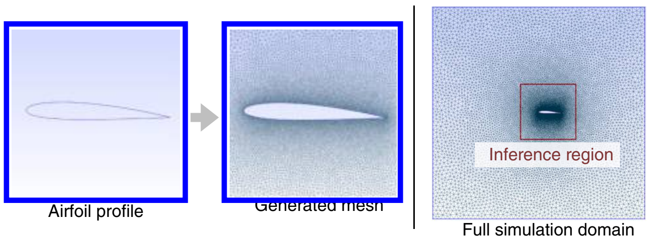


Fig. 3 An example UIUC database entry with the corresponding simulation mesh. The region around the airfoil considered for CNN inference is highlighted with a red square.

with a reduced resolution. Both properties, the reduced spatial extent of the deep learning region and the relaxed requirements for the discretization, highlight advantages of deep learning methods over traditional solvers in our setting.

We randomly choose an airfoil, Reynolds number, and angle of attack from the parameter ranges described above and compute the corresponding RANS solution to obtain data sets for the learning task. This set of samples is split into two parts: a larger fraction that is used for training, and a remainder that is only used for evaluating the current state of a model (the *validation* set). Details of the respective data set sizes are given in the Appendix, and we typically use an 80 to 20% split for training and validation sets, respectively. The validation set allows for an unbiased evaluation of the quality of the trained model during training, for example, to detect overfitting. To later on evaluate the capabilities of the trained models with respect to generalization, we use an additional set of 30 airfoil shapes that were not used for training, to generate a *test data set* with 90 samples (using the same range of Reynolds numbers and angles of attack as described above).

B. Preprocessing

As the resulting solutions of the RANS simulations have a size of $128^2 \times 3$, we use CNN architectures with inputs of the same size. The solutions globally depend on all boundary conditions. Accordingly, the architecture of the network ensures that this information is readily available spatially and throughout the different layers.

Thus freestream conditions and the airfoil shape are encoded in a $128^2 \times 3$ grid of values. As knowledge about the targeted Reynolds number is required to compute the desired output, we encode the Reynolds number in terms of differently scaled freestream velocity vectors. That is, for the smallest Reynolds numbers the freestream velocity has a magnitude of 0.1, whereas the largest ones are indicated by a magnitude of 1. The first of the three input channels contains a [0, 1] mask ϕ for the airfoil shape, 0 being outside and 1 inside. The next two channels of the input contain x and y velocity components, $\boldsymbol{v}_i = (v_{i,x}, v_{i,y})$, respectively. Both velocity channels are initialized to the x and y component of the freestream conditions, respectively, with a zero velocity inside of the airfoil shape. Note that the inputs contain highly redundant information, they are essentially constant, and we likewise, redundantly, encode the airfoil shape in all three input fields.

The output data sets for supervised training have the same size of $128^2 \times 3$. Here, the first channel contains pressure p, whereas the next two channels contain x and y velocity of the RANS solution, $\boldsymbol{v}_o = (v_{o,x}, v_{o,y})$.

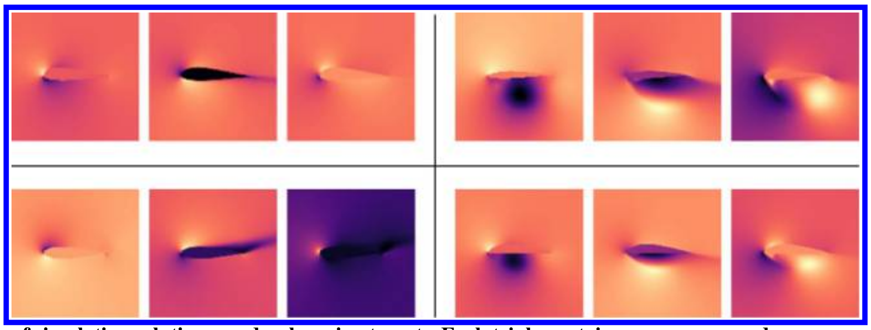


Fig. 2 A selection of simulation solutions used as learning targets. Each triple contains pressure, *x*, and *y* components of the velocity.

Although the simulation data could be used for training in this form, we describe two further data preprocessing steps that we will evaluate in terms of their influence on the learned performance below. First we can normalize all involved quantities with respect to the magnitude of the freestream velocity, that is, make them dimensionless. Thus we consider $\tilde{\boldsymbol{v}}_o = \boldsymbol{v}_o/|\boldsymbol{v}_i|$, and $\tilde{p}_o = p_o/|\boldsymbol{v}_i|^2$. Especially the latter is important to remove the quadratic scaling of the pressure values from the target data. This flattens the space of solutions and simplifies the task for the NN later on.

In addition, only the pressure gradient is typically needed to compute the RANS solutions. Thus, as a second preprocessing variant we can additionally remove the mean pressure from each solution and define $\hat{p}_o = \tilde{p}_o - p_{\text{mean}}$, with the pressure mean $p_{\text{mean}} = \sum_i p_i/n$, where n denotes the number of individual pressure samples p_i. Without this removal of the mean values, the pressure targets represent an ill-posed learning goal as the random pressure offsets in the solutions are not correlated with the inputs.

As a last step, irrespective of which preprocessing method was used, each channel is normalized to the $[-1, 1]$ range in order to minimize errors from limited numerical precision during the training phase. We use the maximum absolute value for each quantity computed over the entire training data set to normalize the data. Both boundary condition inputs and ground truth solutions, that is, output targets, are normalized in the same way. Note that the velocity is not offset, but only scaled by a global per component factor, even when using $\hat{p}_o$.

C. Neural Network Architecture

Our NN model is based on the U-Net architecture [44], which is widely used for tasks such as image translation. The network has the typical bowtie structure, translating spatial information into extracted features with convolutional layers. In addition, skip-connections from in- to output feature channels are introduced to ensure that this information is available in the outputs layers for inferring the solution. We have experimented with a variety of architectures with different amounts of skip connections [45–47] and found that the U-net yields a very good quality with relatively low memory requirements. Hence, we will focus on the most successful variant, a modified U-net architecture in the following.

This U-Net is a special case of an encoder–decoder architecture. In the encoding part, the image size is progressively downsampled by a factor of 2 with strided convolutions. The allows the network to extract increasingly large-scale and abstract information in the growing number of feature channels. The decoding part of the network mirrors this behavior, and increases the spatial resolution with average-depooling layers and reduces the number of feature layers. The skip connections concatenate all channels from the encoding branch to the corresponding branch of the decoding part, effectively doubling the amount of channels for each decoding block. These skip connections help the network to consider low-level input information during the reconstruction of the solution in the decoding layers. Each section of the network consists of a convolutional layer, a batch normalization layer, in addition to a nonlinear activation function.

For our standard U-net with 7.7 m weights, we use 7 convolutional blocks to turn the $128^2 \times 3$ input into a single data point with 512 features, typically using convolutional kernels of size 4^2 (only the inner three layers of the encoder use 2^2 kernels; see Appendix A). As activation functions we use leaky ReLU functions with a slope of 0.2 in the encoding layers, and regular ReLU activations in the decoding layers. The decoder part uses another 7 symmetric layers to reconstruct the target function with the desired dimensionality of $128^2 \times 3$.

Although it seems wasteful to repeat the freestream conditions almost 128^2 times, that is, over the whole domain outside of the airfoil, this setup is very beneficial for the NN. We know that the solution everywhere depends on the boundary conditions, and while the network would eventually learn this and propagate the necessary information via the convolutional bowtie structure, it is beneficial for the training process to ensure that this information is available everywhere right from the start. This motivates the architecture with redundant boundary condition information and skip connections.

D. Supervised Training

We train our architecture with the *Adam* optimizer [67], using 80,000 iterations unless otherwise noted. Our implementation is based on the *PyTorch* (from https://pytorch.org) deep learning framework. Because of the strictly supervised setting of our learning setup, we use a simple L_1 loss $L = |\boldsymbol{y} - \boldsymbol{a}|$, with $\boldsymbol{a}$ being the output of the CNN. Here, an L_2 loss could likewise be used and yields very similar results, but we found that L_1 yields slight improvements. In both cases, a supervision for all cells of the inferred outputs in terms of a direct vector norm is important for stable training runs. The number of iterations was chosen such that the training runs converge to their final inference accuracies across all changes of hyperparameters and network architectures. Hence, to ensure that the different training runs below can be compared, we train all networks with the same number of training iterations.

Because of the potentially large number of local minima and the stochasticity of the training process, individual runs can yield significantly different results due to effects such as nondeterministic GPU calculations and/or different random seeds. Although runs can be made deterministic, slight changes of the training data or their order can lead to similar differences in the trained models. Thus, in the following we present network performances across multiple training runs. For practical applications, a single, best performing model could be selected from such a collection of runs via cross-validation. The graphs below, show the mean and standard error of the mean for five runs with otherwise identical settings apart from different random seeds. Hence, the standard errors indicate the variance in result quality that can be expected for a selected training modality.

First, we illustrate the importance of proper data normalization. As outlined above we can train models either (A) with the pressure and velocity data exactly as they arise in the model equations, that is, $(\boldsymbol{v}_o, p_o)$, or (B) normalize the data by velocity magnitude $(\tilde{\boldsymbol{v}}_o, \tilde{p}_o)$. Lastly, we can remove the pressure null space and train models with $(\tilde{\boldsymbol{v}}_o, \hat{p}_o)$ as target data (C). Not surprisingly, this makes a huge difference. For comparing the different variants, we denormalize the data for (B) and (C), and then compute the averaged, absolute error w.r.t. ground truth pressure and velocity values for 400 randomly selected data sets. While variant (A) exhibits a very significant average error of 291.34, the data variant (B) has an average error of only 0.0566, whereas (C) reduces this by another factor of ca. 4 to 0.0136. An airfoil configuration that shows an example of how these errors manifest themselves in the inferred solutions can be found in Fig. 4.

Thus, in practice it is crucial to understand the data that the model should learn, and simplify the space of solutions as much as possible, for example, by making use of the dimensionless quantities for fluid flow. Note that for the three models discussed above we have already used the training setup that we will explain in more detail in the following paragraphs. From now on, all models will be trained only with fully normalized data, that is, case (C) above.

We have also experimented with adversarial training, and although others have noticed improvements [40], we were not successful with GANs during our tests. Even though individual runs yielded good results, this was usually caused by suboptimal settings for the corresponding supervised training runs. In a controlled setting such as ours, where we can densely sample the parameter space, we found that generating more training data, rather than switching to a more costly adversarial training, is typically a preferred way to improve the results.

1. *Basic Parameters*

To establish a training methodology, we first evaluate several basic training parameters, namely, learning rate, learning rate decay, and normalization of the input data. In the following we will evaluate these parameters for a standard network size with a training data set of medium size (details given in Appendix B).

One of the most crucial parameters for deep learning is the learning rate of the optimizer. The learning rate scales the step size the optimizer takes to update the weights of an NN based on a gradient computed from one mini-batch of data. As the energy landscapes spanned by typical deep NNs are often nonlinear functions with large numbers of minima and saddle-points, small learning rates are not

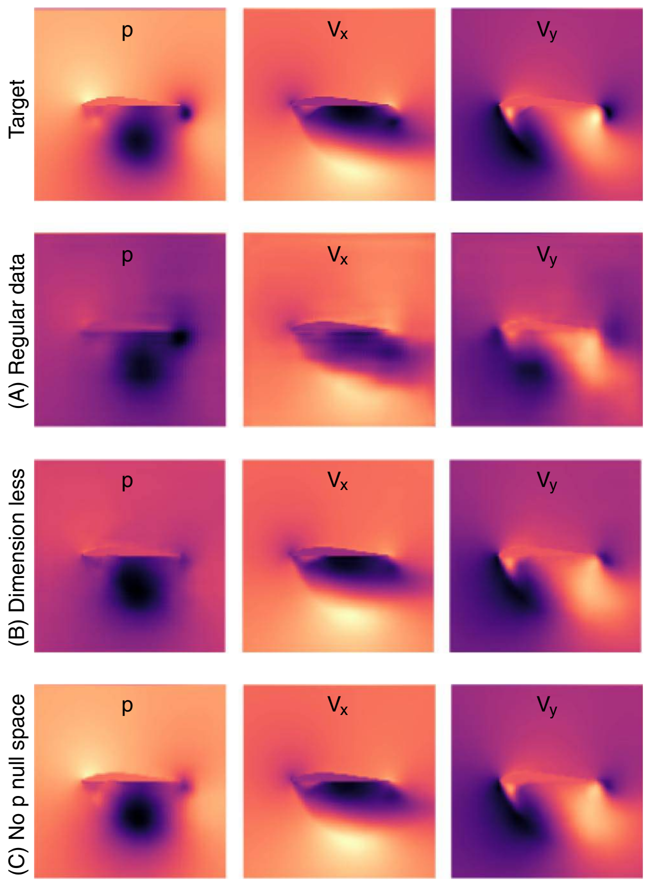


Fig. 4 Ground truth target, and three different data preprocessing variants: using the data directly (A), velocity normalization (B), and removing the pressure null space (C).

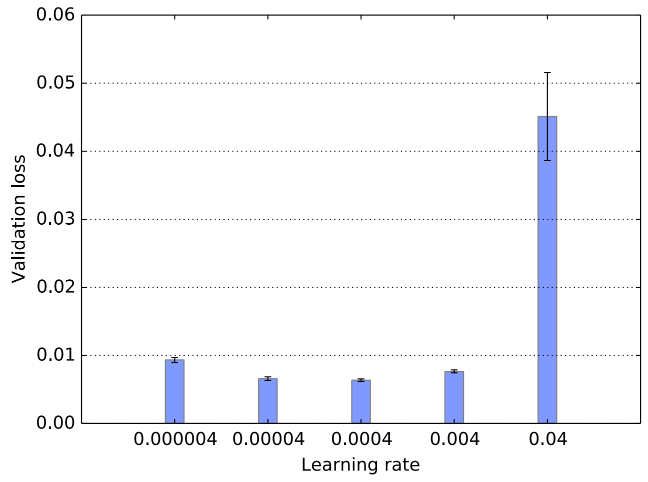
necessarily ideal, as the optimization might get stuck in undesirable states, whereas overly large ones can easily prevent convergence. Figure 5 illustrates this for our setting. The largest learning rate clearly overshoots and has trouble converging. In our case the range of $4 \cdot 10^{-3}$ to $4 \cdot 10^{-4}$ yields good convergence.

In addition, we found that learning rate decay, that is, decreasing the learning rate during training, helps to stabilize the results, and reduce variance in performance. Although the influence is not huge when the other parameters are chosen well, we decrease the learning rate to 10% of its initial value over the course of the second half of the training iterations for the training runs below. A comparison of four different settings each with and without learning rate decay is shown at the bottom of Fig. 5.

2. *Regularization and Hyperparameters*

To prevent overfitting, that is, the reproduction of single input–output pairs rather than a smooth reproduction of the function that should be approximated by the model, regularization is an important topic for deep learning methods. The most common regularization techniques are dropout, data augmentation, and early stopping [64]. In addition, the loss function can be modified, typically with additional terms that minimize parts of the outputs or even the NN weights (*weight decay*) in terms of L_1 or L_2 norms. Although this regularization is especially important for fully connected NNs [35,36], we focus on CNNs in our study, which are inherently less prone to overfitting as they are applied to all spatial locations of an input, and typically receive a large variety of data configurations even from a single input to the NN.

In addition to the techniques outlined above, overfitting can also be avoided by ensuring that enough training data are available. Even though this is not always possible or practical, a key advantage of machine learning in the context of PDEs is that reliable training data can be generated in large amounts given enough computational resources. We will investigate the influence of varying amounts of



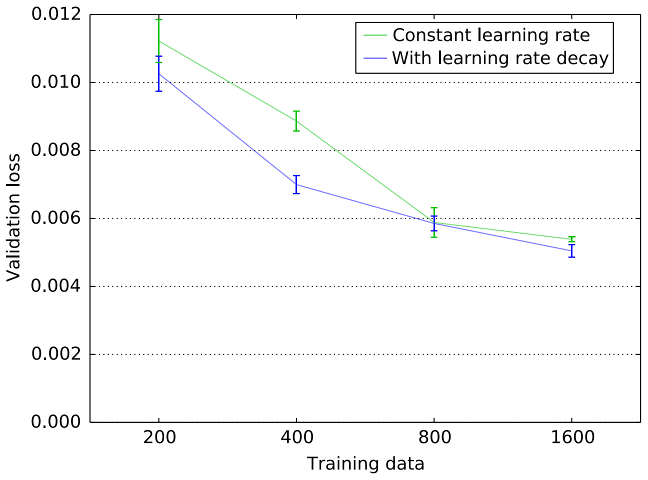


Fig. 5 Top: Varied learning rates shown in terms of validation loss for an otherwise constant learning problem. Bottom: Validation loss for a series of models with and without learning rate decay (amount of training data is varied). The decay helps to reduce variance and performance especially when less training data are used.

training data for our models in more detail below. Although we found a very slight amount of dropout to be preferable (see Appendix A), we will not use any other regularization methods below. For example, we found that our networks converged to stable levels in terms of training and validation losses, and hence do not employ early stopping in order to ensure that all networks below were trained with the same number of iterations. It is also worth noting that *data augmentation* is difficult in our context, as each solution is unique for a given input configuration. In the following, we instead focus on the influence of varying amounts of precomputed training data on the generalizing capabilities of a trained network.

Note that due to the complexity of typical learning tasks for NNs it is nontrivial to find the best parameters for training. Typically, it would be ideal to perform broad hyperparameter searches for each of the different hyperparameters to evaluate their effect on inference accuracy. However, as differences of these searches were found to be relatively small in our case, the training parameters will be kept constant: the following training runs use a learning rate of 0.0004, a batch size of 10, and learning rate decay.

3. *Accuracy*

Having established a stable training setup, we can now investigate the accuracy of our network in more detail. Based on a database of 26,722 target solutions generated as described above, we have measured how the amount of available training data influences accuracy for validation data as well as generalization accuracy for the test data set. In addition, we measure how accuracy scales with the number of weights, that is, degrees of freedom in the NN.

For the following training runs, the amount of training data is scaled from 100 samples to 12,800 samples in factors of two, and we vary the number of weights by scaling the number of feature maps in the convolutional layers of our network. As the size of the kernel tensor of a convolutional layer scales with the number of input channels times number of output channels, a 2× increase of channels leads to a roughly fourfold increase in overall weights (biases change linearly). The corresponding accuracy graphs for five different network sizes can be seen in Fig. 6. As outlined above, five models with different random seeds (and correspondingly different sets of training data) were trained for each of the data points. As before, standard errors of the mean for the five runs are shown with error bars in the graphs. The different networks have 122,979; 487,107; 1,938,819; 7,736,067; and 30,905,859 weights, respectively. The validation loss in Fig. 6 top shows how the models with little amounts of training data exhibit larger errors, and vary very significantly in terms of performance. The behavior stabilizes with larger amounts of data being available for training, and the models saturate in terms of inference accuracy at different levels that correspond to their weight numbers. Comparing the curves for the 122 k and 30.9 m models, the former exhibits a flatter curve with lower errors in the beginning (due to inherent regularization from the smaller number of weights), and larger errors at the end. The 30.9 m instead more strongly overfits to the data initially and yields a better performance when enough data are available. In this case the mean error for the latter model is 0.0033 compared with 0.0063 for the 122 k model.

The graphs also show how the different models start to saturate in terms of loss reduction once a certain amount of training data is available. For the smaller models this starts to show around 1000 samples, whereas the trends of the larger models indicate that they could benefit from even more training data. The loss curves indicate that despite the 4× increase in weights for the different networks, roughly doubling the amount of training data is sufficient to reach a similar saturation point.

The bottom graph of Fig. 6 shows the performance of the different models for a test set of 30 airfoils that were not seen during training. Freestream velocities were randomly sampled from the same distribution used for training data generation (see Sec. IV) to produce 90 test data sets. We use these data to measure how well the trained models can generalize to new airfoil shapes. Instead of the L_1 loss that was used for the validation data, this graph shows the mean relative error of the normalized quantities (the L_1 loss behavior is in line with the relative errors shown here). The relative error is computed with $e_f = 1/n \sum_{\text{in}} |\tilde{f} - f|/|f|$, with n being the number of samples (128^2 in our case), f the function under consideration (e.g., pressure), and $\tilde{f}$ the approximation of the NN. We found the average relative error for all inferred fields, that is, $e_{\text{avg}} = (e_p + e_{v_{o,x}} + e_{v_{o,y}})/3$, to be a good metric to evaluate the models, as it takes all outputs into account and directly yields an error percentage for the accuracy of the inferred solutions. In addition, it sheds light on the relative accuracy of the inferred solutions, whereas the L_1 metric yields estimates of the averaged differences. Hence, both metrics are important for evaluating the overall accuracy of the trained models.

For this test set, the curves in the bottom graph of Fig. 6 exhibit a similar fall-off with reduced errors for larger amounts of training data, but the difference between the different model sizes is significantly smaller. With the largest amount of training data (12.8 k samples), the difference in test error between smallest and largest model is 0.033 versus 0.026. Thus, the latter network achieves an average relative error of 2.6% across all three output channels. Because of the differences between velocity and pressure functions, this error is not evenly distributed. Rather, the model is trained for reducing L_1 differences across all three output quantities, relative errors of 2.15% for the x velocity channel, 2.6% for y, and 14.76% for pressure values. Especially the relatively large amount of small pressure values with fewer large spikes in the harmonic functions leads to increased relative errors for the pressure channel. If necessary, this could be alleviated by changing the loss function, but as the goal of this study is to consider generic CNN performance, we will continue to use L_1 loss in the following.

Also, it is visible in Fig. 6 that the three largest model sizes yield a very similar performance. Although the models improve in terms of capturing the space of training data, as visible from the validation loss

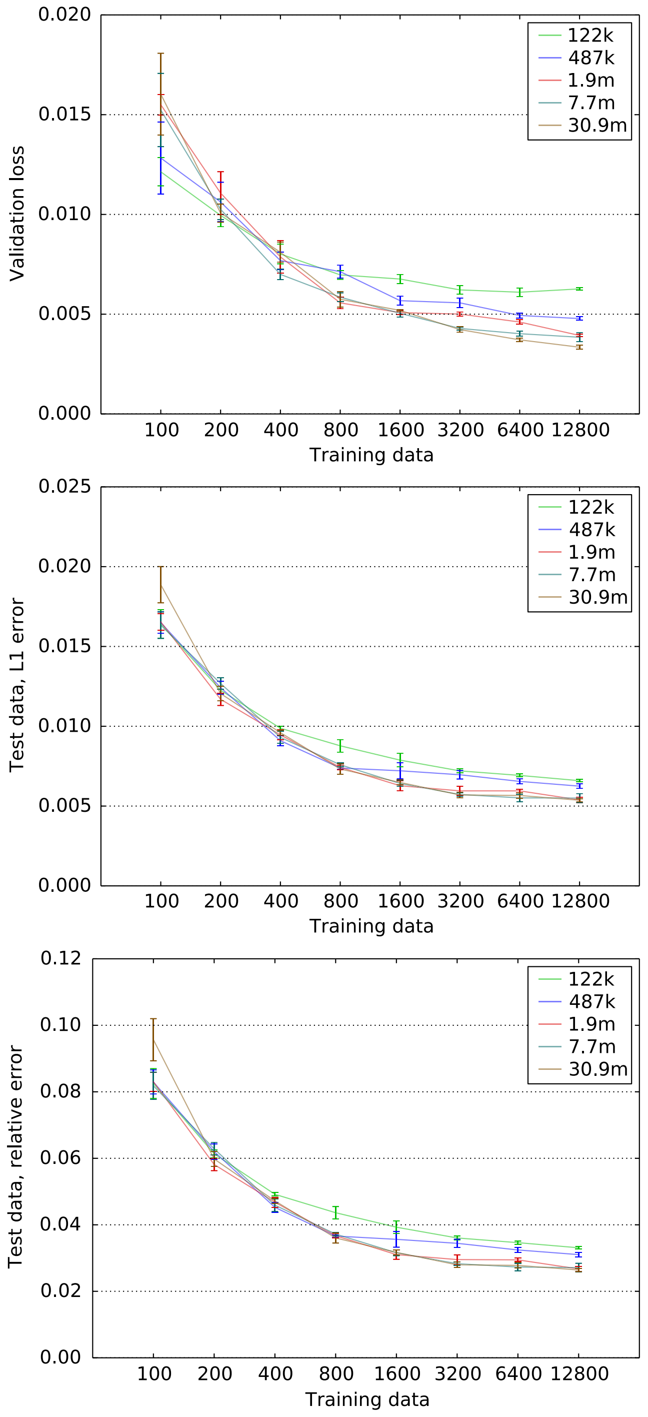


Fig. 6 Validation (L1) and testing accuracy (L1 and relative) for different model sizes and training data amounts.

at the top of Fig. 6, this does not directly translate into an improved generalization. The additional training data in this case do not yield new information for the unseen shapes. An example data set with inferred solutions is visualized in Fig. 7. Note that the relatively small numeric changes of the overall test error lead to significant differences in the solutions.

To investigate the generalization behavior in more detail, we have prepared an augmented data set, where we have sheared the airfoil shapes by ±15 deg along a centered x axis to enlarge the space of shapes seen by the networks. The corresponding relative error graphs for a model with 7.7 m weights are shown in Fig. 8. Here we compare three variants, a model trained only with regular data (this is identical to Fig. 6), models purely trained with the sheared data set, and a set of models trained with 50% of the regular data and 50% of the sheared data. We will refer to these data sets as *regular*, *sheared*, and *mixed* in

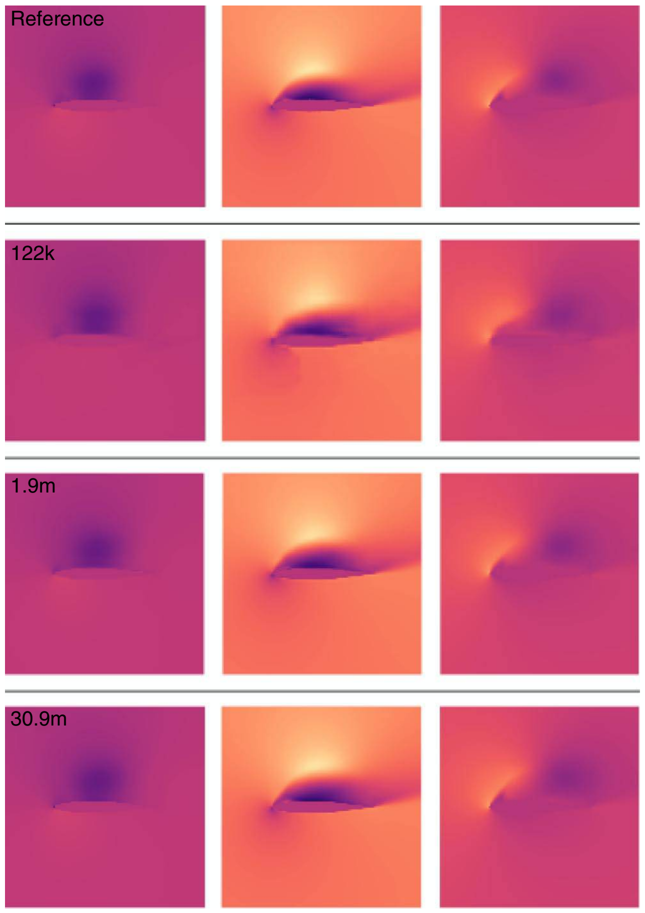


Fig. 7 An example result from the test data set, with the reference, and inferred solutions by three different model sizes (122 k, 1.9 m, and 30.9 m).

the following. It is apparent that both the sheared and mixed data sets yield a larger validation error for large training data amounts (Fig. 8 top). This is not surprising, as the introduction of the sheared data leads to an enlarged space of solutions and hence also a more difficult learning task. Figure 8 bottom shows how the trained models perform on the same test data set that was used to evaluate Fig. 6. It is visible that despite the enlarged space of shapes seen during training, the models do not perform better on the test data set. Rather, the performance decreases when only using the sheared data (blue line in Fig. 8 bottom).

This picture changes when using a larger model. Training the 30.9 m weight model with the mixed data set leads to improvements in performance when there is enough data, especially for the run with 25,600 training samples in Fig. 9. In this case, the large model outperforms the regular data model with an average relative error of 2.77% and yields an error of 2.35%. Training the 30.9 m model with 51 k of mixed samples slightly improves the performance to 2.32%. Hence, the generalization performance depends not only on type and amount of training data, but also on the representative capacities of the chosen CNN architecture.

4. *Performance*

The central motivation for deep learning in the context of physics simulations is arguably performance. In our case, evaluating the trained 30.9 m model for a single data point on an NVidia *GTX 1080* GPU takes 5.53 ms (including data transfer to the GPU). This runtime, like all following ones, is averaged over multiple runs. The network evaluation itself, that is, without GPU overhead, takes 2.10 ms. The runtime per solution can be reduced significantly when evaluating multiple solutions at once; for example, for a batch size of 8, the evaluation time rises only slightly to 2.15 ms. In contrast, computing the solution with OpenFOAM requires 40.4 s when accuracy is adjusted to match the network outputs. The runs for training data generation took 71.9 s.

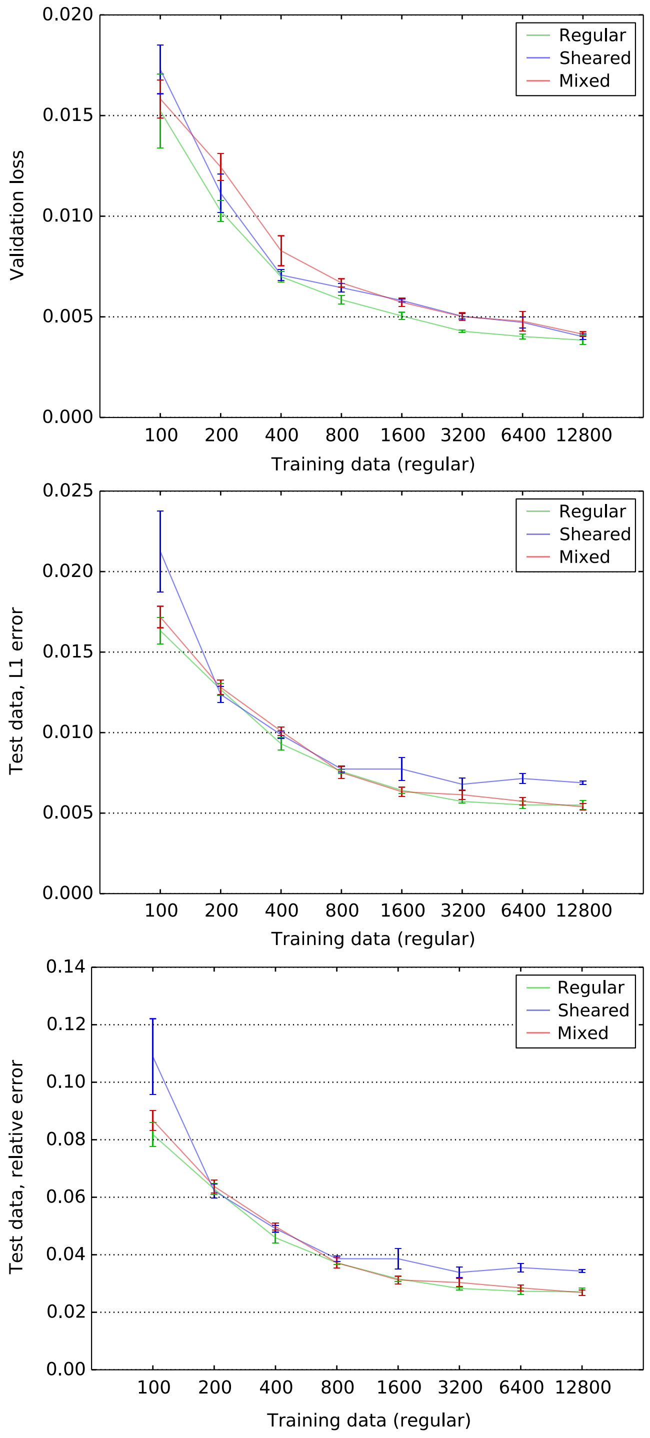


Fig. 8 A comparison of validation loss (L1) and test errors (L1 and relative) for different data sets: regular airfoils, an augmented set of sheared airfoils, and a mixed set of data (50% regular and 50% sheared). All for a model with 7.7 m weights.

Although it is of course problematic to compare implementations as different as the two at hand, it still yields a realistic baseline of the performance we can expect from publicly available open source solvers. OpenFOAM clearly leaves significant room for performance; for example, its solver is currently single-threaded. This likewise holds on the deep learning side: The fast execution time of the discussed architectures can be achieved “out of the box” with PyTorch, and based on future hardware developments such as the NVidia *RTX* series, we expect this performance to improve significantly even without any changes to the trained model itself. Thus, based on the current state of OpenFOAM and PyTorch, our models yield a speed up factor of ca. 1000×.

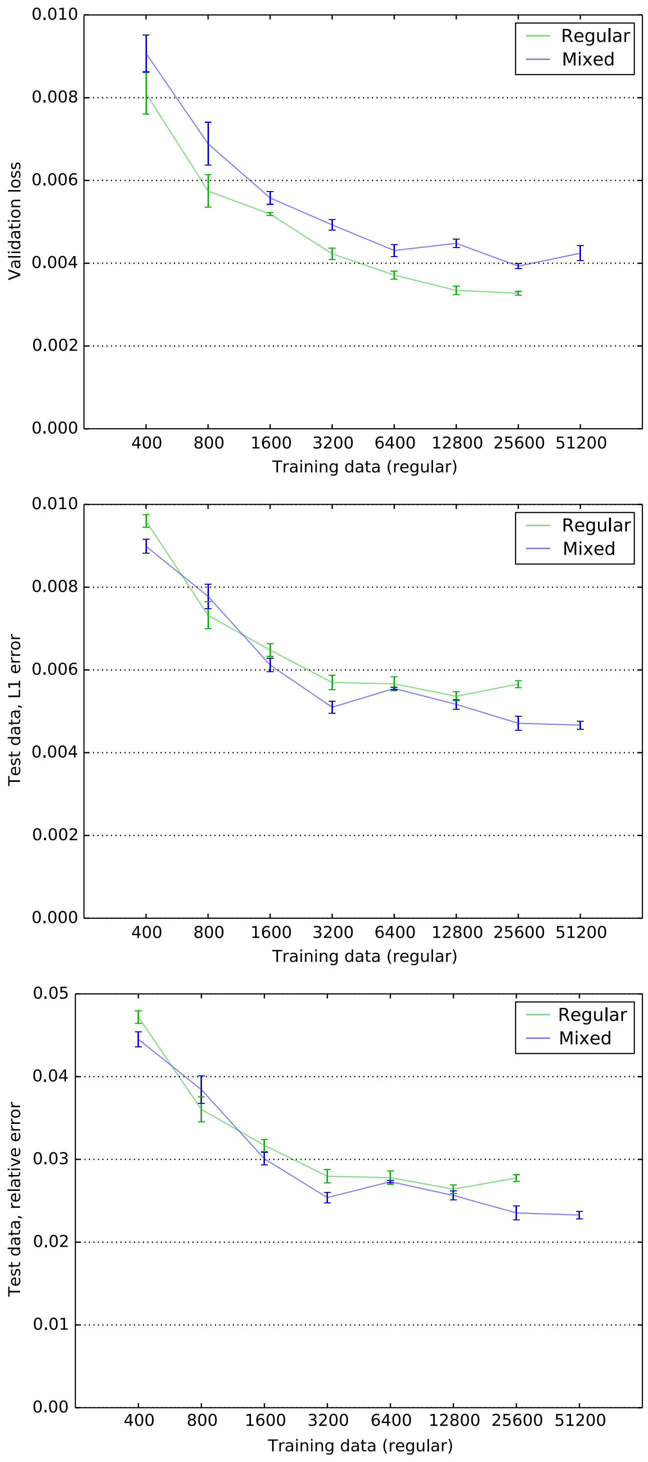


Fig. 9 **Validation loss (L1) and test error (L1 and relative) for a large model (30.9 m weights) trained with regular and mixed airfoil data. Compared with Fig. 8, this model slightly improves, especially for the runs with 25.6 k and 51.2 k data samples.**

The timings of training runs for the models discussed above vary with respect to the amount of data and model size, but start with 26 min for the 122 k models, up to 147 min for the 30.9 m models.

5. Discussion

Overall, our best models yield a very good accuracy of less than 3% relative error. However, it is naturally an important question how this error can be further reduced. Based on our tests, this will require substantially larger training data sets and CNN models. It also becomes apparent from Fig. 6 that simply increasing model and training data size will not scale to arbitrary accuracies. Rather, this

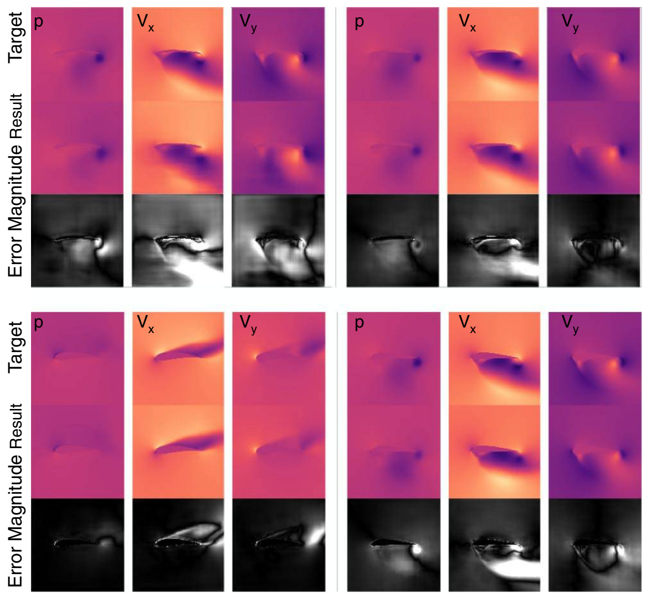


Fig. 10 **Inference test results with particularly high errors. Each triple contains, from left to right, $\hat{p}_o$, $\tilde{v}_{o,x}$, $\tilde{v}_{o,y}$; the model results are shown below. The bottom rows show error magnitudes.**

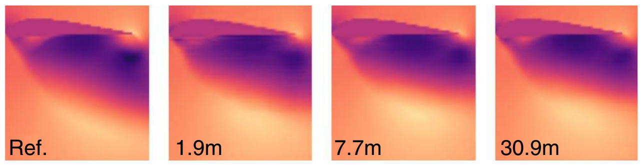


Fig. 11 **A detail of *x* velocity, with reference, and three different inferred solutions shown left to right. The solutions of the larger networks (left) are noticeably sharper, for example, below the airfoil tip.**

points toward the need to investigate and develop different approaches and network architectures.

In addition, it is interesting to investigate how systematic the model errors are. Figure 10 shows a selection of inferred results from the 7.7 m model trained with 12 k data sets. All results are taken from the test data set. As can be seen in the error magnitude visualizations in the bottom row of Fig. 10, the model is not completely off for any of the cases. Rather, errors typically manifest themselves as shifts in the inferred shapes of the wakes behind the airfoil. We also noticed that while they change w.r.t. details in the solution, the error typically remains large for most of the difficult cases of the test data set throughout the different runs. Most likely, this is caused by a lack of new information the models can extract from the training data sets that were used in the study. Here, it would also be interesting to consider an even larger test data set to investigate generalization in more detail.

Finally, it is worth pointing out that despite the stagnated error measurements of the larger models in Fig. 6, the results consistently improve, especially regarding their sharpness. This is illustrated with a zoom in on a representative case for *x* velocity in Fig. 11, and we have noticed this behavior across the board for all channels and many data sets. As can be seen there, the sharpness of the inferred function increases, especially when comparing the 1.9 m and 30.9 m models, which exhibit the same performance in Fig. 6. This behavior can be partially attributed to these improvements being small in terms of scale. However, as it is noticeable consistently, it also points to inherent limitations of direct vector norms as loss functions.

V. Conclusions

This is a first study of the accuracy of deep learning for the inference of RANS solutions for airfoils. Although the results of our study are by no means guaranteed to directly carry over to other problems due to the inherent differences of solution spaces for different physical problems, our results can nonetheless serve as a good starting point with respect to general methodology, data handling, and training procedures.

It is our hope that they would provide a starting point for researchers and help to overcome skepticism from perceived lacks of theoretical results. Flow simulations are a good example of a field that has made tremendous steps forward despite unanswered questions regarding theory: although it is unknown whether a finite time singularity for the Navier–Stokes equations exists, this luckily has not stopped research in the field. Likewise, there is huge potential for deep learning in the CFD context despite the open questions regarding theory.

In addition this paper has outlined a simulation and training setup that is, on the one hand, relatively simple, but nonetheless offers a large amount of complexity for machine learning algorithms. It also illustrates that it is important to formulate the problem such that the relationship between input and output quantities is as simple as possible. In our setting, the dimensionless formulation of the problem leads to significantly improved results without changes to the deep learning components. The supervised inference of RANS solutions with a U-net structure provides a good point of entry for CFD researchers to experiment with deep learning algorithms, as well as a benchmark case for the evaluation of novel learning methods for fluids and related physics problems.

There are numerous avenues for future work in the area of physics-based deep learning, for example, to employ trained flow models in the context of inverse problems. The high performance and differentiability of a CNN model has the potential to provide a good basis for problems such as flow control and shape optimization. The NN-based representation ensures that gradients w.r.t. inferred solutions are available and can typically be computed very efficiently via GPUs for fast iterations in an optimizer. In this context, our work makes steps toward establishing NN methods for inverse problems related to fluid flows.

Appendix A: Architecture and Training Details

The network is fully convolutional with 14 layers and consists of a series of convolutional *blocks*. All blocks have a similar structure: activation, convolution, batch normalization, and dropout. Instead of transpose convolutions we use a linear upsampling "up()" followed by a regular convolution [65].

Convolutional blocks C below are parameterized by channel factor c, kernel size k, and stride s, where we use cX as short form for $c = X$. Batch normalization is indicated by b below. Activation by ReLU is indicated by r, and l indicates a leaky ReLU [68,69] with a slope of 0.2. Slight dropout with a rate of 0.01 is used for all layers. The different models above use a channel base multiplier 2^{c_i} that is multiplied by c for the individual layers. c_i was 3, 4, 5, 6, and 7 for the 122 k, 487 k, 1.9 m, 7.7 m, and 30.9 m models discussed above. Thus, for example, for $c_i = 6$ a C block with $c8$ has 512 channels. Channelwise concatenation is denoted by "conc()."

The network receives an input l_0 with three channels (as outlined in Sec. IV) and can be summarized as:

$$\begin{aligned}
l_1 &\leftarrow C(l_0, c1k4s2)\\
l_2 &\leftarrow C(l_1, c2k4s2lb)\\
l_3 &\leftarrow C(l_2, c2k4s2lb)\\
l_4 &\leftarrow C(l_3, c4k4s2lb)\\
l_5 &\leftarrow C(l_4, c8k2s1lb)\\
l_6 &\leftarrow C(l_5, c8k2s1lb)\\
l_7 &\leftarrow C(l_6, c8k2s1lb)\\
l_8 &\leftarrow \mathrm{up}(C(l_7, c8k2s1rb))\\
l_9 &\leftarrow \mathrm{up}(C(\mathrm{conc}(l_8, l_6), c16k2s1rb))\\
l_{10} &\leftarrow \mathrm{up}(C(\mathrm{conc}(l_9, l_5), c16k2s1rb))\\
l_{11} &\leftarrow \mathrm{up}(C(\mathrm{conc}(l_{10}, l_4), c8k4s1rb))\\
l_{12} &\leftarrow \mathrm{up}(C(\mathrm{conc}(l_{11}, l_3), c4k4s1rb))\\
l_{13} &\leftarrow \mathrm{up}(C(\mathrm{conc}(l_{12}, l_2), c4k4s1rb))\\
l_{14} &\leftarrow \mathrm{up}(C(\mathrm{conc}(l_{13}, l_1), k4s1r))
\end{aligned}$$

Here l_{14} represents the output of the network, and the corresponding convolution generates 3 output channels. Unless otherwise noted, training runs are performed with $i = 80{,}000$ iterations of the Adam optimizer using $\beta_1 = 0.5$ and $\beta_2 = 0.999$, learning rate $\eta = 0.0004$ with learning rate decay and batch size $b = 10$. Figure 5 used a model with 7.7 m weights, $i = 40$ k iterations, and 8 k training data samples, 75% regular and 25% sheared. Figure 4 used a model with 7.7 m weights with 12.8 k training data samples, 75% regular and 25% sheared.

Appendix B: Training Data

In the following, we also give details of the different training data set sizes used in the training runs above. We start with a minimal size of 100 samples and increase the data set size in factors of 2 up to 12,800. Typically, the total number of samples is split into 80% training data and 20% validation data. However, we found validation sets of several hundred samples to yield stable estimates. Hence, we use an upper limit of 400 as the maximal size of the validation data set. The corresponding number of samples is randomly drawn from a pool of 26,732 precomputed pairs of boundary conditions and flow solutions computed with OpenFOAM. The exact sizes used for the training runs of Figs. 5 and 6 are given in Table B1. The dimensionality of the different data sets is summarized in Table B2.

In addition to this regular data set, we employ a *sheared* data set that contains sheared airfoil profiles (as described above). This data set has an overall size of 27,108 samples and was used for the sheared models in Fig. 8 with the same training set sizes as shown in Table B1. We also trained models with a *mixed* data set, shown in Figs. 8 and 9, that contains half regular and half sheared samples. That is, the five models for $x = 800$ in Fig. 9 employed 320 samples drawn from the regular plus 320 drawn from the sheared data set for training, and 80 regular plus 80 sheared samples for validation. For the mixed data set, we additionally train a large model with a total dataset size of 25,600 and 51,200 (i.e., the bottom two lines of Fig. 1) that is shown in Fig. 9.

Table B1 Different data set sizes used for training runs, together with corresponding splits into training and validation sets

Total dataset size	Training	Validation
100	80	20
200	160	40
400	320	80
800	640	160
1,600	1,280	320
3,200	2,800	400
6,400	6,000	400
12,800	12,400	400
25,600	25,200	400
51,200	50,800	400

Table B2 Dimensionality of airfoil shape database and data sets used for neural network training and evaluation

Quantity	Dimension
Airfoil shapes (training, validation)	1,505
Airfoil shapes (test)	30
RANS solutions (regular)	26,732
RANS solutions (sheared)	27,108

Appendix C: Training Evolution and Dropout

In Fig. C1 we show three examples of training runs of typical model used on the accuracy evaluations above. These graphs show

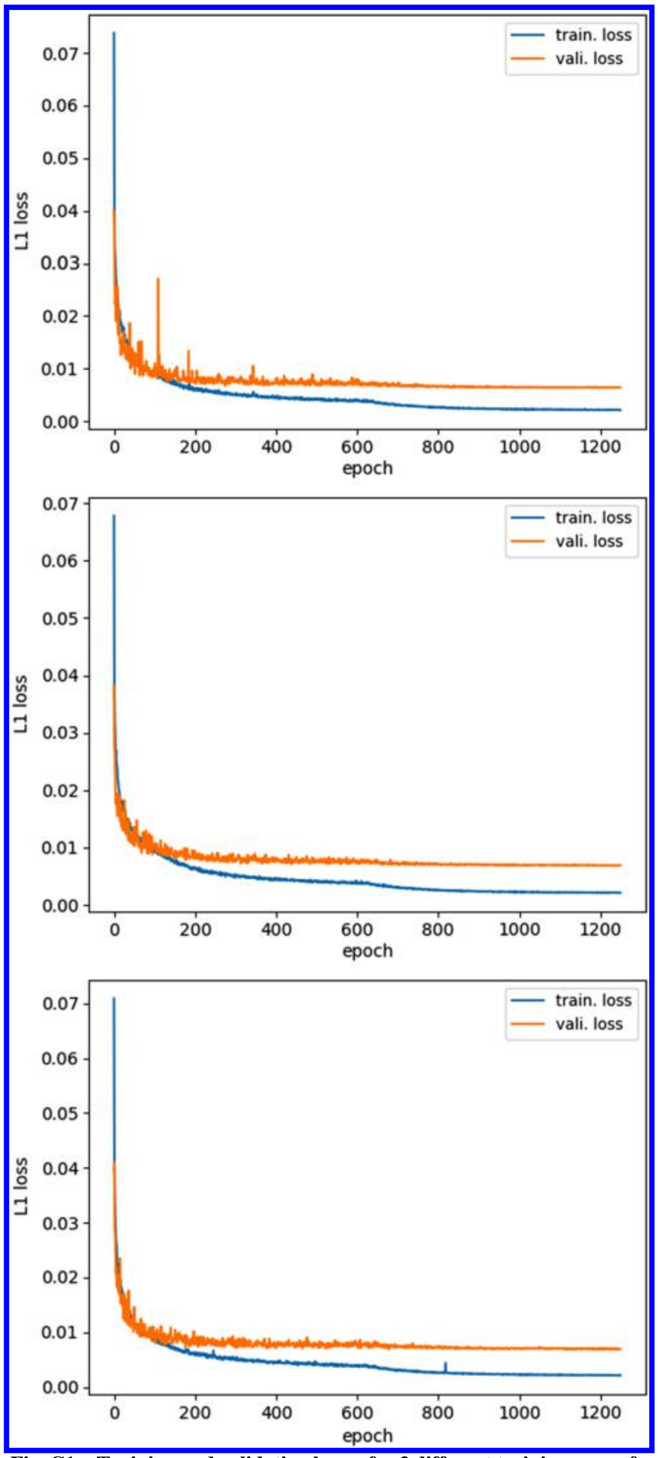


Fig. C1 Training and validation losses for 3 different training runs of a model with 7.7 m weights, and 800 training data samples. For the *x* axis, each epoch indicates the processing of 64 samples.

that the models converge to stable levels of training and validation loss, and do not exhibit overfitting over time. Additionally, the onset of learning rate decay can be seen in the middle of the graph. This noticeably reduces the variance of the learning iterations, and let the training process fine-tune the current state of the model.

Figure C2 evaluates the effect of varying dropout rates. Different amounts of dropout applied at training time are shown along *x* with the resulting test accuracies. It becomes apparent that the effect is relatively small, but overall the accuracy deteriorates with increasing amounts of dropout.

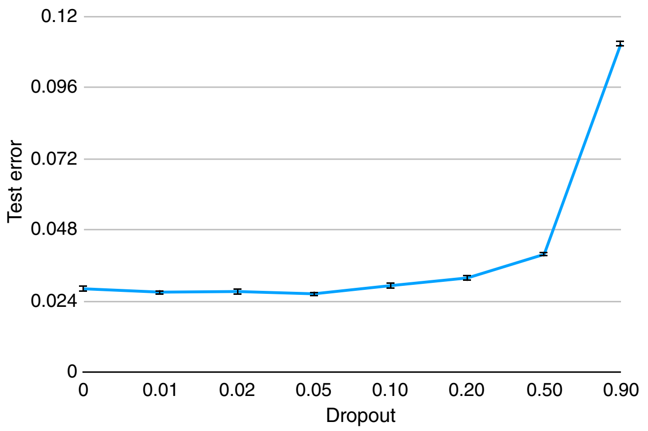


Fig. C2 A comparison of different dropout rates for a model with 7.7 m weights and 8000 training data samples.

Acknowledgments

This work was supported by European Resarch Council (ERC) Starting Grant 637014 (realFlow) and the internship program of the Technical University of Munich. We thank H. Mehrotra and N. Mainali for their help with the deep learning experiments, and the anonymous reviewers for the helpful comments to improve our work.

References

[1] Krizhevsky, A., Sutskever, I., and Hinton, G. E., "Imagenet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105. https://doi.org/10.1145/3065386

[2] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A., "Image-to-Image Translation with Conditional Adversarial Networks," *Proceedings of IEEE Computer Vision and Pattern Recognition*, 2017. https://doi.org/10.1109/cvpr.2017.632

[3] Karras, T., Aila, T., Laine, S., and Lehtinen, J., "Progressive Growing of Gans for Improved Quality, Stability, and Variation," arXiv: 1710.10196, 2017.

[4] Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K., "Accelerating Eulerian Fluid Simulation with Convolutional Networks," *International Conference on Machine Learning*, Vol. 70, 2017, pp. 3424–3433.

[5] Xie, Y., Franz, E., Chu, M., and Thuerey, N., "tempoGAN: A Temporally Coherent, Volumetric GAN for Super-Resolution Fluid Flow," *ACM Transactions on Graphics*, Vol. 37, No. 4, 2018. https://doi.org/10.1145/3197517.3201304

[6] Raissi, M., Perdikaris, P., and Karniadakis, G., "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations," *Journal of Computational Physics*, Vol. 378, Feb. 2019, pp. 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

[7] Durbin, P. A., "Some Recent Developments in Turbulence Closure Modeling," *Annual Review of Fluid Mechanics*, Vol. 50, No. 1, 2018, pp. 77–103. https://doi.org/10.1146/annurev-fluid-122316-045020

[8] Yun, C., Sra, S., and Jadbabaie, A., "A Critical View of Global Optimality in Deep Learning," arXiv:1802.03487, 2018.

[9] Thuerey, N., Mehrotra, H., Mainali, N., Weissenow, K., Prantl, L., and Hu, X., *Deep Flow Prediction*, Technical Univ. of Munich (TUM), 2018, https://github.com/thunil/Deep-Flow-Prediction [retrieved 1 Oct. 2018].

[10] Greenman, R. M., and Roth, K. R., "High-Lift Optimization Design Using Neural Networks on a Multi-Element Airfoil," *Journal of Fluids Engineering*, Vol. 121, No. 2, 1999, pp. 434–440. https://doi.org/10.1115/1.2822228

[11] Schulman, J., Lee, A., Ho, J., and Abbeel, P., "Tracking Deformable Objects with Point Clouds," *2013 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, New York, 2013, pp. 1130–1137. https://doi.org/10.1109/icra.2013.6630714

[12] Zheng, B., Zhao, Y., Joey, C. Y., Ikeuchi, K., and Zhu, S.-C., "Detecting Potential Falling Objects by Inferring Human Action and Natural Disturbance," *2014 IEEE International Conference on Robotics and*

Automation (ICRA), IEEE, New York, 2014, pp. 3417–3424.
https://doi.org/10.1109/icra.2014.6907351

[13] Jia, Z., Gallagher, A. C., Saxena, A., and Chen, T., "3d Reasoning from Blocks to Stability," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 37, No. 5, 2015, pp. 905–918.
https://doi.org/10.1109/TPAMI.2014.2359435

[14] Battaglia, P., Pascanu, R., Lai, M., and Rezende, D. J., "Interaction Networks for Learning About Objects, Relations and Physics," *Advances in Neural Information Processing Systems*, 2016, pp. 4502–4510.

[15] Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A., "Visual Interaction Networks," *Advances in Neural Information Processing Systems*, 2017, pp. 4540–4548.

[16] Chang, M. B., Ullman, T., Torralba, A., and Tenenbaum, J. B., "A Compositional Object-Based Approach to Learning Physical Dynamics," arXiv:1612.00341, 2016.

[17] Yu, R., Zheng, S., Anandkumar, A., and Yue, Y., "Long-Term Forecasting Using Tensor-Train RNNs," arXiv:1711.00073, 2017.

[18] Ehrhardt, S., Monszpart, A., Mitra, N. J., and Vedaldi, A., "Learning A Physical Long-Term Predictor," arXiv:1703.00247, 2017.

[19] Yang, C., Yang, X., and Xiao, X., "Data-Driven Projection Method in Fluid Simulation," *Computer Animation and Virtual Worlds*, Vol. 27, Nos. 3–4, 2016, pp. 415–424.
https://doi.org/10.1002/cav.v27.3-4

[20] Wiewel, S., Becher, M., and Thuerey, N., "Latent-Space Physics: Towards Learning the Temporal Evolution of Fluid Flow," arXiv:1801, 2018.
https://doi.org/10.1111/cgf.13620

[21] Chu, M., and Thuerey, N., "Data-Driven Synthesis of Smoke Flows with CNN-Based Feature Descriptors," *ACM Transactions on Graphics*, Vol. 36, No. 4, 2017, p. 1.
https://doi.org/10.1145/3072959

[22] Um, K., Hu, X., and Thuerey, N., "Splash Modeling with Neural Networks," arXiv:1704.04456, 2017.
https://doi.org/10.1111/cgf.13522

[23] Beck, A., Flad, D., and Munz, C.-D., "Deep Neural Networks for Data-Driven Turbulence Models," *Research Gate Preprint*, arXiv preprint 1806.04482, 2018.

[24] Soize, C., Ghanem, R., Safta, C., Huan, X., Vane, Z. P., Oefelein, J. C., Lacaze, G., and Najm, H. N., "Enhancing Model Predictability for a Scramjet Using Probabilistic Learning on Manifolds," *AIAA Journal*, Vol. 57, No. 1, 2019, pp. 365–378.
https://doi.org/10.2514/1.J057069

[25] Ray, J., Dechant, L., Lefantzi, S., Ling, J., and Arunajatesan, S., "Robust Bayesian Calibration of a k-ε Model for Compressible Jet-in-Crossflow Simulations," *AIAA Journal*, Vol. 56, No. 12, 2018, pp. 4893–4909.
https://doi.org/10.2514/1.J057204

[26] Baqué, P., Remelli, E., Fleuret, F., and Fua, P., "Geodesic Convolutional Shape Optimization," arXiv:1802.04016, 2018.

[27] Umetani, N., and Bickel, B., "Learning Three-Dimensional Flow for Interactive Aerodynamic Design," *ACM Transactions on Graphics*, Vol. 37, No. 4, 2018, pp. 1–10.
https://doi.org/10.1145/3197517

[28] Sekar, V., Zhang, M., Shu, C., and Khoo, B. C., "Inverse Design of Airfoil Using a Deep Convolutional Neural Network," *AIAA Journal*, Vol. 57, No. 3, 2019, pp. 1–11.
https://doi.org/10.2514/1.j057894

[29] Wang, J.-X., Wu, J.-L., and Xiao, H., "Physics-Informed Machine Learning Approach for Reconstructing Reynolds Stress Modeling Discrepancies Based on DNS Data," *Physical Review Fluids*, Vol. 2, No. 3, 2017, Paper 034603.
https://doi.org/10.1103/physrevfluids.2.034603

[30] Wang, J.-X., Huang, J., Duan, L., and Xiao, H., "Prediction of Reynolds Stresses in High-Mach-Number Turbulent Boundary Layers Using Physics-Informed Machine Learning," *Theoretical and Computational Fluid Dynamics*, Vol. 33, No. 1, 2019, pp. 1–19.
https://doi.org/10.1007/s00162-018-0480-2

[31] Tracey, B., Duraisamy, K., and Alonso, J., "Application of Supervised Learning to Quantify Uncertainties in Turbulence and Combustion Modeling," *51st AIAA Aerospace Sciences Meeting*, AIAA Paper 2013-259, 2013.
https://doi.org/10.2514/6.2013-259

[32] Edeling, W., Cinnella, P., Dwight, R. P., and Bijl, H., "Bayesian Estimates of Parameter Variability in the $k-\omega$ Turbulence Model," *Journal of Computational Physics*, Vol. 258, Feb. 2014, pp. 73–94.
https://doi.org/10.1016/j.jcp.2013.10.027

[33] Ling, J., and Templeton, J., "Evaluation of Machine Learning Algorithms for Prediction of Regions of High Reynolds Averaged Navier Stokes Uncertainty," *Physics of Fluids*, Vol. 27, No. 8, 2015, Paper 085103.
https://doi.org/10.1063/1.4927765

[34] Tracey, B. D., Duraisamy, K., and Alonso, J. J., "A Machine Learning Strategy to Assist Turbulence Model Development," *53rd AIAA Aerospace Sciences Meeting*, AIAA Paper 2015-1287, 2015.
https://doi.org/10.2514/6.2015-1287

[35] 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.
https://doi.org/10.1017/jfm.2016.615

[36] Singh, A. P., Medida, S., and Duraisamy, K., "Machine-Learning-Augmented Predictive Modeling of Turbulent Separated Flows over Airfoils," *AIAA Journal*, Vol. 55, No. 7, 2017, pp. 2215–2227.
https://doi.org/10.2514/1.j055595

[37] Prantl, L., Bonev, B., and Thuerey, N., "Generating Liquid Simulations with Deformation-Aware Neural Networks," *ICLR*, 2019.

[38] Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., and Solenthaler, B., "Deep Fluids: A Generative Network for Parameterized Fluid Simulations," arXiv:1806.02071, 2018.
https://doi.org/10.1111/cgf.13619

[39] Yu, J., and Hesthaven, J. S., "Flowfield Reconstruction Method Using Artificial Neural Network," *AIAA Journal*, Vol. 57, No. 2, 2019, pp. 482–498.
https://doi.org/10.2514/1.J057108

[40] Farimani, A. B., Gomes, J., and Pande, V. S., "Deep Learning the Physics of Transport Phenomena," arXiv:1709.02432, 2017.

[41] Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. P., "Data-Driven Discretization: A Method for Systematic Coarse Graining of Partial Differential Equations," arXiv:1808.04930, 2018.

[42] Raissi, M., Yazdani, A., and Karniadakis, G. E., "Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data," arXiv:1808.04327, 2018.

[43] Zhang, Y., Sung, W.-J., and Mavris, D., "Application of Convolutional Neural Network to Predict Airfoil Lift Coefficient," arXiv:1712.10082, 2017.
https://doi.org/10.2514/6.2018-1903

[44] Ronneberger, O., Fischer, P., and Brox, T., "U-Net: Convolutional Networks for Biomedical Image Segmentation," *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2015, pp. 234–241.
https://doi.org/10.1007/978-3-319-24574-4_28

[45] Badrinarayanan, V., Kendall, A., and Cipolla, R., "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," arXiv:1511.00561, 2015.
https://doi.org/10.1109/tpami.2016.2644615

[46] Chaurasia, A., and Culurciello, E., "LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation," arXiv:1707.03718, 2017.
https://doi.org/10.1109/vcip.2017.8305148

[47] Huang, G., Liu, Z., and Weinberger, K. Q., "Densely Connected Convolutional Networks," arXiv:1608.06993, 2016.

[48] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., "Generative Adversarial Nets," *Stat*, Vol. 1050, 2014, p. 10.

[49] Mirza, M., and Osindero, S., "Conditional Generative Adversarial Nets," arXiv:1411.1784, 2014.

[50] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al., "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," arXiv:1609.04802, 2016.
https://doi.org/10.1109/cvpr.2017.19

[51] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A., "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," arXiv:1703.10593, 2017.
https://doi.org/10.1109/iccv.2017.244

[52] Mosser, L., Dubrule, O., and Blunt, M. J., "Reconstruction of Three-Dimensional Porous Media Using Generative Adversarial Neural Networks," arXiv:1704.03225, 2017.
https://doi.org/10.1103/physreve.96.043309

[53] Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L., "Representation Learning and Adversarial Generation of 3D Point Clouds," arXiv:1707.02392, 2017.

[54] Knight, D. D., and Degrez, G., "Shock Wave Boundary Layer Interactions in High Mach Number Flows a Critical Survey of Current Numerical Prediction Capabilities," *Agard Advisory Report*, Vol. 2, 1998, p. 1.

[55] Shur, M., Spalart, P., Strelets, M., and Travin, A., "Detached-Eddy Simulation of an Airfoil at High Angle of Attack," *Engineering*

Turbulence Modelling and Experiments 4, Elsevier, 1999, pp. 669–678.
https://doi.org/10.2514/6.2009-4015

[56] Walters, D. K., and Leylek, J. H., "A New Model for Boundary-Layer Transition Using a Single-Point RANS Approach," *ASME 2002 International Mechanical Engineering Congress and Exposition*, American Soc. of Mechanical Engineers, New York, 2002, pp. 67–79.
https://doi.org/10.1115/imece2002-32740

[57] Bueno-Orovio, A., Castro, C., Palacios, F., and Zuazua, E., "Continuous Adjoint Approach for the Spalart-Allmaras Model in Aerodynamic Optimization," *AIAA Journal*, Vol. 50, No. 3, 2012, pp. 631–646.
https://doi.org/10.2514/1.J051307

[58] Fröhlich, J., and Von Terzi, D., "Hybrid LES/RANS Methods for the Simulation of Turbulent Flows," *Progress in Aerospace Sciences*, Vol. 44, No. 5, 2008, pp. 349–377.
https://doi.org/10.1016/j.paerosci.2008.05.001

[59] Rung, T., Bunge, U., Schatz, M., and Thiele, F., "Restatement of the Spalart-Allmaras Eddy-Viscosity Model in Strain-Adaptive Formulation," *AIAA Journal*, Vol. 41, No. 7, 2003, pp. 1396–1399.
https://doi.org/10.2514/2.2089

[60] Gerolymos, G. A., Sauret, E., and Vallet, I., "Contribution to Single-Point Closure Reynolds-Stress Modelling of Inhomogeneous Flow," *Theoretical and Computational Fluid Dynamics*, Vol. 17, Nos. 5–6, 2004, pp. 407–431.
https://doi.org/10.1007/s00162-004-0109-5

[61] Poroseva, S., and Murman, S. M., "Velocity/Pressure-Gradient Correlations in a FORANS Approach to Turbulence Modeling," *44th AIAA Fluid Dynamics Conference*, AIAA Paper 2014-2207, 2014.
https://doi.org/10.2514/6.2014-2207

[62] Spalart, P., and Allmaras, S., "A One-Equation Turbulence Model for Aerodynamic Flows," *30th Aerospace Sciences Meeting and Exhibit*, AIAA Paper 1992-439, 1992.
https://doi.org/10.2514/6.1992-439

[63] Bishop, C. M., *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer–Verlag, New York, 2006.

[64] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, MIT Press, 2016.
https://doi.org/10.1007/s10710-017-9314-z

[65] Odena, A., Dumoulin, V., and Olah, C., "Deconvolution and Checkerboard Artifacts," *Distill*, 2016, https://distill.pub/2016/deconv-checkerboard [retrieved 1 Oct. 2018].
https://doi.org/10.23915/distill.00003

[66] Selig, M., "UIUC Airfoil Database," Department of Aeronautical and Astronautical Engineering, University of Illinois, Urbana-Champaign, IL, 1996, https://books.google.de/books?id=dWTTjwEACAAJ [retrieved 1 Oct. 2018].

[67] Kingma, D., and Ba, J., "Adam: A Method for Stochastic Optimization," arXiv:1412.6980, 2014.

[68] Maas, A. L., Hannun, A. Y., and Ng, A. Y., "Rectifier Nonlinearities Improve Neural Network Acoustic Models," *Proceedings of ICML*, Vol. 30, No. 1, 2013.

[69] Radford, A., Metz, L., and Chintala, S., "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *Proceedings of ICLR*, 2016.

J. Larsson
Associate Editor

This article has been cited by:

1. Haosu Zhou, Qingfeng Xu, Zhenguo Nie, Nan Li. 2022. A Study on Using Image-Based Machine Learning Methods to Develop Surrogate Models of Stamp Forming Simulations. *Journal of Manufacturing Science and Engineering* **144**:2. . [Crossref]
2. Antonio Alguacil, Michaël Bauerheim, Marc C. Jacob, Stéphane Moreau. 2021. Predicting the propagation of acoustic waves using deep convolutional neural networks. *Journal of Sound and Vibration* **512**, 116285. [Crossref]
3. Masaki Morimoto, Kai Fukami, Kai Zhang, Aditya G. Nair, Koji Fukagata. 2021. Convolutional neural networks for fluid flow analysis: toward effective metamodeling and low dimensionalization. *Theoretical and Computational Fluid Dynamics* **35**:5, 633-658. [Crossref]
4. Quang Tuyen Le, Chinchun Ooi. 2021. Surrogate modeling of fluid dynamics with a multigrid inspired neural network architecture. *Machine Learning with Applications* **812**, 100176. [Crossref]
5. Stefania Fresca, Andrea Manzoni, Luca Dedè, Alfio Quarteroni. 2021. POD-Enhanced Deep Learning-Based Reduced Order Models for the Real-Time Simulation of Cardiac Electrophysiology in the Left Atrium. *Frontiers in Physiology* **12**. . [Crossref]
6. Rohan S. Sharma, Serhat Hosder. Investigation of Mission-Driven Inverse Aircraft Design Space Exploration with Machine Learning. *Journal of Aerospace Information Systems*, ahead of print1-16. [Abstract] [Full Text] [PDF] [PDF Plus]
7. Longyan Wang, Zhaohui Luo, Jian Xu, Wei Luo, Jianping Yuan. 2021. A novel framework for cost-effectively reconstructing the global flow field by super-resolution. *Physics of Fluids* **33**:9, 095105. [Crossref]
8. Antonio Alguacil, Michaël Bauerheim, Marc C. Jacob, Stephane Moreau. A generic deep learning framework for propagation and scattering of acoustic waves in quiescent flows . [Abstract] [PDF] [PDF Plus]
9. Rohit K S S Vuppala, Kursat Kara. A Novel Approach in Realistic Wind Data Generation for The Safe Operation of Small Unmanned Aerial Systems in Urban Environment . [Abstract] [PDF] [PDF Plus]
10. Christian Sabater, Philipp Stürmer, Philipp Bekemeyer. Fast Predictions of Aircraft Aerodynamics using Deep Learning Techniques . [Abstract] [PDF] [PDF Plus]
11. Hamid Reza Attar, Haosu Zhou, Alistair Foster, Nan Li. 2021. Rapid feasibility assessment of components to be formed through hot stamping: A deep learning approach. *Journal of Manufacturing Processes* **68**, 1650-1671. [Crossref]
12. Alvaro Abucide-Armas, Koldo Portal-Porras, Unai Fernandez-Gamiz, Ekaitz Zulueta, Adrian Teso-Fz-Betoño. 2021. A Data Augmentation-Based Technique for Deep Learning Applied to CFD Simulations. *Mathematics* **9**:16, 1843. [Crossref]
13. Koldo Portal-Porras, Unai Fernandez-Gamiz, Ainara Ugarte-Anero, Ekaitz Zulueta, Asier Zulueta. 2021. Alternative Artificial Neural Network Structures for Turbulent Flow Velocity Field Prediction. *Mathematics* **9**:16, 1939. [Crossref]
14. Steven L. Brunton, J. Nathan Kutz, Krithika Manohar, Aleksandr Y. Aravkin, Kristi Morgansen, Jennifer Klemisch, Nicholas Goebel, James Buttrick, Jeffrey Poskin, Adriana W. Blom-Schieber, Thomas Hogan, Darren McDonald. 2021. Data-Driven Aerospace Engineering: Reframing the Industry with Machine Learning. *AIAA Journal* **59**:8, 2820-2847. [Abstract] [Full Text] [PDF] [PDF Plus]
15. Jing Wang, Cheng He, Runze Li, Haixin Chen, Chen Zhai, Miao Zhang. 2021. Flow field prediction of supercritical airfoils via variational autoencoder based deep learning framework. *Physics of Fluids* **33**:8, 086108. [Crossref]
16. Hao Ma, Botao Zhang, Chi Zhang, Oskar J. Haidn. 2021. Generative adversarial networks with physical evaluators for spray simulation of pintle injector. *AIP Advances* **11**:7, 075007. [Crossref]
17. Stefania Fresca, Andrea Manzoni. 2021. Real-Time Simulation of Parameter-Dependent Fluid Flows through Deep Learning-Based Reduced Order Models. *Fluids* **6**:7, 259. [Crossref]
18. Xabier Morales Ferez, Jordi Mill, Kristine Aavild Juhl, Cesar Acebes, Xavier Iriart, Benoit Legghe, Hubert Cochet, Ole De Backer, Rasmus R. Paulsen, Oscar Camara. 2021. Deep Learning Framework for Real-Time Estimation of in-silico Thrombotic Risk Indices in the Left Atrial Appendage. *Frontiers in Physiology* **12**. . [Crossref]
19. Cihat Duru, Hande Alemdar, Özgür Uğraş Baran. 2021. CNNFOIL: convolutional encoder decoder modeling for pressure fields around airfoils. *Neural Computing and Applications* **33**:12, 6835-6849. [Crossref]
20. Michael Leer, Andreas Kempf. 2021. Fast Flow Field Estimation for Various Applications with A Universally Applicable Machine Learning Concept. *Flow, Turbulence and Combustion* **107**:1, 175-200. [Crossref]
21. H R Attar, H Zhou, N Li. 2021. Deformation and thinning field prediction for HFQ® formed panel components using convolutional neural networks. *IOP Conference Series: Materials Science and Engineering* **1157**:1, 012079. [Crossref]

22. Qin Lin, Jiaqian Wang, Feiyan Cai, Rujun Zhang, Degang Zhao, Xiangxiang Xia, Jinping Wang, Hairong Zheng. 2021. A deep learning approach for the fast generation of acoustic holograms. *The Journal of the Acoustical Society of America* **149**:4, 2312-2322. [Crossref]

23. Qian Chen, Yue Wang, Hui Wang, Xubo Yang. 2021. Data-driven simulation in fluids animation: A survey. *Virtual Reality & Intelligent Hardware* **3**:2, 87-104. [Crossref]

24. Nils Wandel, Michael Weinmann, Reinhard Klein. 2021. Teaching the incompressible Navier–Stokes equations to fast neural surrogate models in three dimensions. *Physics of Fluids* **33**:4, 047117. [Crossref]

25. A. Haeri, K. Skonieczny. Accurate and Real-time Simulation of Rover Wheel Traction 1-9. [Crossref]

26. Zifeng Guo, João P. Leitão, Nuno E. Simões, Vahid Moosavi. 2021. Data-driven flood emulation: Speeding up urban flood predictions by deep convolutional neural networks. *Journal of Flood Risk Management* **14**:1. . [Crossref]

27. Romit Maulik, Themistoklis Botsas, Nesar Ramachandra, Lachlan R. Mason, Indranil Pan. 2021. Latent-space time evolution of non-intrusive reduced-order models using Gaussian process emulation. *Physica D: Nonlinear Phenomena* **416**, 132797. [Crossref]

28. Wenhui Peng, Yao Zhang, Michel Desmarais. Spatial Convolution Neural Network for Efficient Prediction of Aerodynamic Coefficients . [Abstract] [PDF] [PDF Plus]

29. Ashley D. Scillitoe, Pranay Seshadri, Chun Yui Wong. Instantaneous Flowfield Estimation with Gaussian Ridges . [Abstract] [PDF] [PDF Plus]

30. Sandeep Reddy Bukka, Rachit Gupta, Allan Ross Magee, Rajeev Kumar Jaiman. 2021. Assessment of unsteady flow predictions using hybrid deep learning based reduced-order models. *Physics of Fluids* **33**:1, 013601. [Crossref]

31. Qi Wang, Li Yang, Yu Rao. 2021. Establishment of a generalizable model on a small-scale dataset to predict the surface pressure distribution of gas turbine blades. *Energy* **214**, 118878. [Crossref]

32. Suprosanna Shit, Dhritiman Das, Ivan Ezhov, Johannes C. Paetzold, Augusto F. Sanches, Nils Thuerey, Bjoern H. Menze. Velocity-To-Pressure (V2P) - Net: Inferring Relative Pressures from Time-Varying 3D Fluid Flow Velocities 545-558. [Crossref]

33. JingYuan Zhu, HuiMin Ma, TianYu Hu, Jian Yuan. Frequency Transfer Model: Generating High Frequency Components for Fluid Simulation Details Reconstruction 478-489. [Crossref]

34. Oliviu Şugar-Gabor. 2020. Parameterized nonintrusive reduced-order model for general unsteady flow problems using artificial neural networks. *International Journal for Numerical Methods in Fluids* **30**. . [Crossref]

35. Romit Maulik, Bethany Lusch, Prasanna Balaprakash. 2020. Non-autoregressive time-series methods for stable parametric reduced-order models. *Physics of Fluids* **32**:8, 087115. [Crossref]

36. Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, Aparna Chandramowliswharan. CFDNet 1-12. [Crossref]

37. Hongwei Tang, Jean Rabault, Alexander Kuhnle, Yan Wang, Tongguang Wang. 2020. Robust active flow control over a range of Reynolds numbers using an artificial neural network trained through deep reinforcement learning. *Physics of Fluids* **32**:5, 053605. [Crossref]

38. Liwei Hu, Jun Zhang, Yu Xiang, Wenyong Wang. 2020. Neural Networks-Based Aerodynamic Data Modeling: A Comprehensive Review. *IEEE Access* **8**, 90805-90823. [Crossref]

39. A.V. Umanovskiy. 2020. Generative adversarial neural networks for the heuristic modelling of a two-phase flow in porous media. *Computational Continuum Mechanics* **13**:2, 231-241. [Crossref]